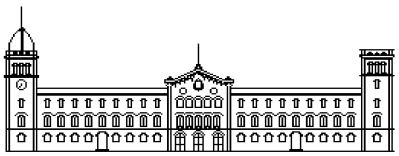


Trabajo final de grado

Grado en INGENIERÍA INFORMÁTICA

Facultad de Matemáticas
Universidad de Barcelona

Multi-clasificación discriminativa por partes mediante Códigos Correctores de Errores

---

Juan Carlos Ortega Pérez

Director: Sergio Escalera Guerrero
Miguel Ángel Bautista Martín
Realizado en: Departamento de Matemática Aplicada y Análisis. UB

Barcelona, 29 de junio de 2012

Agradecimientos.

Este proyecto no hubiera sido posible sin la ayuda y el apoyo de muchas personas. Me gustaría por lo tanto dedicárselo a ellas:

A Sergio Escalera y Miguel Ángel Bautista, que siempre han encontrado el momento para discutir y solucionar las dudas que han ido surgiendo a través de todas las etapas del proyecto, especialmente las más difíciles.

A todos mis familiares, amigos y compañeros que me han apoyado y dado ánimos en los momentos más duros, para sacar el proyecto adelante.

A Daniel Sánchez Abril por la estrecha colaboración que hemos tenido en la realización de algunas fases comunes del proyecto.

A mis compañeros Daniel Ferreiro, Gabriel Dominguez, Alberto Casellas, Pablo Ezequiel, Oriol Iniesta y David Osuna, que me han dado consejos en todo momento. A Úrsula Velasco por estar siempre a mi lado dándome ánimos y ayudándome en todo lo que ha hecho falta.

Y al resto de amigos, compañeros y profesores, que aunque no hayan formado parte de este proyecto, han sido los responsables de que me decantara por la investigación en el área de la Visión Artificial. Para finalizar, sólo decir que estos cuatro últimos años han sido una gran experiencia y que nunca los olvidaré.

Abstract.

This project aims to study the functioning and potential of different techniques in the field of artificial vision for the detection and segmentation of limbs of a person on a large volume of images. The database used contains a large number of images where different subjects are performing various poses. The objective is to detect different extremities and to perform this detection has been chosen to train different classifiers cascades on Haar features on targeted regions. Once trained, they may send images from the database to detect the aforementioned extremities. In addition to using the cascades for the detection, some methods will be used to verify if that detection is true or not. As a next step segmentation techniques are applied to achieve two different purposes: to segment only the subject from the background image and segment each limb of the subject provided to separate the subject of the picture but width his limbs detected. In this case we have chosen segmentation using Graph Cuts formulation. The methods presented have been validated on a dataset made by a research group of the department, showing satisfactory results.

Resum.

El present projecte té com a finalitat estudiar el funcionament i les possibilitats que ofereixen les diferents tècniques del camp de la visió artificial per a la detecció i segmentació d'extremitats d'una persona sobre un volum d'imatges. La base de dades emprada conté un gran nombre d'imatges on apareixen diversos subjectes realitzant diverses postures. D'aquests subjectes, es vol detectar diferents extremitats i per poder realitzar aquesta detecció, s'ha optat per entrenar diferents cascades de classificadors amb característiques Haar sobre regions orientades. Un cop entrenades, se'ls podran enviar imatges de la base de dades per detectar les extremitats ja esmentades. A més d'utilitzar les cascades per a la detecció, s'utilitzaran uns mètodes per corroborar si el que detecten és cert o no. Com a següent pas s'apliquen tècniques de segmentació per assolir dues finalitats diferents: segmentar únicament el subjecte del fons de la imatge i segmentar cada extremitat del subjecte per tal de separar aquest de la imatge, però amb les seves extremitats detectades. En aquest cas s'ha optat per segmentació mitjançant la formulació GraphCut. Els mètodes presentats s'han validat sobre un conjunt de dades realitzats per un grup de recerca del departament, mostrant resultats satisfactoris.

Resumen.

El presente proyecto tiene como finalidad estudiar el funcionamiento y las posibilidades que ofrecen las diferentes técnicas del campo de la visión artificial para la detección y segmentación de extremidades de una persona sobre un gran volumen de imágenes. La base de datos empleada contiene un gran número de imágenes donde aparecen diversos sujetos realizando varias poses. De estos sujetos, se tienen que detectar diferentes extremidades y para poder realizar esta detección se ha optado por entrenar diferentes cascadas de clasificadores utilizando características Haar sobre regiones orientadas. Una vez entrenadas, se les podrán enviar imágenes de la base de datos para detectar las extremidades ya mencionadas. Además de utilizar las cascadas para la detección, se utilizarán unos métodos para corroborar si lo que detectan es cierto o no. Como siguiente paso se aplican técnicas de segmentación para alcanzar dos finalidades diferentes: segmentar únicamente el sujeto del fondo de la imagen y segmentar cada extremidad del sujeto con tal de separar el sujeto de la imagen pero con sus extremidades detectadas. En este caso se ha optado por segmentación mediante la formulación GraphCut. Los métodos presentados se han validado sobre un conjunto de datos realizados por un grupo de investigación del departamento, mostrando resultados satisfactorios.

Índice

1. Introducción

En este capítulo se realiza la introducción del proyecto, describiendo el contexto, motivación y el planteamiento de las necesidades detectadas que dieron origen a la creación de éste. Además, se incluye un estudio del estado del arte sobre otras tecnologías/procesos/técnicas existentes en la literatura que abordan problemáticas similares

1.1 Contexto

Una de las mejores maneras de poder detectar y analizar las extremidades de las personas es mediante la grabación de secuencias de vídeo en las que aparecen una serie de personas realizando movimientos, poses, etc. La clave radica en tener una amplia variedad de éstas para asegurar una alta generablilidad y discriminabilidad en los métodos de reconocimiento.

La eficacia en la detección de las poses es esencial ya que el análisis que se realizará a posteriori depende totalmente de éstas. Ya sea corriendo, saltando, agachado, riendo, chillando, etc. Es válido cualquier tipo de pose que se pueda apreciar sin entrar en detalles.

En consecuencia, se ha obtenido de una base de datos de imágenes con un conjunto de sujetos realizando una serie de varias poses. Estos datos serán útiles para el análisis de diversas técnicas de visión por computador para reconocer las extremidades y las posiciones que ocupan según las poses. De esta manera, se podrán detectar las extremidades que nosotros queramos y estén en la imagen.

La metodología de detección de las extremidades que se aborda en este proyecto se basa en el aprendizaje por cascadas, que no es más que un conjunto de clasificadores encadenados. Para cada extremidad dispondremos de una cascada que aprenderá a reconocer dicha parte del cuerpo a partir de un gran conjunto de entrenamiento.

Además de tener las cascadas entrenadas, una forma de asegurarse de que el resultado sea el esperado, es el uso de unos métodos de corrección de errores para poder corregir una posible desviación en el resultado. Por otro lado, es también interesante no

sólo detectar extremidades sino segmentarlas del fondo de la imagen. Para este propósito se utilizarán un conjunto de métodos de segmentación de visión por computador que permitirán segmentar la persona del fondo y distinguir el conjunto de extremidades del sujeto. De esta manera, el resultado final del proceso será el sujeto separado del fondo de la imagen y además también se podrán diferenciar las extremidades de dicho sujeto a nivel de píxel.

1.2 Motivación

La motivación en este Proyecto Fin de Carrera (PFC) es por un lado el diseño, implementación e integración modular de un sistema capaz de detectar y segmentar extremidades de personas dado un gran volumen de imágenes. Para ello se partirá de varios algoritmos representativos del estado del arte para esta cuestión, que hemos considerado ventajosos en términos de eficiencia y calidad de los 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 cantidad de poses que pueden realizar los humanos, recuperar la posición del cuerpo es un problema que tiene cierta 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.

Además, diversas aplicaciones de este sistema pueden ser utilizadas en el ámbito de la medicina, por ejemplo: monitorizar las personas con discapacidad con el fin de detectar automáticamente los comportamientos de los pacientes en los hospitales, como el dolor o la agitación, extraer automáticamente un conjunto de indicadores de actitud y de comportamiento a fin de apoyar el diagnostico de enfermedades mentales, etc. En segundo lugar, intentaremos mejorar las técnicas, proponiendo una combinación idónea de las mismas para tener un sistema cuyos resultados sean lo más satisfactorios posibles.

1.3 Estado del arte

La combinación de técnicas de visión por computador utilizadas en este proyecto no son las únicas y no necesariamente las óptimas, por este motivo vamos a comentar técnicas del estado del arte parecidas a cada una de las que hemos utilizado y así explicar los motivos que nos han llevado a usarlas. Conviene aclarar que las explicaciones dadas a continuación por cada técnica serán algo breves ya que más adelante se explicarán las elegidas con más detalle.

Las extremidades que vamos a comentar a continuación vienen dadas por el orden en que se implementan y conectan tales métodos. Primero de todo se extraen de la base de datos de imágenes sus características a partir del descriptor de características, y luego, enviar estas a los clasificadores para que aprendan. Una vez los clasificadores están entrenados y se quieren probar, la forma de proceder será enviarles imágenes para validar si lo que detectan es correcto o no. Aquí es donde se utilizarán los códigos correctores de errores, para corregir los posibles errores a la hora de la clasificación. Por último, se utilizarán técnicas de segmentación para separar persona del fondo de la imagen y extremidades de dicho fondo.

1.3.1 Descriptor de características

Los descriptores visuales son herramientas muy necesarias para extraer la información que se considera importante de la imagen para que los clasificadores puedan realizar el aprendizaje. En este caso utilizamos los descriptores HOG y Haar que son dos de las características más utilizadas en el estado del arte. Los HOG son histogramas de

gradientes orientados que nos codifican la información que contienen las imágenes. Son relativamente costosos de calcular y se suelen utilizar para aprender clasificadores discriminativos. En el segundo caso, los descriptores Haar se utilizan como derivadas discretas sobre imágenes y se usan para ser aprendidos por las cascadas mediante el clasificador Adaboost.

1.3.2 Clasificadores

Un clasificador es un sistema capaz de proporcionar una predicción de pertenencia a una clase como salida a partir de un conjunto de características tomadas como entradas. En la rama de aprendizaje supervisado, construir un clasificador se basa en definir una regla que pueda asignar una etiqueta a cualquier otro dato de entrada a partir de un conjunto de ejemplos etiquetados.

Un ejemplo de un clasificador es aquel que acepta datos de sueldos de una persona, edad, estado civil, dirección e historial de crédito y clasifica a la persona como aceptables o inaceptables para recibir una nueva tarjeta de crédito o préstamo.

Una de las posibles combinaciones de un conjunto de clasificadores es la cascada y ésta dará un mejor resultado dependiendo de la cantidad de datos que se introduzcan, el número de clasificadores, etc.

La motivación para hacer uso de cascadas se debe a que éstas se usan para aprender conjunto desbalanceados de datos. Por ejemplo, en el caso de querer aprender cabeza contra cualquier otro elemento del mundo visual, incluiremos en la cascada más imágenes de lo que no es cabeza. El uso de cascadas en este contexto nos permite hacer viable el aprendizaje desbalanceado a partir de la concatenación de clasificadores semi-balanceados.

1.3.3 Códigos correctores de errores

Para tratar problemas de multi-clasificación (donde el número de clases N > 2) de objetos o partes, la mayoría de las técnicas usan votación. No obstante, los ECOC han

demostrado que además de permitir votar entre múltiples clasificadores binarios también permiten corrección.

Los códigos de corrección de errores son un marco general para combinar clasificadores binarios a fin de abordar el problema multi-clase. En base a los principios de corrección de errores de teoría de la comunicación y debido a su capacidad para corregir los errores de la varianza de los clasificadores base, ECOC se ha aplicado con éxito a una amplia gama de aplicaciones de reconocimiento de patrones, con resultados satisfactorios.

1.3.4 Graph Cuts

Los métodos de segmentación que vamos a utilizar en las diferentes imágenes de la base de datos son dos: grabcut y graphcut. Grabcut usa la teoría de saturación (algoritmo de Ford-Fulkerson), mientras que GraphCut se utiliza para la segmentación binaria y alpha-expansion, que extiende la teoría de graphcuts al contexto multi-etiqueta (en nuestro caso multi-extremidad). El primero aborda la segmentación binaria, en nuestro caso lo utilizaremos para segmentar la persona del fondo de la imagen. Por otro lado, el segundo método es utilizado para realizar segmentación multi-clase, es decir, puede segmentar varias clases previamente especificadas. Éste último lo utilizaremos para segmentar varias extremidades como pueden ser cuerpo, brazos…

1.4 La propuesta

A partir de una base de datos de imágenes donde aparecen personas realizando varias poses, se desea aplicar técnicas de visión por computador para detectar las extremidades de dichas personas. Además, se quiere segmentar la persona de diferentes maneras para obtener dos resultados: separar la persona del fondo de la imagen y segmentar cada extremidad para tener el conjunto de extremidades de la persona separadas del fondo.

Todo este trabajo se realizará primeramente entrenando una serie de clasificadores en cascada, los cuales estarán asociados a unos descriptores de características que extraerán de cada imagen las características más importantes.

Una vez los clasificadores están entrenados, a los resultados que podemos obtener de ellos al enviarles imágenes de validación, se les aplicará unos códigos para corregir los posibles errores a la hora de la clasificación. Por último, se aplicarán dos métodos de segmentación para obtener los resultados finales, comentados en el párrafo anterior.

1.5 Objetivos

Los objetivos que nos hemos marcado superar en este proyecto son los siguientes:

1. Usar descriptores visuales eficientes para extraer las características relevantes de las imágenes.

2. Entrenar diferentes clasificadores en cascada capaces de clasificar las extremidades de una persona.

3. Aplicar códigos de corrección de errores para corregir los posibles errores que los clasificadores pueden cometer

4. Utilizar la segmentación binaria y multi-clase.

1.6 Estructura de la memoria

La memoria está dividida en las partes siguientes, explicadas extensamente:

1. Descripción de la base de datos de imágenes utilizadas.

2. Explicación de los distintos clasificadores que vamos a utilizar.

3. Descripción de los distintos sistemas de extracción de características.

4. Uso de los árboles de decisión en combinación con SVM.

5. Conjunto de pruebas realizadas.

6. Resultados obtenidos.

7. Conclusiones.

2. Base de datos HuPBA

En este capítulo se va a explicar en que consiste la base de datos HuPBA, su estructura y el proceso que ha sido necesario para extraer de imágenes normales todas las extremidades de los actores en sus distintas posturas a lo largo de las imágenes que componen el video de cada actor.

2.1 Introducción

En el campo de la Visión por Computador, uno de los problemas que ha adquirido más relevancia históricamente ha sido la detección de personas y el reconocimiento de la pose. Sin embargo, la literatura no aporta grandes bases de datos en este campo (tanto a nivel de tamaño como de protocolos de validación). Por eso, en este proyecto se propone la base de datos HuPBA desarrollada en la Universidad de Barcelona.

El mundo real donde nos movemos está lleno de objetos muy diferentes pero también de otros que son muy parecidos. El objetivo de este sistema es llegar a distinguir aquello que es una persona del resto de elementos visuales. Por lo tanto, tenemos que tener información (imágenes) de personas físicas como también de todo aquello que no es una persona, como pueden ser animales, objetos, etc.

Dentro del reconocimiento de las acciones humanas y su postura, hay una gran variedad de problemas a resolver como pueden ser describir sus partes, inferir su posición, reconocer los gestos que realizan, etc. A todo esto hay que sumarle las limitaciones del dispositivo con el que obtenemos la información del mundo, problemas en el punto de vista, problemas de iluminación, deformaciones… Por lo tanto, este es un campo donde existe una gran dificultad en el problema que se plantea y en el cual se investigan nuevas técnicas de Visión Artificial.

Los datos de las personas nos vienen dados por una base de datos desarrollada en la Universidad de Barcelona, llamada HuPBA (Human Pose and Behaviour Analisis). En nuestro caso, hemos seleccionado un subconjunto formado por 9 personas que han sido grabadas mediante una cámara digital, en forma de vídeo estando en distintas posturas y

realizando distintos gestos. El proceso para tratar estos videos ha sido transformar éste en una sucesión de imágenes que posteriormente se tratarán para detectar sus partes o extremidades.

Posteriormente, para cada imagen (frame) de cada secuencia, se han etiquetado todas sus extremidades. Este proceso de etiquetaje se realiza mediante el uso de máscaras. Una máscara es una imagen en negro (ceros) de las mismas dimensiones que la imagen original pero que tienen una región blanca (unos) que indica donde se encuentra la extremidad que nos interesa. Por lo tanto, para cada secuencia, tenemos 14 directorios cada uno conteniendo máscaras para cada frame que indican donde se encuentran las extremidades del actor/actores en un determinado momento.

Estas regiones previamente definidas son las que utilizaremos para entrenar nuestros sistemas clasificadores, por lo que hace falta procesar toda la base de datos con sus máscaras e ir extrayendo todas las partes para posteriormente procesarlas.

La creación de esta base de datos fue realizada como proyecto final de carrera del Sr. Jordi Suñé Fontanals en 2011. El hecho de etiquetar tantas imágenes es una operación muy costosa, por lo que se intentó simplificar un poco mediante el uso de la interpolación de la posición de la extremidad. Esta técnica se basa en que se sabe que entre frame y frame la posición de las extremidades varía mínimamente. Teniendo en cuenta esta premisa, las extremidades se etiquetaron cada 2 frames, y mediante el uso de un programa en Matlab, se pudo completar la posición de las extremidades en los frames intermedios que no estaban etiquetados. En la Fig. 1 y 2 se muestra un ejemplo del contenido de la base de datos.

Figura 1: Imagen original

Figura 2: Imagen de máscara asociada

Nuestra función ha sido procesar toda la base de datos, compuesta por 288827 imágenes y 4.5 Gb de capacidad aproximadamente. Hemos procesado y recortado la totalidad de las extremidades de todos los actores. Las partes del cuerpo que ha sido etiquetadas son las siguientes:

1. Cabeza	8. Brazo Derecho
2. Torso	9. Pie Izquierdo
3. Mano Izquierda	10. Pie Derecho
4. Mano Derecha	11. Pierna Izquierda
5. Antebrazo Izquierdo	12. Pierna Derecha
6. Antebrazo Derecho	13. Muslo Izquierdo
7. Brazo Izquierdo	14. Muslo Derecho

Cuadro 1: Conjunto total de extremidades

2.2 Obtención de las imágenes de fondo

Una vez tenemos toda la información referente a la posición de las extremidades de las personas, nos hace falta incorporar a nuestro sistema todas aquellas imágenes que no son persona. El mundo real es un conjunto inmenso de elementos distintos, y que nuestros clasificadores tienen que aprender para distinguir las extremidades del fondo de una imagen, y este fondo puede ser cualquier imagen.

Para solucionar este problema, construimos una pequeña aplicación en Python, que se encargaba de hacer la tarea de crawler. Un crawler, o también llamada comúnmente “araña de internet” se encarga de inspeccionar un sitio web y de ir saltando a diferentes páginas mediante los hipervínculos que encuentra en la página actual, para seguir realizando un rastreo de la web de forma automática.

En nuestro caso, hemos programado un crawler que se encarga de obtener imágenes de la página web de Flickr. Mediante la creación de una cuenta de Flickr, podemos tener acceso a un sinfín de imágenes que abarcan todo tipo de temas, desde construcciones, coches inmuebles hasta animales, paisajes, objetos, etc. Todas estas imágenes nos servirán como conjunto de negativos y se consideran el fondo que los clasificadores tienen que aprender a distinguir de las extremidades.

Propusimos un filtro lo bastante general para obtener imágenes de todo tipo y a partir de aquí, fuimos descartando todas aquellas que contenían personas y niños, ya que esta información ya ha sido procesada anteriormente. En total, obtuvimos unas 2.500 imágenes finalmente filtradas de cualquier persona u extremidad. Este conjunto de imágenes los utilizaremos como conjunto de negativos, mientras que las imágenes de las extremidades serán el conjunto de positivos. Mediante diferentes técnicas que se explicarán más adelante, podremos conseguir que nuestro sistema aprenda a distinguir una extremidad del fondo de la imagen.

En la Fig. 3 se muestran algunas imágenes extraídas de Flickr y que han sido utilizadas para entrenar los clasificadores, para así detectar el fondo de la imagen.

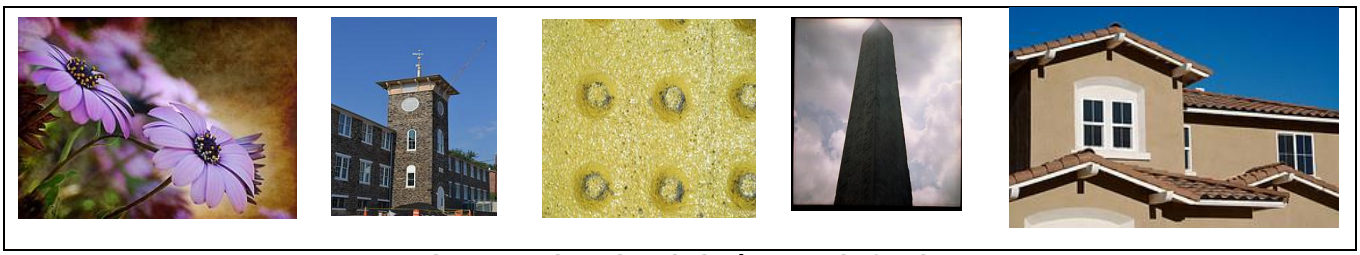


Figura 3: Ejemplos de imágenes de fondo

2.3 El problema de la orientación de las extremidades

Llegados a este punto, se nos presenta un problema bastante importante y que consiste en la orientación de las extremidades. Nuestra primera intuición nos diría que para obtener las extremidades simplemente tendríamos que realizar una operación AND entre la imagen original y la máscara, de tal forma que nos quedaría una imagen con la extremidad en cuestión. Pero esto no es del todo cierto, ya que las personas que hay en las imágenes están en distintas posturas y las posiciones de sus extremidades son diferentes. Esto es un problema, ya que pueden haber muchísimas combinaciones de orientación para una extremidad, lo que nos complica mucho más el proceso de aprendizaje, al no haber un patrón homogéneo.

Este problema ha sido solucionado mediante el análisis de un código en Matlab que obtiene la orientación predominante de una imagen. El funcionamiento consiste en analizar la región mediante las derivadas gausianas en X e Y. De esta manera se obtienen dos

imágenes que nos indican el grado de cambio de intensidad dentro de la imagen. El paso siguiente es crear un histograma con todas las orientaciones obtenidas y quedarnos con la mayor. Este grado será el que nos indique cuántos grados tenemos que rotar la imagen.

Este proceso se realiza para tener todas las imágenes con una misma orientación, y de esta manera facilitar el proceso de entrenamiento de los clasificadores. Además, podemos evitar la confusión entre partes que se parecen mucho como son pies y las piernas.

El código estaba montado en Matlab, pero dado la gran cantidad de imágenes y extremidades que hay que procesar, al final decidimos pasar ese código a C/C++ mediante el uso de las librerías OpenCV y de algunas funciones ya implementadas. Después de algunas optimizaciones, el tiempo de ejecución por extremidad mejoró radicalmente. Esta mejora de velocidad hizo que un proceso que tarda aproximadamente una semana en Matlab, se transforme en horas.

Lenguaje	**Tiempo de Ejecución por Extremidad**
Matlab	2.5 Segundos
C/C++	0.003 Segundos

Cuadro 2: Comparativa de tiempo de ejecución en Matlab y C/C++

2.4 Propósito de los datos

La parte de investigación de este PFC está enfocada en el análisis de los clasificadores en cascada (Cascadas AdaBoost) y SVM (Support Vector Machine). Un clasificador no es más que un sistema, el cuál dará un resultado o predicción en función de su conocimiento y la información de entrada. Para realizar este proceso, hemos tenido que separar dos conjuntos de información:

1. Imágenes de Training (imágenes que se usarán en el entrenamiento del clasificador)
2. Imágenes de Test (Imágenes que se usarán para comprobar la validez del clasificador)

Por lo tanto, se ha realizado una organización de los datos. De todo el conjunto de imágenes de la base de datos, se han separado un conjunto para la fase de training de los clasificadores y otra para la parte de test de estos mismos. Por lo tanto, los datos de training no se pueden utilizar en la fase de test, ya que entonces nuestro sistema no serviría de nada. Siempre acertaría en la predicción ya que las imágenes de prueba forman ya parte del conocimiento de ese clasificador.

2.5 Procesamiento de los datos de entrada

Para cada actor, primero procesamos sus extremidades. En cada imagen o frame, seleccionamos la máscara asociada a esta y procedemos a obtener la información de la región que nos indica la máscara.

El proceso consiste en primero rotar y luego cortar, ya que si se realiza al revés, nos queda una imagen de la extremidad con bordes negros. Además, hay que sumarle a todo esto que las imágenes o regiones no son cuadradas. Por ejemplo, si tenemos una pierna etiquetada, su máscara tendrá una forma de rectángulo. Por lo tanto hay que obtener una región cuadrada de la misma longitud de lado, para tener todas las extremidades homogéneas.

El paso siguiente es analizar esa región con el algoritmo que calcula la orientación predominante y obtener el grado por el que hay que rotar la imagen. A continuación, tenemos que rotar la imagen de entrada, pero con el centro de rotación situado en el centro de la máscara. Finalmente realizamos el recorte de la imagen resultante, pero sin cambiar la posición de la nueva máscara cuadrada.

El resultado de este proceso nos devuelve todas las extremidades en imágenes cuadradas y que ya están rotadas. Estas imágenes serán las que utilizaremos para el primer paso del PFC, que será el entrenamiento de unas cascadas Adaboost con todas las imágenes de cada extremidad como positivos y las imágenes de fondo como negativos. El algoritmo es el siguiente:

1. Obtenemos la imagen actual original de la persona.
2. Para cada extremidad, obtener su máscara correspondiente a la imagen original.

3. Convertir la máscara a forma cuadrada.
4. Realizar un primer recorte de la imagen correspondiente a la máscara.
5. Calcular la orientación predominante de esa región.
6. Con el grado obtenido anteriormente, rotar la imagen original como punto de rotación central que sea el centro de la máscara original cuadrada.
7. Realizar un segundo recorte de la imagen original rotada y la mascara cuadrada sin rotar.
8. Guardar la imagen de la extremidad recortada en su carpeta correspondiente.

El hecho de optimizar todo este proceso en C/C++ fue una gran ventaja, ya que todas estas operaciones necesitaban de mucho tiempo en Matlab. El punto crítico lo tuvimos cuanto hicimos la conversión del código de obtención de la orientación predominante, ya que al estar montado en Matlab, no lo podíamos utilizar directamente. En la Fig. 4 se muestra el funcionamiento por bloques de todo el sistema de una manera más detallada.

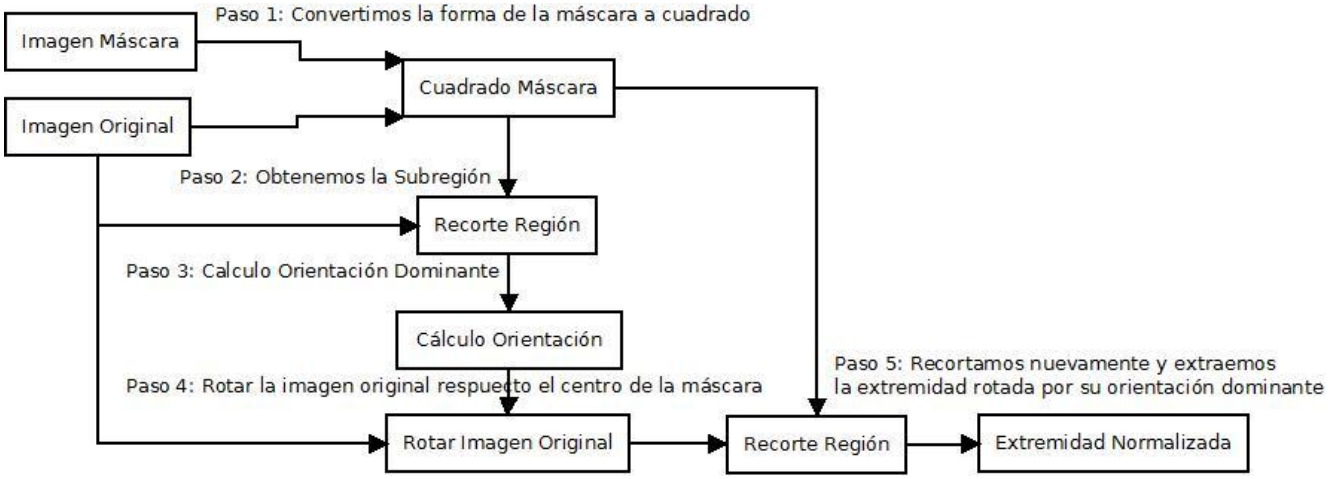


Figura 4: Proceso de extracción de las extremidades

2.6 Elección del número de extremidades

En nuestra implementación, hemos utilizado 6 extremidades. Hemos unido para cada extremidad sus componentes izquierda y derecha, en caso de tener para simplificar aún más y utilizar menos clasificadores. Ha habido componentes como las manos o los pies, que no hemos tenido en cuenta, ya que teníamos el problema de que el resultado recortado tenía unas dimensiones muy pequeñas. Esto obligaba a re-escalar la imagen, con la pérdida

consiguiente de calidad. Finalmente, nos decidimos por incluir las 6 extremidades más importantes, y que son las siguientes:

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

Cuadro 3: Extremidades seleccionadas

El siguiente paso una vez tenemos todas las imágenes de las extremidades recortadas y procesadas es el entrenamiento de estas mediante las técnicas de Adaboost y SVM que a continuación explicaremos detalladamente.

3. Método

En este apartado se realizará una descripción detallada de cada método utilizado. Esto incluye tanto explicaciones teóricas, formulaciones como diagramas en los que se muestra el flujo de ejecución de las utilidades programadas. Primero se explicará el funcionamiento y las bases de las clasificadores en cascada Adaboost, a continuación se explicará el funcionamiento de los clasificadores SVM, a continuación el proceso de extracción de las características de las imágenes mediante HOG y finalmente el uso del ECOC Tree para crear un árbol de decisión.

3.1 Cascadas Adaboost

En este punto se va a describir el funcionamiento del algoritmo de clasificador basado en una cascada de clasificadores débiles, el uso de las características Haar, los conjuntos de datos de entrenamiento y algunos factores que pueden afectar en el funcionamiento del propio clasificador.

3.1.1 Explicación inicial

Nuestro primer paso ha sido la utilización de varias cascadas Adaboost para detectar las distintas extremidades. En nuestro proyecto utilizamos el detector de Viola-Jones, conocido popularmente para la detección de rostros y caras en tiempo real. Podemos utilizar dos formas distintas de clasificación:

- Utilizar un único clasificador con mucho conocimiento.
- Utilizar varios clasificadores con poco conocimiento.

En nuestro primer caso, tendríamos un clasificador con mucho conocimiento, el cual sería capaz de detectar todas las extremidades, pero el problema que nos surge es que tardaría muchísimo tiempo en aprender todas las características. Por otro lado, el segundo caso sería utilizar una cascada o pipeline de clasificadores débiles, los cuales se han entrenado a cada nivel para ir discriminando en cada etapa la entrada. De estas dos

situaciones, se ha demostrado que es más eficiente combinar *n clasificadores débiles* que un único clasificador experto.

3.1.2 Características Haar

El detector se basa en tres grandes aportaciones: la utilización de la imagen integral para obtener una rápida evaluación de las características, un algoritmo que aprenderá las características para llevar a cabo la detección y un clasificador en cascada que descartará las partes de la imagen que no sean los objetos a estudiar, para centrarse en los candidatos a serlo.

Nuestro primer paso en esta etapa es describir la imagen de entrada para obtener patrones que sean parecidos para cada extremidad. Para describir una imagen o región, el detector de Viola-Jones utiliza un conjunto de características simples para llevar a cabo la detección. El hecho de utilizar características simples y no utilizar información sobre los píxeles, hace que el funcionamiento del sistema sea muy rápido. Estas características están formadas con regiones en forma de rectángulo que se aplican a un conjunto de píxeles de la imagen de entrada. Por lo tanto, una característica está formada por un conjunto de regiones blancas y negras, 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 por cada parte. Por lo tanto, a partir de los rectángulos podemos extraer las características de la imagen como evaluaciones de la intensidad del conjunto de píxeles. La suma de la luminancia de los píxeles en la región blanca se resta con la misma suma de la región oscura. El valor obtenido mediante esta diferencia será el valor de la característica y se puede combinar con otros formando hipótesis de las regiones de una imagen.

Las características de estos rectángulos pueden ser rápidamente calculadas utilizando una representación intermedia de la imagen, llamada imagen integral. Viola-Jones construyen esta imagen tomando la suma de los valores de luminancia de los píxeles que se encuentra por encima y a la izquierda de un punto de la imagen. En la Fig. 5 se muestran ejemplos de características Haar que se utilizan en la fase de entrenamiento de las cascadas Adaboost.

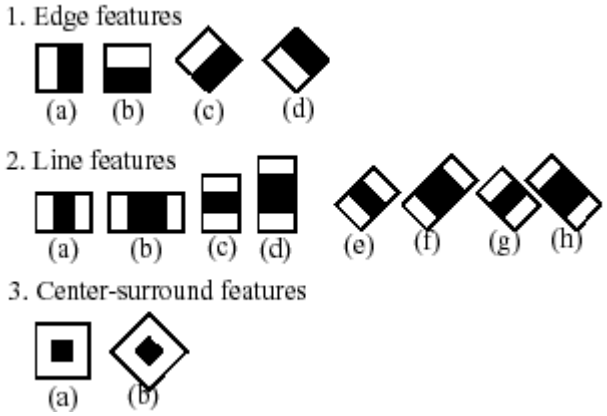


Figura 5: Distintas características Haar

3.1.3 Cálculo de la imagen integral

Como se ha comentado antes, estas características se calculan en base a la imagen integral procesada anteriormente Un ejemplo del cálculo de la imagen integral es el que se muestra en la Fig. 6:

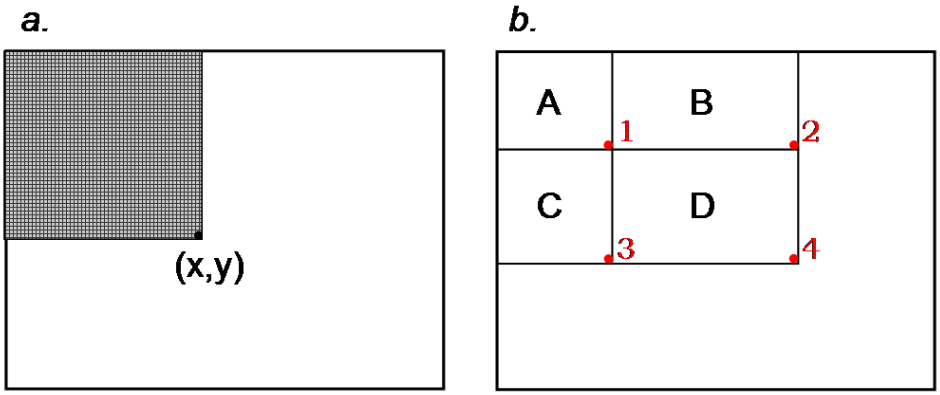


Figura 6: Imagen integral

La fórmula que corresponde al cálculo de la imagen integral tiene en cuenta la posición (x, y) donde se está analizando. Básicamente es la suma de la intensidad de los píxeles de las posiciones (a, b) hasta llegar a la posición (x, y):

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

[1]

El resultado de este cálculo es una imagen en escala de grises y su generación es muy rápida, ya que se puede realizar en tiempo lineal. Como las operaciones que hay que realizar son simplemente sumas y restas de valores enteros en regiones, la ejecución es muy eficiente, lo que hace que se pueda describir cualquier región en tiempo real.

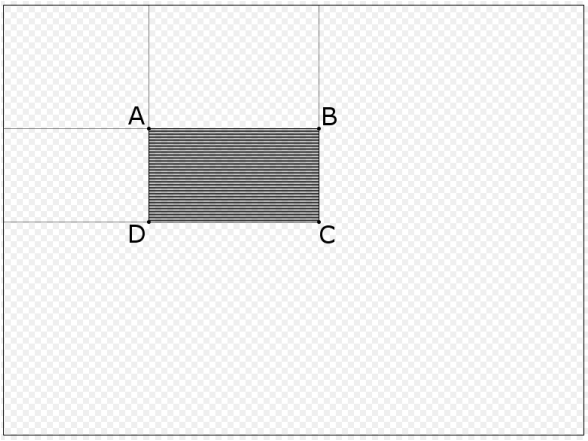


Figura 7: Imagen integral

El cálculo de la área oscura de la Fig. 7 sería la suma de el área de A más la suma de el área C menos la suma de el área B menos la suma de el área D. Por lo tanto, equivaldría a la siguiente expresión:

$$\sum_{\substack{A(x)<x'\leq C(x)\\ A(y)<y'\leq C(y)}} i(x',y') = \mathrm{sum}(A) + \mathrm{sum}(C) - \mathrm{sum}(B) - \mathrm{sum}(D). \quad [2]$$

Este proceso se realiza para todo el conjunto de imágenes de entrenamiento que contienen el objeto a detector, de forma que se puedan encontrar de forma más exacta todas las características del objeto a detectar. Como se puede ver, este proceso suma, independientemente de la imagen o región, todas las intensidades de los pixeles en tiempo constante. Hay que aclarar que primero se aplican a la imagen de entrada varias características y que estas se redimensionan automáticamente a las dimensiones de la imagen de entrada.

Estoy es muy útil para detectar cambios o patrones en una imagen de entrada, ya que si la resta de la suma de las regiones positivas y las regiones negativas es diferente de 0, significa que la imagen no es constante y que la característica Haar reacciona a un patrón.

En la Fig 8 se puede ver la manera de detectar varias partes de un rostro mediante estas características. Las dos características nos servirían para detectar los ojos de la persona, ya que la primera comprueba la diferencia entre la región de las cejas y los ojos, mientras que la segunda comprueba la diferencia entre la región de los ojos y la nariz:

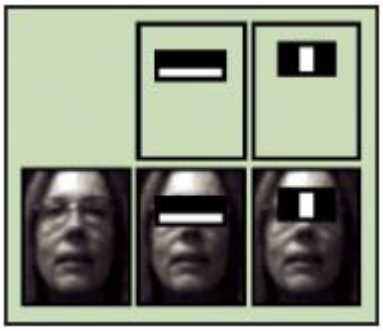


Figura 8: Detección de patrones

En definitiva, este es el patrón que se usa para extraer y describir una imagen de entrada con el método de Viola-Jones. Una vez tenemos descrita la imagen con muchas características HAAR distintas, el siguiente paso a realizar el algoritmo que aprenderá las características para realizar la detección.

3.1.4 Aprendizaje de las características

Adaboost es un algoritmo de aprendizaje automático inventado por Yoav Freund y Robert Shapire. Para entenderlo mejor se va a detallar el algoritmo propuesto por Viola-Jones, los cuales implementaron un detector de caras basado en Adaboost.

La detección de rostros propuesto por Viola-Jones es muy utilizada debido a su robustez y alta velocidad. Utiliza detectores que se basan en una cascada de clasificadores que es explorada por toda la imagen en múltiples escalas y localizaciones. Cada etapa de la cascada se basa en el uso de simples características Haar (eficientemente computadas utilizando la imagen integral) seleccionadas y combinadas mediante AdaBoost durante el entrenamiento. El funcionamiento de una cascada de clasificadores se basa en que inicialmente tenemos un conjunto de clasificadores débiles que son aplicados en distintos niveles o etapas, donde la entrada del conjunto de ejemplos positivos y negativos depende directamente de los resultados de clasificación del anterior nivel, consiguiendo de esta

manera que los clasificadores que componen la cascada se enfoquen en los ejemplos que el clasificador previo no ha conseguido aprender. La eficiencia de este esquema reside en el hecho de que los negativos (la inmensa mayoría de las ventanas a explorar) van siendo eliminados progresivamente, de forma que las primeras etapas eliminan a un gran número de ellos (los más fáciles) con muy poco procesado. Esto permite que las etapas finales tengan tiempo suficiente para encargarse de clasificar correctamente los casos más difíciles.

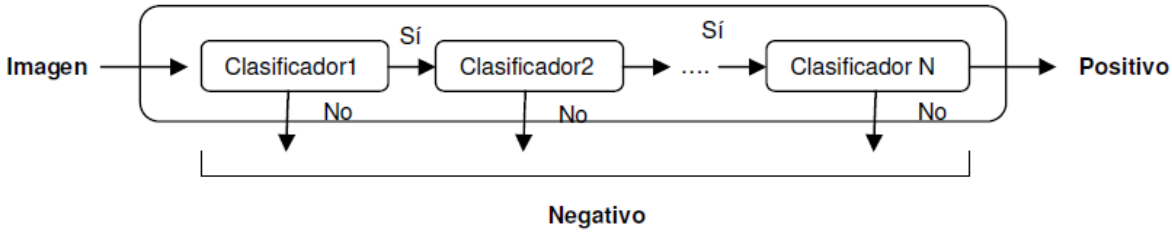


Figura 9: Cascada de detectores, diseñada por Viola-Jones

Este método se basa en una cascada de clasificadores que clasificaran la subregión como objeto o no objeto. Si no es objeto, simplemente se descartará y se pasará a la siguiente subregión. En cambio, se clasificará como objeto detectado si esta subregión pasa por todas las etapas o los clasificadores, es decir, que todos los clasificadores la considerarán el objeto a detectar.

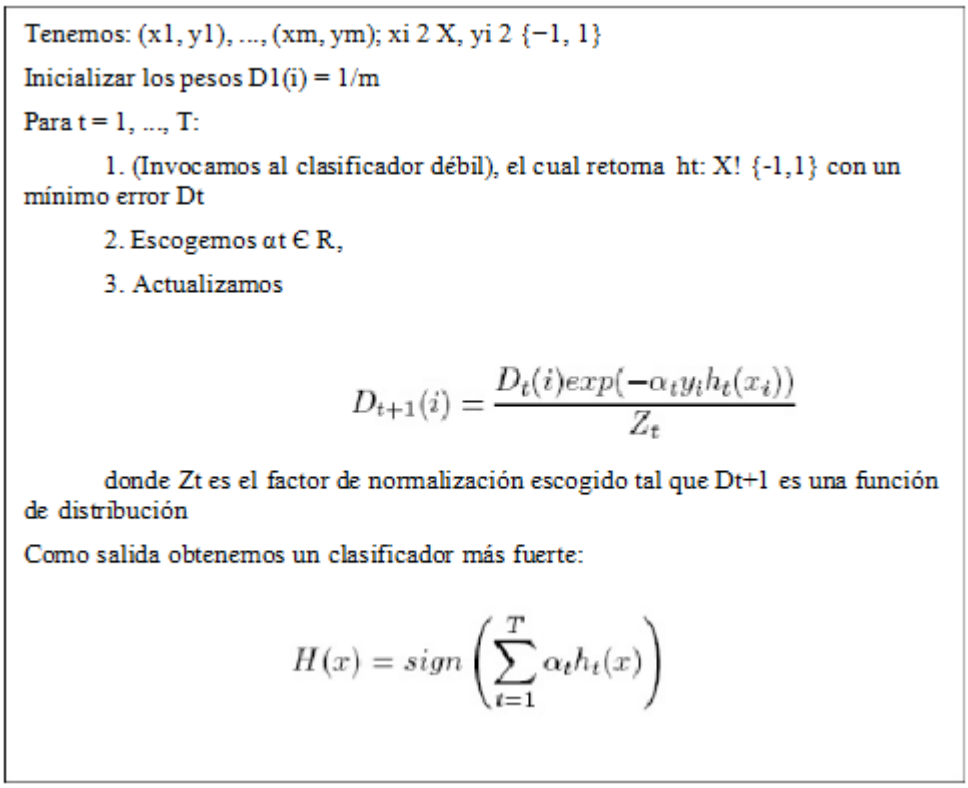
Las primeras etapas están diseñadas para realizar una separación entre los datos positivos y los negativos (el conocimiento del clasificador de nivel n) muy simple. Estos primeros clasificadores son los más rápidos y descartan la entrada en el caso de que la entrada no coincida con la descripción del objeto. De esta manera se consigue una arquitectura muy rápida en tiempo de testeo y con un rendimiento muy alto.

3.1.5 Algoritmo Adaboost

Adaboost es un algoritmo utilizado para construir clasificadores sólidos utilizando la combinación lineal de clasificadores simples o débiles. El primer paso consiste en generar los ejemplos, y a estos se les asigna el mismo peso (1/m, donde m es el número de ejemplos). A medida que se genera un nuevo modelo, se cambian los pesos de los nuevos ejemplos utilizados para el siguiente clasificador. El objetivo consiste en minimizar en cada

iteración el error esperado. Por lo tanto, se asignan pesos superiores a los ejemplos mal clasificados. Con este algoritmo se pretende crear modelos que se vuelvan expertos en datos que no pudieron ser explicados por los modelos anteriores.

Después de cada iteración, los pesos van reflejando la medida en que las instancias han seguido mal clasificadas por los clasificadores que se tienen entrenados hasta el momento. Se generan igual *T* clasificadores de muestras de ejemplos pesados. El clasificador final se construye utilizando un esquema de votación pesado que depende del trabajo de cada clasificador en su conjunto de entrenamiento.


Tenemos: (x1, y1), ..., (xm, ym); xi 2 X, yi 2 {−1, 1}

Inicializar los pesos D1(i) = 1/m

Para t = 1, ..., T:

1. (Invocamos al clasificador débil), el cual retorna ht: X! {-1,1} con un mínimo error Dt

2. Escogemos αt Є R,

3. Actualizamos

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

donde Zt es el factor de normalización escogido tal que Dt+1 es una función de distribución

Como salida obtenemos un clasificador más fuerte:

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

Figura 10: Algoritmo Adaboost

Inicialmente tenemos un conjunto o set de entrenamiento definido como:

(x1, y1), …, (xm, ym) [3]

Donde para cada X_i pertenece a una clase o espacio ***X*** y cada etiqueta Y_i pertenece a un set de etiquetas ***Y***. Para simplificar la explicación se supone que los identificadores de las clases pueden ser +1 o -1, por lo tanto:

Y = {+1, -1} [4]

El clasificador Adaboost se genera a partir de un algoritmo de aprendizaje estadístico en ***t = 1, …, T*** ciclos. Una de las principales ideas del algoritmo es mantener un conjunto de pesos sobre el set de entrenamiento, que se irán actualizando en cada iteración del algoritmo. El peso perteneciente a este conjunto de pesos, sobre el ejemplo ***i*** en el ciclo ***t*** se llama $D_t(i)$.

En el comienzo de todo el proceso se asignan a todos los conjuntos unos pesos equivalentes, que inicialmente tendrá valor ***1 / m***, donde ***m*** es el numero total de elementos que hay en el set de entrenamiento. En cada ciclo o iteración, los pesos de todos aquellos ejemplos mal clasificados aumentarán, para que el próximo clasificador débil esté forzado a enfocarse en los ejemplos difíciles del conjunto de datos de entrenamiento que intuitivamente se ubican más cerca de la frontera de clasificación. Es por este motivo por el cual el algoritmo Adaboost viene de las palabras Adaptative Boosting (Boosting Adaptativo), ya que el clasificador se va adaptando al conjunto de datos de entrenamiento.

La tarea del clasificador débil es encontrar una hipótesis débil que seleccione para cada conjunto D_t aquella solución que tenga un mínimo error. Se debe tener en cuenta que este error se obtiene con respecto al conjunto D_t en el cual el clasificador débil fue entrenado y además que el error se calcula con respecto a los ejemplos mal clasificados y es la suma de estos. Finalmente la respuesta del clasificador débil viene dada por:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) \quad [5]$$

Una vez la hipótesis débil h_t está completa, Adaboost elige el parámetro alpha, midiendo la importancia que se asigna a este clasificador débil. A continuación, la distribución D_t es actualizada usando la regla siguiente:

$$D_{t+1}(i) = \frac{D_t(i) exp(-\alpha_t y_i h_t(x_i))}{Z_t} \quad [6]$$

El efecto de esta regla es incrementar los pesos de los ejemplos mal clasificados por h_t y disminuir el peso de los ejemplos correctamente clasificados. De esta manera, el peso

tiende a concentrarse 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 alpha es la importancia asignada al clasificador h_t. Como se puede entender, el clasificador devolverá una salida binaria donde nos indicará a la partición o la clase a la que pertenece el ejemplo de entrada.

3.1.6 Factores que afectan al clasificador

En la Fig. 11 se puede ver claramente la relación que hay entre cada cascada de Adaboost y los clasificadores débiles. Este diagrama representa una cascada de clasificadores Adaboost. Esta cascada por si sola no alcanza mejores resultados que un único clasificador Adaboost debido solamente a la arquitectura. Para obtener mejores resultados, es muy importante la forma en que se monta la cascada. Esta forma de entrenar llamado Boostraping, consiste en una buena forma de obtener los mejores ejemplos negativos de lo que se desea clasificar.

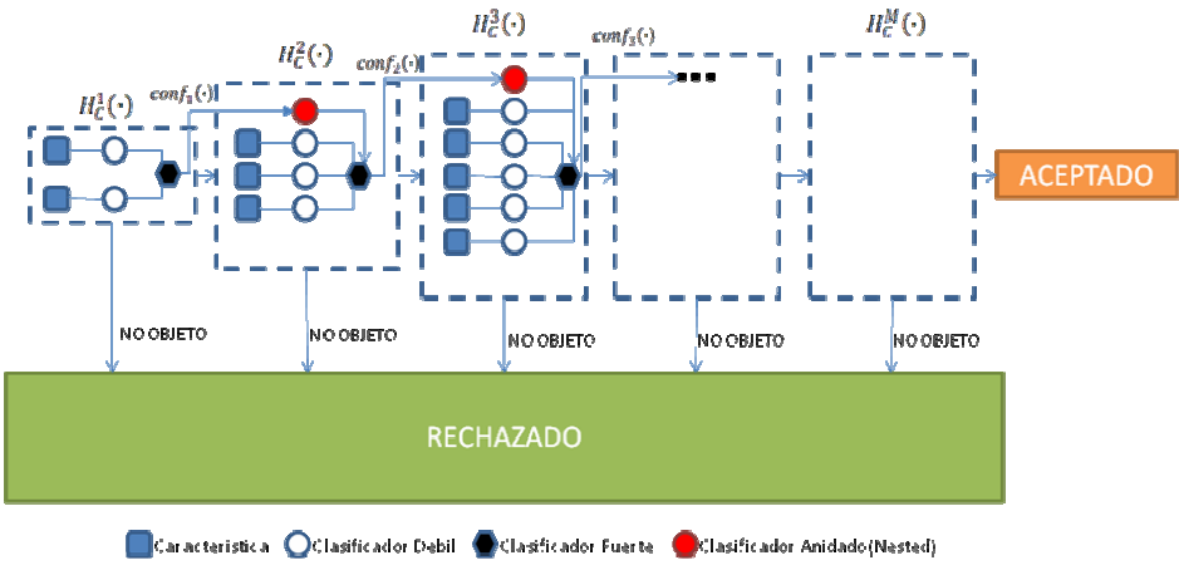


Figura 11: Diagrama de un Clasificador Adaboost en Cascada

Por ejemplo, si se está detectando un objeto concreto, cada parte de una imagen donde no se encuentra este objeto es un posible candidato a ejemplo negativo. Sin embargo, lo que se desea siempre es encontrar los ejemplos negativos más difíciles, es decir, que más se parezcan a la clase positiva. Entonces el Boostraping corresponde a entrenar cada etapa iterativamente adhiriendo todos los ejemplos clasificados de forma

errónea a la base de entrenamiento. De esta manera, el clasificador se entrena con ejemplos cada vez más difíciles. Con esto se incrementa la calidad del conjunto de entrenamiento en cada iteración.

También hay otro parámetro importante, que es el número de etapas o de clasificadores débiles conectados en cascada. Si utilizamos muy pocas etapas, corremos el riesgo de que puedan pasar ejemplos negativos como positivos y tengamos problemas de detección. En este caso, el proceso de filtraje que se realiza en todas las etapas no sería suficiente para distinguir claramente ante cualquier ejemplo de entrada, si este pertenece a una clase o a otra. Por lo tanto, tendríamos una cascada que ha aprendido poco.

En otro caso, también tenemos el problema de utilizar demasiadas etapas de clasificadores débiles. Aquí, lo que nos ocurriría es que estaríamos aprendiendo demasiado bien el objeto, lo que implicaría que sólo detectaría aquellos que fueran exactamente igual a los conjuntos de entrada, cosa que tampoco nos funcionaría. Estaríamos sobre-aprendiendo todo el conjunto de datos y entonces solo seríamos capaces de detectar aquellos ejemplos casi igual de parecidos que los conjuntos de entrenamiento.

Todo esto lo veremos más adelante, en el apartado de resultados, en el cual se podrá observar el cambio que existe en el número de etapas y el resultado obtenido. Principalmente el mayor problema que se nos aparece cuando usamos muchas etapas es el gran tiempo de procesamiento que se requiere, ya que este proceso de aprendizaje se realiza de manera exponencial.

Mediante las utilidades que incorpora OpenCV podremos entrenar varias cascadas y comprobar el funcionamiento.

3.2 Clasificadores SVM

En este punto se va a describir el funcionamiento del algoritmo de clasificador SVM, el uso del vector de características, los conjuntos de datos de entrenamiento y algunos factores que pueden afectar en el funcionamiento del propio clasificador.

3.2.1 Explicación inicial

El clasificador SVM, llamado Support Vector Machines (Máquinas de Soporte Vectorial) consiste en un sistema, que al igual que las cascadas Haar Adaboost, nos permite realizar una separación entre unos datos para poder predecir resultados. Está compuesta por una serie de algoritmos que se encargan de encontrar una separación entre todo el conjunto de datos.

En este caso, también tendremos un conjunto de datos de training o aprendizaje y otro conjunto de datos de testeo. En este caso, el funcionamiento de los clasificadores SVM es bastante distinto a las cascadas estudiadas anteriormente. Mientras que Adaboost teníamos un conjunto de clasificadores débiles conectados en cascada, ahora tenemos un único clasificador, que realiza la predicción de una información de entrada en función del modelo que tiene aprendido.

Los clasificadores SVM son clasificadores derivados de la teoría de aprendizaje estadístico, postulada por Vapnik y Chervonenkis. El primer prototipo de clasificadores SVM fue presentado en 1992 y desde ese momento fueron muy populares y obtuvieron fama, ya que los resultados que se obtenían eran muy superiores a la tecnología que había en ese momento, basada en redes neuronales, para el aprendizaje de letra manuscrita, utilizando como entrada un conjunto de píxeles que forman el carácter a detectar.

La información de aprendizaje consiste en un conjunto de datos de entrenamiento que se dividen en distintos tipos o clases y otro conjunto que serán los datos destinados a la comprobación del correcto funcionamiento del clasificador. Tanto en la fase de aprendizaje como en la de testeo a cada clase, se le asigna una etiqueta que será su identificador del tipo. En función de ese identificador, se podrá distinguir la clase a la que pertenece el dato de entrada.

3.2.2 Funcionamiento

Como se ha comentado antes, los clasificadores SVM aprenden un modelo, sobre el cual realizarán las predicciones. Este modelo se genera en la fase de entrenamiento y consiste en un espacio multidimensional donde para cada unidad de información, se descompone en un descriptor, donde cada una de sus componentes pertenecerá a un espacio distinto, dando como resultado la descripción del objeto como un punto en este espacio n-dimensional.

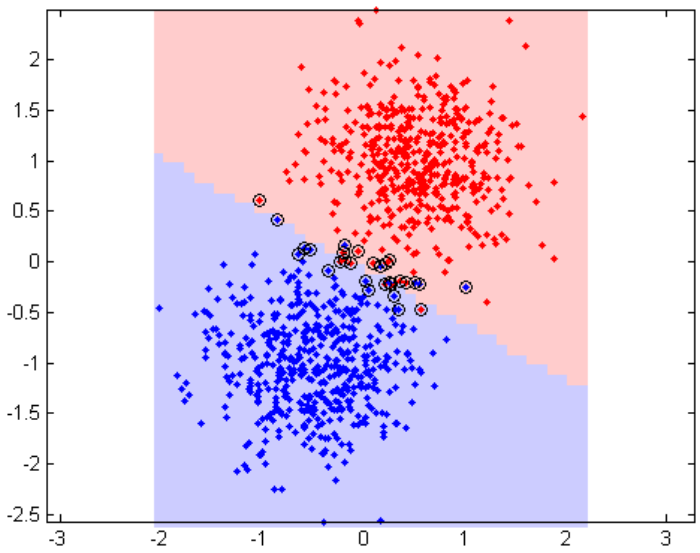


Figura 12: Representación del modelo

Como se puede ver en la Ilustración 10, cada objeto se traduce en un punto de este espacio en el cual, si tenemos muchos ejemplos de cada clase, se formarán cúmulos de puntos bien diferenciados, y por lo tanto se pueden distinguir claramente donde están las clases.

Pero una vez que tenemos estos datos insertados en el modelo, hay que tratarlos de alguna forma. El siguiente paso que realiza un clasificador SVM es hacer una separación inicial entre los dos. Esta separación se realiza mediante el cálculo del hiperplano óptimo. Básicamente un plano nos separa entre dos conjuntos en una única dimensión, mientras que un hiperplano realiza la misma operación de separación pero en varias dimensiones.

Se ha comentado que el modelo trabaja con muchísimas dimensiones, por lo que la estrategia consiste en hacer tantos hiperplanos como haga falta en el conjunto de datos para separar de la manera más optima todo el conjunto de datos.

En el siguiente ejemplo, mostrado en la Fig. 13, observamos que la clasificación se realiza en dos dimensiones, entre los ejes x e y. Por lo tanto, en este caso, el algoritmo SVM trata de encontrar el hiperplano que une los conjuntos del mismo tipo y por lo tanto, define si un elemento de entrada pertenece a uno u otro conjunto. Hay que tener en cuenta, que dado un conjunto n-dimensional, hay infinitos hiperplanos posibles que separan los datos claramente. Pero el truco consiste en separarlos de la manera más óptima. La mejor solución es la que permite el margen más amplio o máximo entre los elementos de ambas categorías.

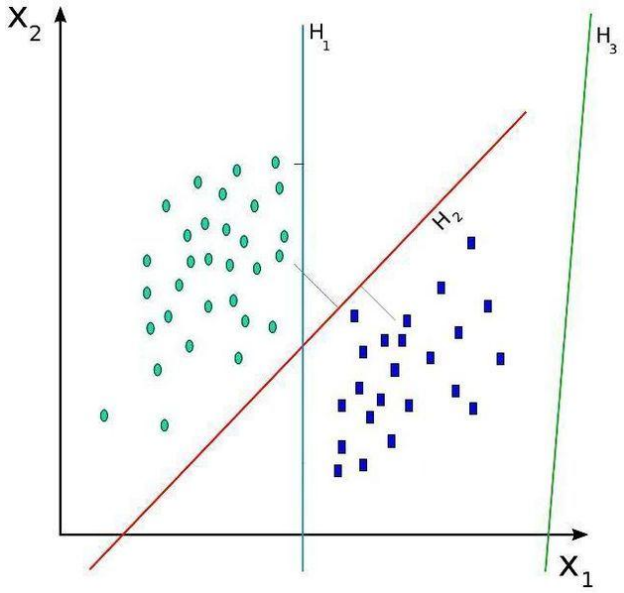


Figura 13: Separación mediante hiperplanos

Es decir, esta separación se realizará de tal manera que haya la máxima distancia posible entre cada conjunto de datos o nube. Esto es así ya que en el momento del testeo de nuevos objetos, la descripción de estos se transformará en un punto dentro del modelo. Por lo tanto, si el hiperplano se desvía hacia un conjunto de datos más que el otro, las probabilidades de que sea de una clase u otra varían. Por lo tanto, la línea del hiperplano se trazará sobre aquella zona donde la distancia entre ambas clases sea máxima.

Hay que tener en cuenta que esta separación no siempre se realiza de la mejor manera. Idealmente este sistema produce un hiperplano que separa completamente los datos del universo estudiado en dos categorías. Sin embargo, esta separación perfecta entre los conjuntos no siempre se puede realizar, por lo que se dispone de unos parámetros de ajuste llamados overfitting que más adelante comentaremos.

3.2.3 Separación entre los datos

Como se ha comentado anteriormente, la separación de los datos se realiza mediante un hiperplano. Pero hay veces en que no se puede separar mediante una simple recta todo el conjunto de datos, ya sea por su posición espacial, o su representación dentro del espacio N-dimensional. La manera más simple para hacer la separación entre dos clases es una línea recta, un plano recto o un hiperplano N-dimensional. Pero hay ocasiones en que la separación de los datos no es fácil de realizar. Esto es debido a que el mundo real no es ideal y por lo tanto no podemos separar cualquier tipo de distribución de datos con una recta. El algoritmo SVM tiene que tratar con varias variables predictorias, curvas no lineales de separación, casos en los que los conjuntos de datos no pueden ser completamente separados y clasificaciones de más de dos categorías.

Por lo tanto, los clasificadores SVM utilizan un kernel para realizar la separación entre los datos que soluciona los problemas anteriores. Esta técnica consiste en proyectar la información a clasificar a un espacio de características de mayor dimensión, el cual aumenta la capacidad computacional. Es decir, transformaremos el espacio de entradas X a un nuevo espacio de características de mayor dimensionalidad.

3.2.4 Ejemplo visual del funcionamiento

A continuación, mostramos un ejemplo donde queremos realizar una separación entre los puntos rojos y azules. El problema que se nos presenta, y que se muestra en la Fig. 14, es que no podemos trazar una línea recta que separe una clase de otra:

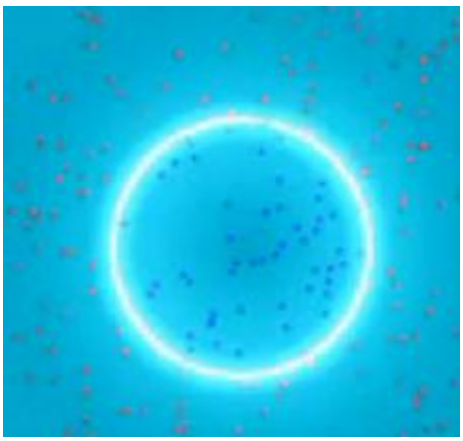


Figura 14: Ejemplo de separación de clases

Como se ha comentado antes, la solución pasa por representar la misma información en un espacio de mayor dimensionalidad. Para ello, se utiliza un kernel que se encarga de transformar toda esa información a un nuevo espacio en el cual se pueda separar, tal y como se muestra en la Fig. 15. Una vez tenemos todos estos datos representados, se puede intentar trazar una linea recta o en su defecto un hiperplano, que separe las clases en este nuevo espacio de representación.

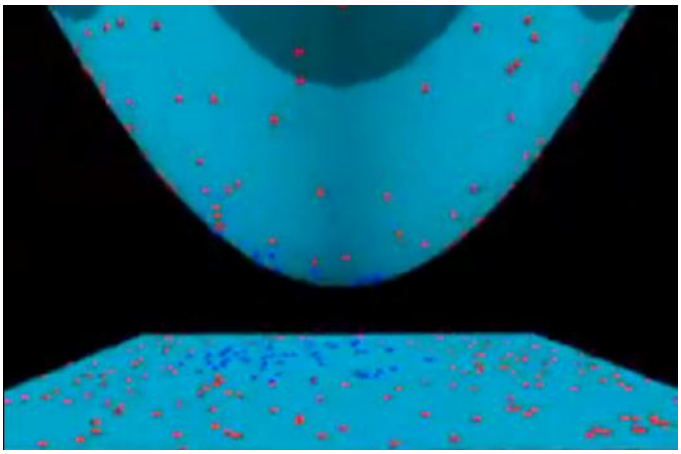


Figura 15: Proyección de los datos en un nuevo espacio

Aplicando el kernel, podemos separar claramente las clases en un nuevo espacio de más dimensionalidad, donde si se puede aplicar un hiperplano que separe ambas clases de la manera más eficiente. El resultado de esa separación la tenemos representada en la Fig. 16, mostrada a continuación.

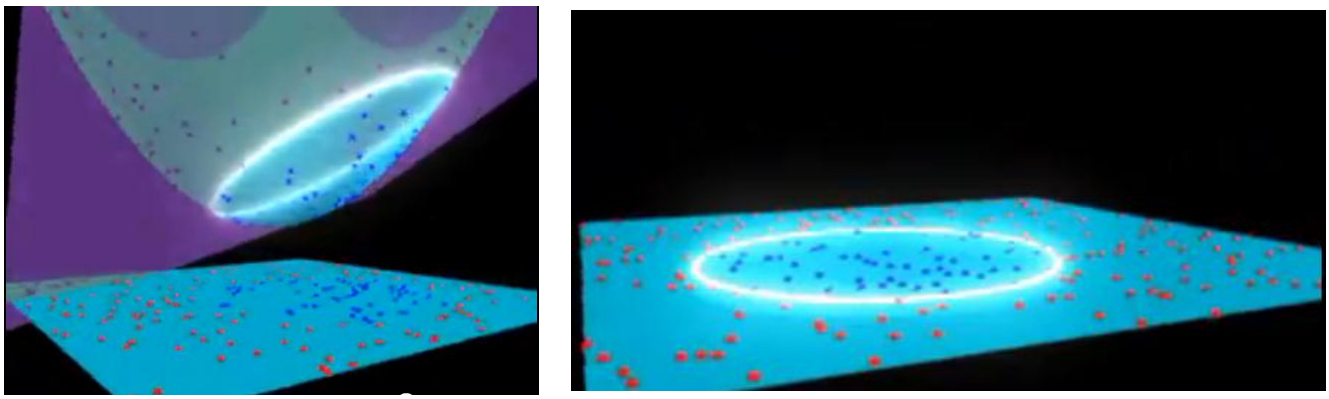


Figura 16: Separación en un nuevo espacio y obtención de los resultados finales

Finalmente, una vez que tenemos la separación óptima entre las dos clases, el resultado de todo este proceso se obtiene volviendo a representar todos estos datos al espacio o dimensionalidad inicial, donde por fin, tenemos todos los datos perfectamente clasificados. De esta manera, podemos realizar una separación en de cualquier tipo de datos y distribución, aplicando la transformación con un kernel.

3.3 Uso del descriptor HOG

En este punto se explicará el funcionamiento y cálculo del descriptor HOG que se utiliza para describir todas las imágenes y convertirlas a formato vector para ser clasificadas por los clasificadores SVM.

3.3.1 Explicación inicial

Como se ha comentado antes, para montar el modelo nos hace falta un conjunto de datos. En nuestro caso son imágenes que hay que describirlas de alguna forma, es decir, representar la misma información pero representación numérica. Para describir las imágenes utilizamos el descriptor HOG (Histogram Oriented Gradients), que es muy utilizado en los campos de visión artificial y procesamiento de imágenes para la detección de objetos, y que en conjunción con los clasificadores SVM, ofrece muy buenos resultados.

La idea básica de HOG es que cualquier imagen se puede describir como una distribución de intensidad de los gradientes o direcciones de cambio de intensidad. La implementación consiste en dividir la imagen de entrada en varias subregiones conectadas del mismo tamaño, llamadas celdas. Y para cada celda, se calcula el histograma de las orientaciones de los gradientes, o en otras palabras, un histograma de todas las orientaciones detectadas en función del cambio de intensidad.

La combinación de los histogramas de todas las regiones es lo que al final se llama el descriptor de la imagen, y que nos permite almacenar la mayoría de características de la imagen, en forma de vector numérico. Para mejorar el resultado, se realiza una serie de pre-procesado de la imagen, en la cual se calcula la media de la intensidad de la imagen para cada bloque o celda. En definitiva, la operación de normalización nos permite que haya menos variancia en la intensidad en toda la región de la imagen y tener menos errores producidos por el dispositivo de entrada.

El cálculo del descriptor HOG tiene varias ventajas frente a otros descriptores. El hecho de que se utilicen varias regiones en vez de analizar la imagen entera, hace que afecte menos la forma geométrica del objeto a detectar. Esto no se aplica en la orientación del objeto, pero el resultado sería muy parecido, ya que finalmente se obtendría un

descriptor muy parecido, porque las orientaciones seguirían siendo las mismas, independientemente de la orientación del objeto. Además, para temas de rendimiento, cálculo de las distintas regiones se puede paralelizar, ya que el cálculo de cada celda es independiente de las demás.

3.3.2 Implementación del algoritmo HOG

El primer paso en la obtención del descriptor es el cálculo de los gradientes de la imagen. Para ello, primero se suaviza la imagen de entrada, con un filtro Gausiano, y a continuación se aplica una máscara que realice la derivada en la imagen. Para este caso, se utiliza un máscara Sobel que se convoluciona con la imagen original. Esta operación devolverá un impulso en aquella zona donde hay un cambio de intensidad grande. El resultado que obtenemos de esta primera operación se muestra en la Fig. 17.

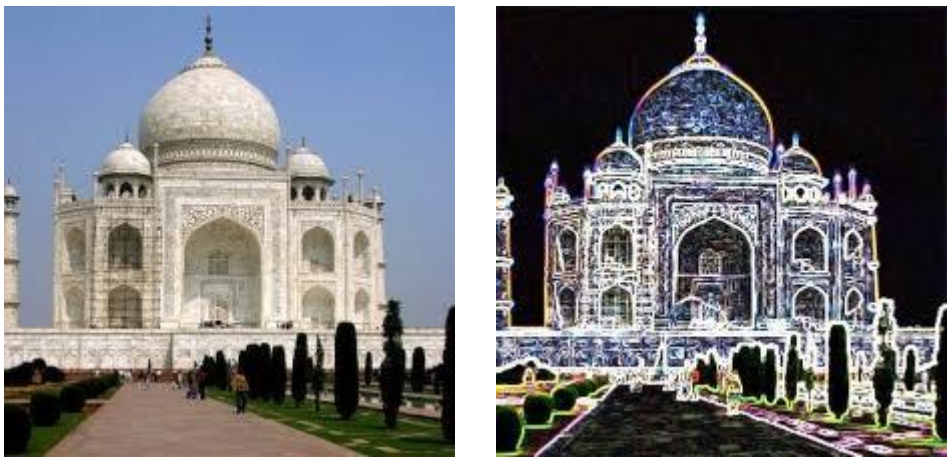


Figura 17: Resultado de la operación Sobel

Las máscaras Sobel pueden tener tamaños distintos. En función de este se detectarán mas detalles o menos. La fórmula que representa la generación de la máscara Sobel es la siguiente:

$$[-1, 0, 1] \text{ and } [-1, 0, 1]^T$$ [7]

Esta operación se realiza tanto en dirección horizontal (para detectar bordes verticales) y en dirección vertical (para detectar bordes horizontales). Una vez tenemos esta

imagen resultante con los valores de gradiente, el siguiente paso es calcular las orientaciones de todos estos.

La imagen de gradientes se divide en distintas celdas, y para cada celda se calculará un histograma que indicará que distribución de orientaciones sigue cada celda analizada. Estas orientaciones van de 0º a 180º o de 0º a 360º, en función de si se toman ángulos con signo o sin signo.

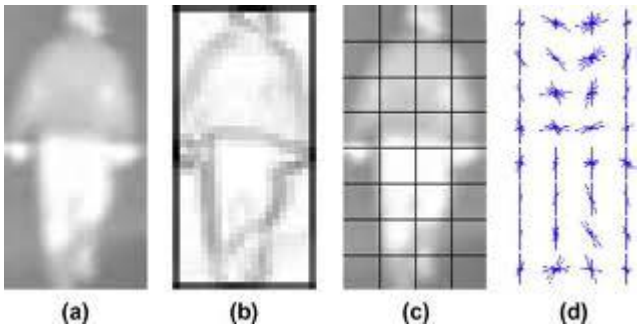


Figura 18: Ejemplo de HOG para una persona

En la Fig. 18 se puede observar la división de la imagen en celdas, donde para cada una existe un histograma distinto, que indica para esa región la distribución del número de ángulos.

Para tener aún un mejor resultado, las celdas se unen en bloques de mayor tamaño, donde el resultado será un descriptor mucho más completo. Finalmente estos descriptores finales se unen a un único resultante. En este punto podemos tener variaciones de magnitud en la salida, es decir, el descriptor no necesariamente tiene que salir normalizado. Por lo tanto, antes de entrenar los SVM, normalizamos el vector que forma el descriptor, en valores comprendidos entre 0 y 1. De esta forma conseguimos que todos los descriptores tengan la misma magnitud y que en el momento de realizar la representación en el modelo, se puedan diferenciar claramente una clase de otra.

Por lo tanto, este paso es muy importante para facilitarle la tarea al clasificador SVM, ya que el cálculo del hiperplano mejor se podrá realizar de una manera más eficiente. Por lo tanto, al final tendremos una función de decisión más potente, que nos dirá para cualquier objeto en particular, si este pertenece a una clase del modelo o no.

3.3.3 Observaciones sobre el tamaño del descriptor

Hay que tener en cuenta que la implementación que hemos usado de HOG se encuentra en las librerías OpenCV. Todos los parámetros del tamaño de la imagen, el número de bloques y de celdas, tienen que estar previamente definidos para calcular el descriptor. Esto implica que se obliga a tener un tamaño mínimo de imagen sobre la que se va a realizar la descripción, y esto es un problema, ya que las imágenes de extremidades que hemos recortado anteriormente tienen dimensiones distintas.

Para solucionar este problema, hemos recurrido a re-escalar todas aquellas regiones, el tamaño de las cuales fuera más pequeño que el mínimo especificado en la inicialización de HOG, de esta manera, podemos describir cualquier imagen, independientemente de su tamaño. Además, estos parámetros no pueden cambiar durante la ejecución del código, es decir, no se pueden ajustar dinámicamente durante la marcha, ya que en función del número de bloques, el tamaño de la imagen de entrada y el número de celdas, implicarán que el tamaño del descriptor sea diferente.

Por lo tanto, estos parámetros tienen que estar fijos, ya que el descriptor de cualquier imagen tiene que tener el mismo tamaño, para poderlo posteriormente localizarlo en el modelo que SVM calculará durante la fase de Training. Y lo mismo nos pasa con las imágenes de test, donde todas aquellas que sean inferiores al tamaño mínimo de entrada, deberán ser re-escaladas. Esta operación si que se realiza automáticamente, incluyendo un pequeño código que comprueba antes de la ejecución del HOG si el tamaño cumple con el prerrequisito anterior.

Finalmente el descriptor será enviado a todos los clasificadores, que siguen un patrón que se explicará a continuación, y finalmente devolverán si pertenecen a una clase u otra de las que han sido previamente entrenadas. Esto se realiza mediante la creación de una matriz ECOC.

3.4 ECOC Tree

En este punto se va a describir el funcionamiento del árbol de decisión basado en ECOC. La idea básica es definir la manera de montar el conjunto de clasificadores SVM para que entre ellos se pueda tener un resultado más eficiente. Finalmente, mediante el uso de ECOC, se podrá mejorar el resultado final.

3.4.1 Nociones básicas de ECOC

Una vez tenemos todos los clasificadores preparados, el objetivo es obtener la clase a la que pertenece el objeto de entrada a testear. Para montar el sistema de detección, hemos entrenado un conjunto de clasificadores, cada uno con una combinación de distintas extremidades. Y aplicando una matriz de corrección de errores, finalmente obtenemos la clase que el conjunto de clasificadores ha detectado.

Esta matriz se llama ECOC y consiste en codificar el resultado del conjunto de clasificadores para predecir la extremidad detectada por todo el conjunto. El funcionamiento del sistema ECOC está descrito en profundidad en el PFC del Sr. Daniel Sánchez Abril. La idea consiste en el problema de la detección de un objeto en sistemas de multi-clasificación, donde el número de clases es superior a 2. La mayoría de las técnicas están basadas en la votación, pero se ha demostrado recientemente que a parte de permitir votar entre varios clasificadores binarios, también permiten incluir un sistema de detección de errores.

Básicamente ECOC nos permite combinar varios clasificadores binarios para poder abordar correctamente la solución de un problema multi-clase. Mediante el uso de la corrección de errores, aplicada en la tecnología de la comunicación, si un ejemplo es incorrectamente clasificado por alguno de los clasificadores aprendidos, todavía puede ser correctamente clasificado después de haber sido decodificado.

En la Fig. 19 se muestra un ejemplo de matriz ECOC, donde las columnas serán los clasificadores y las filas los ejemplos a entrenar. A continuación se muestra el ejemplo.

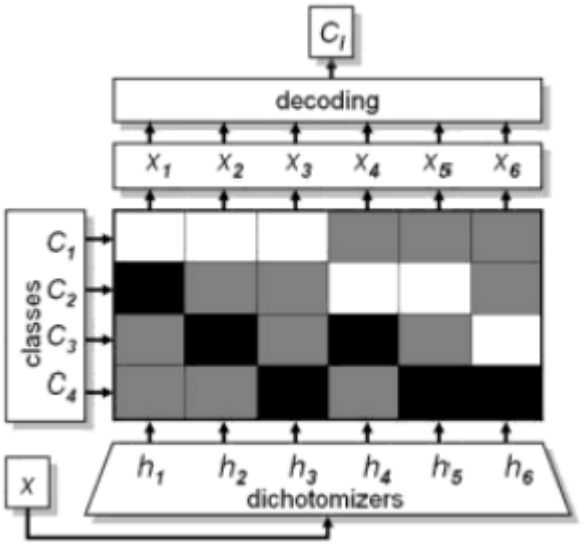


Figura 19: Sistema ECOC

En nuestro caso tenemos en cada columna un clasificador binario distinto y en las filas tenemos las distintas clases de nuestro sistema multi-clase.

Dado un ejemplo de entrada, no utilizado en el conjunto de datos del entrenamiento, este se envía a cada clasificador SVM de forma independiente, y este devuelve 1 o -1 en función de la clase a la que pertenezca. Por lo tanto, para un ejemplo, se obtendrá un vector con un conjunto de predicciones. Este vector se compara con la matriz ECOC fila a fila utilizando el cálculo de una distancia, y se asigna el identificador de aquella clase, la cual tenga la distancia inferior.

Podemos optar por distintas clasificaciones, como pueden ser la Euclidiana y la Loss-Weighted. Estas se encuentran extensamente explicadas en el PFC citado anteriormente y son distintas maneras de montar la matriz ECOC en función del cálculo de distancias que se aplique.

3.4.2 El árbol de decisión

En todo el conjunto de datos tenemos 6 extremidades posibles. El siguiente paso es crear la matriz ECOC que nos permitirá que en función de las predicciones que nos devolverán los clasificadores, obtener la clase a la que pertenece el objeto de entrada. Para ello, se ha utilizado un árbol de decisión, donde se comparan distintas clases y en función

del resultado conjunto de todos los clasificadores, se obtendrá el resultado. Estos árboles se utilizan en conjunción con los clasificadores SVM, y se obtienen muy buenos resultados.

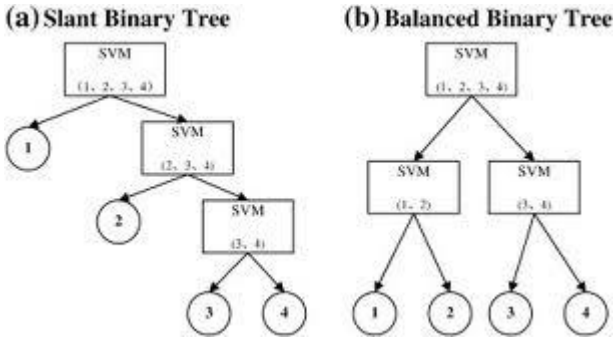


Figura 20: Tipos de árboles binarios

Como se puede ver en la Fig. 20, el árbol de decisión esta compuesto por varios nodos. El nodo raíz contiene una clasificación de todo el conjunto de clases, mientras que en función del resultado de esta, se procesará una parte u otra del árbol. Podemos tener árboles balanceados o desbalanceados, esto va en función del número de clases que se quieran incorporar al sistema. Por lo general, lo normal es utilizar árboles balanceados, teniendo en mente la eficiencia del sistema.

En nuestro caso, tenemos un 6 extremidades que se pueden definir con 5 clasificadores distintos, cada uno entrenado para detectar un subconjunto, excepto la raíz, que los contiene a todos. El árbol de decisión que hemos diseñado queda de esta manera:

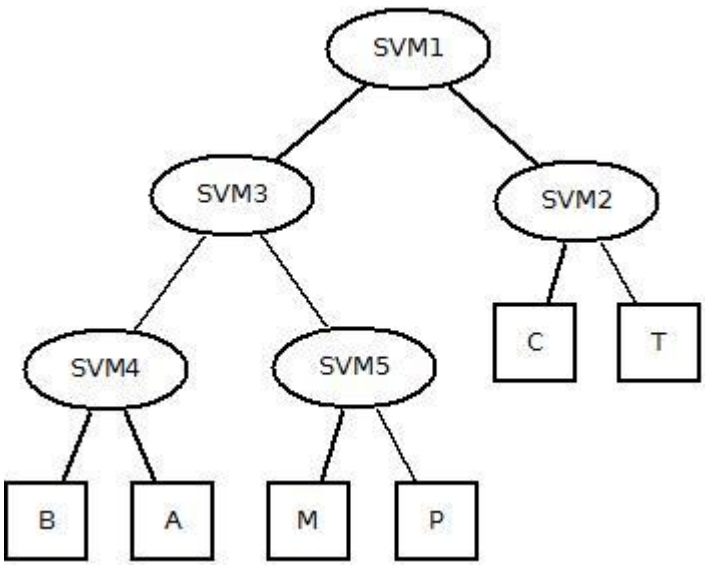


Figura 21: Árbol binario con las extremidades

Este árbol binario contiene todas las extremidades. La nomenclatura se representa a continuación, para simplificar la Fig. 21 mostrada anteriormente:

B	**Detección de Brazo**
A	**Detección de Antebrazo**
M	**Detección de Muslo**
P	**Detección de Pierna**
C	**Detección de Cabeza**
T	**Detección de Torso**

Cuadro 4: Nomenclatura de las extremidades

3.4.3 SVM y los conjuntos de datos

Como se ha comentado anteriormente, tenemos 5 clasificadores: SVM1, SVM2, SVM3, SVM4, SVM5. El conjunto de entrenamiento es diferente para cada clasificador. A continuación incluyo las extremidades que se ha utilizado en cada clasificador en función del diagrama anterior del árbol de decisión.

SVM1	Torso, Cabeza **VS** Antebrazo, Brazo, Muslo, Pierna
SVM2	Torso **VS** Cabeza
SVM3	Antebrazo, Brazo **VS** Muslo, Pierna
SVM4	Antebrazo **VS** Brazo
SVM5	Muslo **VS** Pierna

Cuadro 5: Conjunto de datos de entrenamiento

Una vez que tenemos toda esta parametrización montada, el último paso es construir la matriz ECOC donde se realizarán las predicciones con corrección de error. Esta matriz será fija y no se podrá cambiar. Por lo tanto, con este sistema, ahora en vez comparar extremidad a extremidad por cada clasificador, tenemos 5 clasificadores que trabajan en conjunción gracias a la matriz para obtener un resultado más óptimo. La matriz ECOC resultante se muestra en el Cuadro 6, que se encuentra a continuación:

	SVM1	SVM2	SVM3	SVM4	SVM5
Torso	1	1	0	0	0
Cabeza	1	-1	0	0	0
Antebrazo	-1	0	1	1	0
Brazo	-1	0	1	-1	0
Muslo	-1	0	-1	0	1
Pierna	-1	0	-1	0	-1

Cuadro 6: Matriz ECOC basada en el árbol de decisión

Hay que notar que en todas aquellas posiciones donde hay 0, significa que no se ha tenido en cuenta las clases marcadas en las filas con el clasificador marcado en la columna. Esto se realiza así, ya que como hay que calcular una distancia entre la predicción que obtenemos en la fase de test y la matriz ECOC, sólo tendremos en cuenta aquellas posiciones donde los valores sean 1 o -1.

4 Resultados

En este punto se va a explicar la metodología de las pruebas, mostrar los resultados que se han obtenido realizando distintas pruebas con parámetros diferentes, imágenes mostrando resultados visuales para toda la fase de funcionamiento del sistema.

4.1 Proceso de Entrenamiento de las cascadas Adaboost

En este punto vamos a detallar la información sobre la implementación de los distintos algoritmos y sus resultados que hemos obtenido. Como se ha comentado anteriormente, en una primera fase, nuestra intención es obtener la posición de la persona, es decir, poderla segmentar del fondo de la imagen. Para ello, nuestro primer paso es realizar un entrenamiento de las cascadas, utilizando un clasificador Adaboost por cada extremidad comparándola con imágenes de fondo.

Las librerías OpenCV nos proporcionan una serie de utilidades, que hay que compilar por separado, que se encargan de realizar el entrenamiento con los datos que nos interesa. Las utilidades que nos proporciona OpenCV para Adaboost con características HAAR son:

- **Opencv_createsamples**
- **Opencv_haartraining**
- **Opencv_performance**
- **Opencv_convertcascade**

Inicialmente necesitamos preparar todo el conjunto de datos de las imágenes. Para ello, hemos creado una carpeta por cada extremidad donde contiene el conjunto de datos de positivos y el de negativos. Para que las imágenes puedan ser utilizadas, se ha generado unos archivos para cada conjunto de datos, donde se indica la ruta de cada imagen utilizada en la fase de entrenamiento. Además, se informa del tipo de clase a la que pertenece y las dimensiones de esta. A continuación pongo un ejemplo:

/home/juancarlos/CascadaCabeza/posiivos/1.jpg 1 0 0 56 56

Como se puede observar, la primera parte es la ruta absoluta de la imagen, el 1 nos indica que pertenece al conjunto de datos de positivos, (0,0) nos indica la posición inicial de la imagen, y (56, 56) nos indica la posición final. Hay que realizar el mismo proceso para el conjunto de datos de negativos.

Una vez tenemos la primera parte de los datos montada, el siguiente paso es convertir todo esto al formato que utiliza las utilidades de OpenCv. Para ello, utilizamos la utilidad *opencv_createsamples*. En este punto, utilizamos una serie de parámetros que los hemos fijado por defecto. En principio, como tenemos muchas imágenes de tamaños diferentes, lo que se quiere es normalizarlo todo. Por lo tanto, vamos a convertir de forma automática todas las imágenes de positivos a tamaño 32x32. Hay que tener en cuenta, que cuanto más grande sea este valor, más nos va a costar entrenar los clasificadores, por lo que decidimos que 32x32 era un tamaño bastante bueno y que se puede procesar razonablemente rápido. La ejecución de *opencv_createsamples* es la siguiente:

opencv_createsamples –info positivos.dat –vec positivos.vec –w 32 –h 32 –num 11388

En este caso, estamos llamando a la utilidad para generar los datos que va a procesar el entrenador. El parámetro *-info* se encarga de darnos información de cómo realiza la fase de preparación de los datos. El parámetro *positivos.dat* contiene todas las rutas de las imágenes de training, *-vec positivos.vec* será el archivo de salida donde contiene toda la información ya procesada, *-w 32* especifica el ancho de las imágenes y *–h 32* la altura. Finalmente el parámetro *–num 11388* indica el número de imágenes a procesar.

Hay que comentar que este proceso se ha tenido que realizar con todas las cascadas, por lo que en función de la extremidad, el parámetro *–num* puede variar. Por ejemplo, hay casos donde tenemos bilateralidad en la extremidad, lo que hace que el conjunto de datos se duplique, en el caso de los brazos, antebrazos, piernas y muslos.

Finalmente, el siguiente paso es entrenar los clasificadores. Hay que tener en cuenta que en función del número de ejemplos y del número de niveles, es una operación que nos va a tardar bastante. En este punto tenemos varios parámetros que nos afectan directamente en la fase de entrenamiento y que vamos a explicar a continuación. La ruta de ejecución de *opencv_haartraining* es la siguiente:

Time opencv_haartraining –data /home/juancarlos/CascadaCabeza –vec positivos.vec –bg negativos.dat –nstages 8 –minhitrate 0.97 –maxfalsealarm 0.4 –npos 1500 –nneg 1500 –w 32 –h 32 –nonsym –mem 512 –mode ALL

Como se puede ver, tenemos una gran cantidad de parámetros que nos afectan directamente.

- **Nstages**: Número de etapas o clasificadores débiles que tendrá la cascada.
- **Minhitrate**: El mínimo número de acertados en cada etapa.
- **Maxfalsealarm**: Es la máxima falsa alarma en cada etapa.
- **Npos**: El número de ejemplos positivos.
- **Nneg**: El número de ejemplos negativos.
- **Nonsym**: Indica que los ejemplos no necesariamente son simétricos.
- **Mem**: La cantidad de memoria RAM que se debe reservar.
- **Mode**: Especifica la cantidad de features HAAR que se utilizarán.
- **W**: Especifica los píxeles que tiene la imagen de ancho.
- **H**: Especifica los píxeles que tiene la imagen de alto.

Una vez que la operación de entrenamiento finaliza, *opencv_haartraining* nos genera una carpeta por cada nivel de la cascada con un archivo que contiene la configuración de ese clasificador débil en cuestión. Este archivo contiene todo el conocimiento que necesita ese clasificador débil para poder funcionar de acuerdo a los parámetros fijados anteriormente. El paso siguiente es transformar todas las carpetas correspondientes a los clasificadores débiles a un único archivo XML.

Para realizar la transformación a XML utilizamos la utilidad *opencv_convertcascade* que básicamente nos empaqueta todas las carpetas en un único archivo que utilizaremos en la siguiente fase. Los parámetros de generación del archivo son los siguientes:

opencv_convertcascade –size=32x32 /home/juancarlos/CascadaCabeza

En este caso lo que estamos indicando es que la utilidad nos genere el XML en la carpeta justo anterior a la especificada, con los tamaños de imagen 32x32 y la carpeta que

contiene todas las carpetas de cada clasificador débil. Por lo tanto, el archivo resultante se almacenará en **/home/juancarlos/CascadaCabeza.xml**.

Con conjunto de datos de negativo, el funcionamiento es un tanto diferente. En este caso no hace falta obtener distintas subregiones de estas imágenes, sino que el entrenador ya obtiene tantas subregiones como se hayan especificado en los parámetros. En nuestro caso, especificamos 5000 subregiones, que se extraerán de forma aleatoria en todas las imágenes que componen el conjunto de datos de negativos.

Hay que comentar que el proceso de entrenamiento es muy costoso computacionalmente, ya que hay que ir aprendiendo exponencialmente el número de elementos que no han sido detectados en las etapas anteriores. Esto se verá reflejado en los tiempos de ejecución, indicados más adelante en esta memoria.

4.2 Proceso de Testeo de las cascadas Adaboost

Hasta ahora, esto ha sido la parte de entrenamiento de los clasificadores. Ahora tocaría entrar en la fase de testeo. Para ello, las librerías OpenCV nos ofrecen la utilidad *opencv_performance*, que nos ejecuta automáticamente un escaneo de toda la imagen en busca de objetos. El problema que tuvimos en este caso, es que la utilidad realiza un escaneo cuantitativo y no cualitativo. Es decir, nos dice si en la imagen se ha detectado la extremidad pero no donde se encuentra. Además, nos interesa tener una cierta información extra de todas las detecciones que hemos obtenido, las posiciones donde encuentran, por lo que al final desistimos de utilizar esta utilidad.

Por otra parte, realizando un proceso de investigación, encontremos que a parte de las utilidades, en las librerías también disponemos de los métodos y las clases para realizar el mismo procesado, pero al nivel que nos interesa. Como se ha comentado antes, el hecho de convertir todo el conocimiento de las cascadas a un XML es una ventaja, ya que en esta fase podemos cargar este archivo y realizar el proceso de testing.

La implementación de esta parte ha sido optimizada. Hasta ahora, para cada ejemplo o subregión de test obtenida desde el sliding window, requería ejecutar cada vez la aplicación que realiza la detección. Esto significa que para cada subregión, tenemos que

cargar los archivos XML para procesarla, lo que es bastante costoso. La mejora consiste en reducir el tiempo de ejecución mediante la carga única de los archivos XML. Ahora, desde el inicio de la fase de sliding window, los archivos XML se cargan una única vez al inicio y se realiza todo el proceso de sliding window. Cuando finaliza esta fase, finalmente se libera la memoria.

Como se ha comentado anteriormente, el proceso de sliding window consiste en recorrer toda la imagen con un tamaño de ventana variable y un cierto desplazamiento entre las posiciones de la ventana. La ventaja de este sistema es que nos permite la detección de objetos independientemente del tamaño de estos, ya que se realiza un barrido en todos los posibles tamaños de ventana dentro de la imagen.

Posteriormente, estos datos se almacenan en un vector donde serán procesados para aplicar un sistema de votación y dar probabilidades a aquellas zonas donde se han detectado más extremidades del mismo tipo.

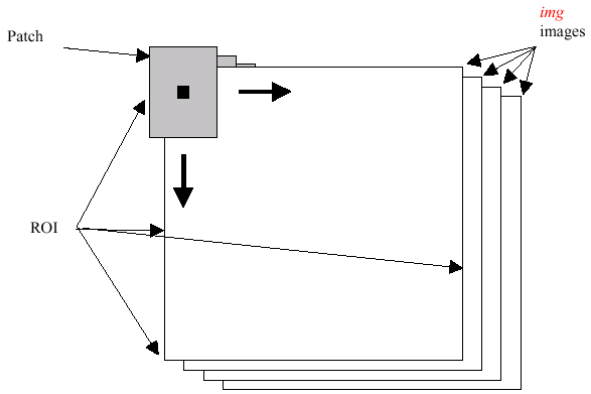


Figura 22: Funcionamiento del sliding window (ventana deslizante)

A continuación, para cada subregión de la imagen, se realiza una llamada a las 6 cascadas que previamente hemos entrenado (una por extremidad) y en función de su predicción, estas nos darán un resultado. Este resultado será un vector de 6 posiciones (tantas como clasificadores tengamos) donde posteriormente se aplicará la técnica ECOC y se obtendrá la extremidad que definitivamente se ha detectado, en función del resultado de todos los clasificadores. Se han aplicado pesos que se multiplican al resultado que dan cada una de las cascadas, ya que hay clasificadores que aprenden mejor que otros, y por lo tanto, se pueden considerar más fiables.

Este sistema nos va a permitir, mediante el cálculo de probabilidades, que regiones de la imagen tienen más probabilidad de ser una extremidad, y en caso de serla, tener también el identificador asociado a esta. En función del cómputo final, se obtendrá una imagen de intensidad donde se muestran con más intensidad, donde hay más probabilidad. Esta imagen nos servirá posteriormente para segmentar a la persona y pasar por la segunda fase, basada en clasificadores SVM, que se explicarán más adelante.

4.3 Pruebas realizadas con Adaboost y distintas configuraciones

Para poder contrastar como de buena es nuestra solución, hemos utilizado 3 configuraciones distintas para el entrenamiento de las cascadas Adaboost. Las configuraciones son:

- *Adaboost 5000 VS 5000 8 Niveles: Extremidad VS Fondo.*
- *Adaboost 5000 VS 5000 8 Niveles: Extremidad VS Extremidades y Fondo.*
- *Adaboost 5000 VS 5000 5 Niveles: Extremidad VS Extremidades y Fondo.*

4.3.1 Adaboost 5000 VS 5000 8 Niveles Extremidad VS Fondo

Esta fue la primera prueba que se realizó. Se decidió inicialmente una profundidad de la cascada de 8 niveles y para el entrenamiento se eligieron un total de 10000 muestras repartidas al 50% entre la extremidad y las imágenes de fondo. En total se han obtenido 6 cascadas, una por cada extremidad. El tiempo de entrenamiento fue aproximadamente de 5 días. En este caso, no se obtuvo ningún tipo de información sobre pesos.

4.3.2 Adaboost 5000 VS 5000 8 Niveles Extremidad VS Extremidades y Fondo

Esta fue la segunda prueba que se realizó. Se intentó ver si había algún tipo de mejora respecto a la primera versión. Esta segunda versión consiste en entrenar el mismo número de positivos (las mismas imágenes) pero con imágenes de fondo y extremidades. Esto supone un problema, ya que si estamos entrenando la cascada que detecta cabezas, como negativos no puede haber ninguna.

Por lo tanto, a parte de las imágenes de fondo, se incluyeron una serie de imágenes de los fotogramas que se utilizaron para extraer las extremidades, de tal manera, que el mismo clasificador obtuviera esas 5000 imágenes de negativos de forma totalmente aleatoria.

Para el proceso de introducción de los fotogramas de los vídeos, hizo falta relacionar cada fotograma con la máscara en cuestión de la extremidad que se quería eliminar. Al final, montemos un programa que automatizaba todo el proceso.

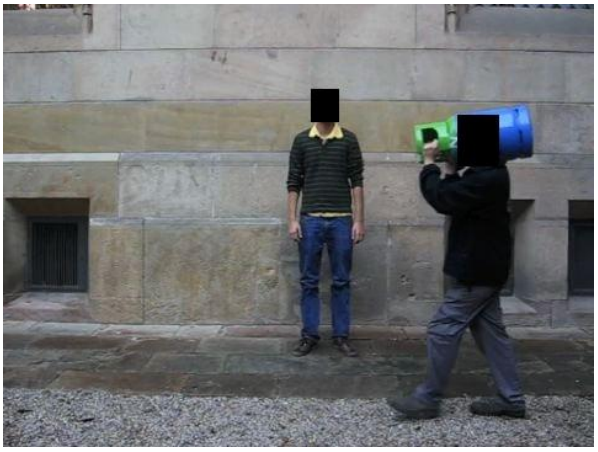


Figura 23: Fotograma con la cabeza eliminada

Este mismo proceso se tuvo que realizar para cada extremidad que se utilizaba en el conjunto de entrenamiento. Al finalizar todo el proceso, en función de todos los resultados que nos da la utilidad de entrenamiento, montemos un sistema de pesos, en el cual se daba mayor importancia a aquellos clasificadores que daban mejores resultados, o tenían un porcentaje de acierto más elevado. El algoritmo utilizado se muestra en la Fig. 24. El cálculo de los pesos de las cascadas, se realizaba mediante la fórmula siguiente:

$$\frac{\prod_{i=1}^{N} P_i + \prod_{i=1}^{N} (1 - N_i)}{2} \quad [7]$$

Para la configuración de 8 niveles, los pesos para cada extremidad son los siguientes:

Cabeza	Torso	Antebrazo	Brazo	Pierna	Muslo
0.53420	0.40452	0.24554	0.23288	0.27404	0.30747

Cuadro 7: Pesos de las cascadas

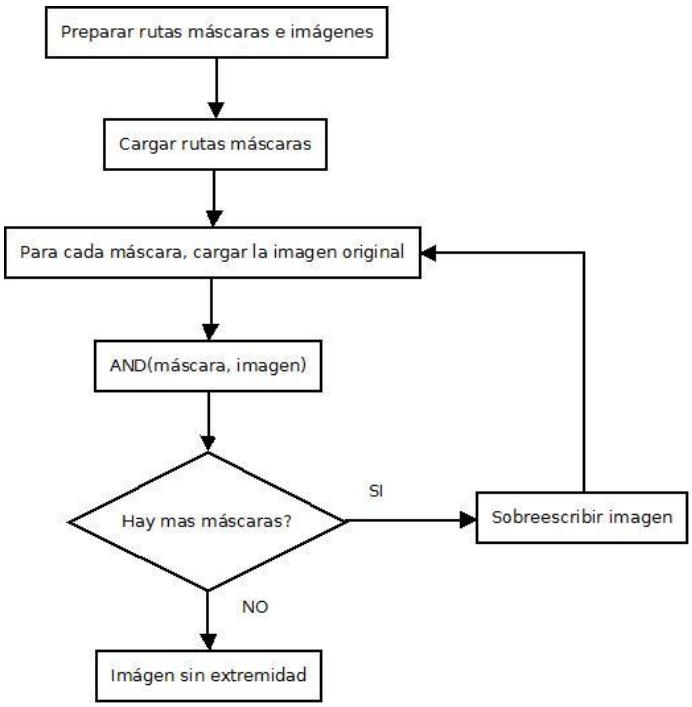


Figura 24: Algoritmo de obtención de la imagen sin extremidad

Estos pesos son los que van directamente al cálculo de la distancia en el sistema ECOC. De esta manera, se puede dar más importancia a los clasificadores que tienen mejor coeficiente. En este caso, vemos que las cascadas de cabeza y torso tienen unos coeficientes bastante altos, lo que nos dice que serán más efectivas que las demás.

Hay que comentar que en este caso, el tiempo de entrenamiento fue muy parecido al primero caso, sobre unos 5 días de constante funcionamiento, hasta encontrar una solución que cumpla los parámetros de entrada. En el Cuadro 8 se puede observar el tiempo de entrenamiento de cada clasificador:

Cabeza	Torso	Antebrazo	Brazo	Pierna	Muslo
554min 38s	813min 29s	2802min 5s	2653min 4s	2252min 14s	2658min 31s

Cuadro 8: Tiempo de entrenamiento por extremidad

Como se puede observar, los clasificadores que han requerido de más tiempo han sido los que clasificaban antebrazos, brazos piernas y muslos. Esto es debido a que estas extremidades son muy parecidas entre si, lo que implica que haya una mayor cantidad de features Haar que se deban aplicar a la imagen para poder describirla correctamente.

4.3.3 Adaboost 5000 VS 5000 5 Niveles Extremidad VS Extremidades y Fondo

En este caso, el procedimiento ha sido prácticamente igual al explicado anteriormente, simplemente se ha reducido el número de etapas de los clasificadores. Los resultados de esta etapa daban grandes cantidades de falsos positivos, ya que al pasar por menos clasificadores débiles, había más margen para que ejemplos mal clasificados pasasen como buenos, por lo tanto, esta solución finalmente se desestimó.

4.4 Resultados visuales obtenidos por Adaboost

Finalmente se comprobó por los resultados que la mejor opción era la configuración de cascada Adaboost con 8 niveles y extremidad VS extremidades y fondo (explicado en el punto 4.3.2 de esta memoria). Hay que tener en cuenta que es razonablemente normal que obtengamos mejores resultados, ya que estamos entrenando una extremidad contra el fondo y todas las demás menos ella misma. Esto hace que tengamos menos confusiones, ya que se ha entrenado para detectar por ejemplo cabezas entre todo el universo que contiene el conjunto de entrenamiento.

En las siguientes imágenes, se podrá observar como después de realizar un recorrido por toda la imagen con diferentes tamaños de ventana, los resultados son bastante buenos. El resultado es una imagen en blanco y negro donde se refleja la probabilidad de que haya en una posición el objeto a buscar. Se muestra en la Fig. 25.

Imagen original **Resultado de Adaboost**

Figura 25: Ejemplo 1, detección de la persona por probabilidades

Como se puede observar en la Fig. 25, los resultados son muy buenos. En ell resultado que obtenemos de las cascadas Adaboost, tenemos una zona marcada con gran intensidad que es la correspondiente a la persona, y que está exactamente en la misma región. Se puede ver también que hay varios falsos positivos alrededor de toda la imagen, pero no se tienen en cuenta ya que la probabilidad de estos es bastante inferior a la probabilidad donde se encuentra el cúmulo de detecciones, alrededor de la persona.

Imagen original **Resultado de Adaboost**

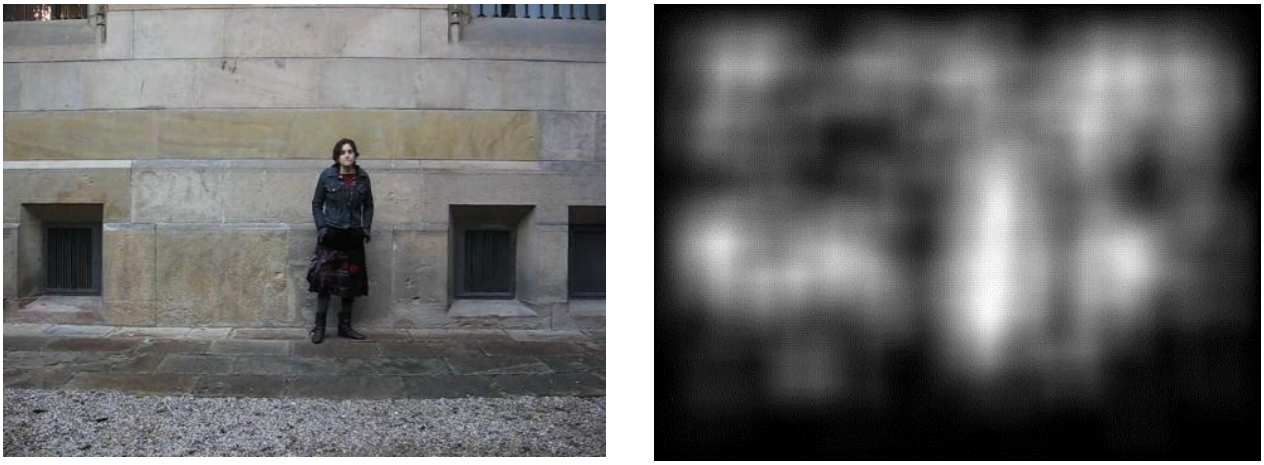


Figura 26: Ejemplo 2, detección de la persona por probabilidades

En este caso, como en el ejemplo anterior, en la Fig. 26 se representa con mayor intensidad la zona donde se encuentra la persona. Por lo tanto, los resultados son bastante buenos para detectar en que zona se encuentra la persona. Como antes, también tenemos zonas de falsos positivos, pero que al tener menos peso, no se tienen en cuenta para la siguiente fase.

4.5 Post-procesamiento después de aplicar las cascadas

Una vez que tenemos un mapa de probabilidad que nos indica donde se encuentra la persona, por el nivel superior de intensidad, el siguiente paso es realizar una segmentación de la imagen, de tal manera que se obtendrá aquella región donde se encuentre únicamente la persona.

Desde un inicio se pensó en recortar una región cuadrada sobre aquella zona donde había más probabilidad de que fuera una persona. Pero esto nos suponía un problema, ya

que en la segunda fase de clasificación basada en SVM no tenemos en cuenta el conjunto de datos de fondo. Esto es un problema ya que el fondo se puede detectar como una extremidad más cuando no lo es. Por lo tanto, nos decidimos a utilizar una técnica más elaborada, en la que extraeremos a la persona sin fondo. Esta tarea se realiza mediante la utilidad GrabCut.

Nuestro objetivo, por lo tanto es hacer que las cascadas Adaboost eliminen todo lo que no sea persona e indiquen claramente donde se encuentra esta. En función de esta información, se extraerá la persona sin fondo gracias a GrabCut y posteriormente podrá ser procesada por los clasificadores SVM en la segunda fase de detección.

El funcionamiento de GrabCut se encuentra detallado en el PFC del Sr. Daniel Sánchez Abril, por lo que si se desea más información, se puede consultar su documento. Un ejemplo de todo el funcionamiento de la primera parte del sistema es la siguiente:

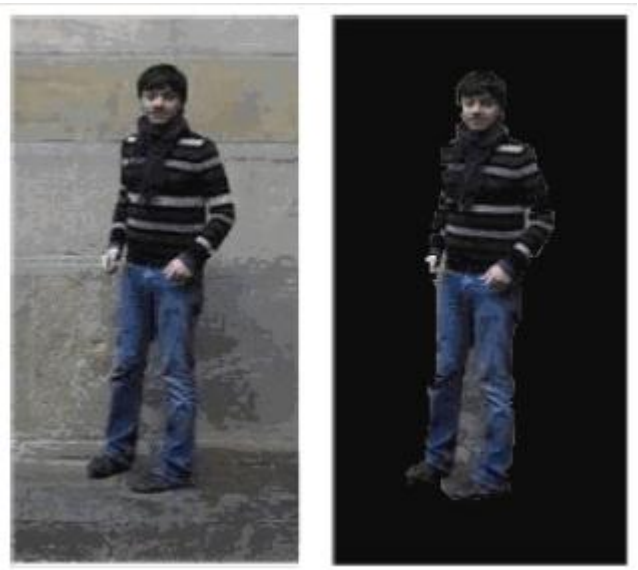


Figura 27: Imagen original y segmentación de la persona por GrabCut

Esta imagen se obtiene mediante la imagen original y la imagen que nos devuelven las cascadas Adaboost. En este caso, mediante GrabCut, se selecciona aquella área donde hay mayor concentración de probabilidad y se recorta, de tal manera que tendremos una imagen resultado como la mostrada a la derecha de la Fig. 27. La imagen segmentada con la persona ya no contiene información o regiones sobre el fondo, por lo que ya nos podremos centrar directamente en detectar sus extremidades mediante el uso de los clasificadores SVM.

4.6 Parámetros del descriptor HOG

El cálculo del descriptor HOG es prioritario para poder tener ejemplos de entrenamiento y test que se puedan utilizar en los clasificadores. Para ello, hemos utilizado la implementación de HOG que nos ofrece OpenCV. Dado el gran volumen de imágenes que tenemos recortadas y la gran variedad de tamaños que tienen estas, hace falta especificar unos tamaños estándar para que el cálculo del descriptor HOG no se vea afectado por las dimensiones de las imágenes.

En el capítulo de HOG se comentó que el descriptor necesita unos parámetros fijados de antemano y que no pueden ser modificados, ya que entonces el tamaño del descriptor nos cambiaría y al generar el modelo, lo haría malamente. Realizando un análisis de las extremidades recortadas, al final decidimos utilizar un tamaño mínimo de 64x64 píxeles, cosa que obliga que las imágenes que tienen tamaño inferior a este, necesiten ser re-escaladas. Los parámetros que se han utilizado para el cálculo de HOG son los siguientes:

- **Win Size**: 64x64 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.

4.7 Proceso de entrenamiento de los clasificadores SVM

En este caso, el proceso de entrenamiento ha consistido en la creación del conjunto de 5 clasificadores, incluyendo en cada uno el conjunto de datos siguiente:

SVM1	Torso, Cabeza **VS** Antebrazo, Brazo, Muslo, Pierna
SVM2	Torso **VS** Cabeza
SVM3	Antebrazo, Brazo **VS** Muslo, Pierna
SVM4	Antebrazo **VS** Brazo
SVM5	Muslo **VS** Pierna

Cuadro 9: Distribución de las extremidades por clasificador

En este punto, tenemos todas las imágenes de las extremidades que hemos utilizado para entrenar las cascadas Adaboost preparadas. El siguiente paso es convertirlas mediante HOG en descriptores de un mismo tamaño y guardarlas independientemente en 6 archivos distintos, que contendrán un descriptor por imagen. El proceso de extracción está reflejado en el diagrama de la Fig. 28, y es el siguiente:

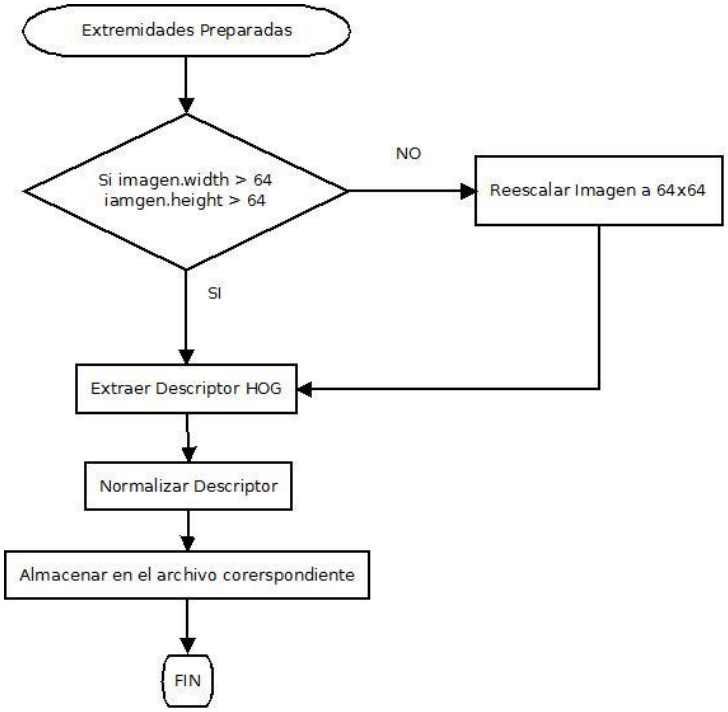


Figura 288: Diagrama de extracción del descriptor HOG

Una vez tenemos todas las extremidades descritas con HOG y almacenadas en sus correspondientes archivos en función de la extremidad, el siguiente paso es montar el conjunto de datos de entrenamiento de los clasificadores SVM. Para ello, el conjunto de datos puede variar, ya que en función del clasificador, tendrá más carga que otros. A continuación muestro la tabla con el número de ejemplos a entrenar por clasificador:

SVM1	1000 + 1000 **VS** 1000 + 1000 + 1000 + 1000 = 6000 ejemplos
SVM2	1000 **VS** 1000 = 2000 ejemplos
SVM3	1000 + 1000 **VS** 1000 + 1000 = 4000 ejemplos
SVM4	1000 **VS** 1000 = 2000 ejemplos
SVM5	1000 **VS** 1000 = 2000 ejemplos

Cuadro 10: Número de ejemplos por extremidad y total por clasificador

Esta tabla pertenece al conjunto de entrenamiento, mientras que en la fase de testeo se utilizan la mitad de cada clase. Es decir, en vez de 1000 ejemplos por clase, se utilizan 500 que no están incluidos dentro del conjunto de aprendizaje, si no el resultado es inválido.

Hay que tener en cuenta que dado el gran conjunto de datos que estamos tratando y que todos ellos se encuentran en archivos diferentes, implica que hay que unificarlos para crear el archivo de entrenamiento. Para ello, se ha desarrollado una pequeña aplicación en lenguaje Python que se encarga de asignar a la clase una etiqueta (+1 o -1), un rango de búsqueda donde se insertarán todos los elementos comprendidos dentro de este. Conforme se van montando los archivos, los descriptores se van concatenando a la información ya existente en el archivo, por lo que permite unificar todos descriptores de las extremidades en un único fichero que nos servirá para generar el modelo.

Para la preparación de esta fase, se ha utilizado la librería *LibSVM*, creada por Chih-Chung Chang y Chih-Jen Lin. Esta implementación incluye gran cantidad de bindings para distintos lenguajes de programación. En nuestro caso, utilizamos C/C++ por temas de eficiencia y velocidad. Esta librería incluye dos utilidades que son las que se encargan de entrenar y predecir los resultados:

- **Svm-train**: Se encarga de ejecutar el entrenamiento y generar el modelo.
- **Svm-predict**: Testea un archivo de entrada con descriptores a que clase pertenece.

En este punto nos surgió un problema y es que tal cual está montada la solución de *LibSVM*, para realizar una predicción, tendríamos que llamar al programa ejecutable *svm-predict* con el ejemplo de test a predecir. Esto es un problema, ya que implica guardar el descriptor de este objeto en un archivo, realizar una llamada al sistema para que arranque la utilidad de predicción y posteriormente volver a leer desde el archivo de resultados que genera. Además, hay que tener en cuenta que nos hace falta cargar los modelos, y estos en algunos casos llegan hasta los 150Mb de capacidad, por lo tanto, la solución de utilizar directamente la utilidad no nos sirvió.

La solución a este problema ha consistido en modificar el código fuente de *svm-predict* de tal manera que se puede ejecutar en tiempo real sin necesidad de llamar a ningún ejecutable. Por lo tanto, antes de realizar ninguna operación, se cargan todos los modelos, y

en función del clasificador se cambia la referencia al modelo actual al vuelo. De esta manera, tenemos un sistema que predice muy rápido.

Volviendo al tema de los parámetros, como hay que tener en cuenta distintos valores de c y gamma, hemos creado un pequeño programa que encuentra la combinación óptima que da mejores resultados. Este cálculo tiene que ser manual, es decir a base de ir comprobando distintas configuraciones, ya que depende de cómo están distribuidos los datos dentro del modelo. El parámetro *c* puede tomar los valores [0, 10, 100 y 1000] mientras que el parámetro *g* se mueve en rango de [0.01 a 0.99].

Para cada combinación de c y g, generamos un modelo nuevo mediante *svm-train* y los datos de entrenamiento. Una vez que el modelo está entrenado, el paso siguiente es testearlo con los ejemplos de test. Para ello, el número de aciertos se calcula por la formula:

$$\text{PROB} = \frac{\text{Nº ejemplos acertados}}{\text{Nº ejemplos totales}} \quad [8]$$

Los ejemplos de test son descriptores que tienen un label asignado que sabemos la clase a la que pertenece. El proceso consiste en atacar al clasificador con todos los ejemplos de test de su propia clase y comprobar en que casos acierta las extremidades y en que casos falla. Con la fórmula anterior se extraen unos coeficientes que indicarán como de bueno es el modelo que hemos entrenado.

Los comandos a ejecutar para realizar este proceso son los siguientes:

Svm-train –c valor_coste –g valor_gamma archivo_entrenamiento
Svm-predict archivo_testeo archivo_modelo archivo_predict

Básicamente, cuando se ejecuta la utilidad *svm-train*, se genera un archivo con el mismo nombre del de entrada pero con extensión *.model*. Este es el modelo que se utilizará para testear el clasificador. En la fase de testeo, *svm-predict* carga la lista de descriptores que contienen los ejemplos de test (*archivo_testeo*), el modelo generado anteriormente por *svm-train* (*archivo_modelo*) y el archivo donde se va a guardar para cada registro de test la clase predicha (*archivo_predict*).

Con todo esto, obtenemos una serie de resultados, que se muestran en el Cuadro 11 siguiente. En ellos se puede ver para cada extremidad, los coeficientes que se han obtenido y que se adecuan más a la predicción en función del número de aciertos. En el anexo incluyo las tablas completas con el resultado de cada combinación. Hay que comentar que primero se realizó una primera fase con saltos de gamma de 5 en 5, para acelerar el proceso. Posteriormente, cuando se obtuvo las zonas donde teníamos mejores resultados, entonces volvimos a realizar la misma prueba pero en saltos de gamma de 1 en 1. En resumen, para la primera fase de escaneo de los parámetros tenemos los siguientes resultados mostrados en la tabla:

Clasificador	Valor de C	Valor de Gamma	Probabilidad de Acierto
SVM1	10	0.95	97.7 %
SVM2	10	0.95	99.2 %
SVM3	10	0.95	93.5 %
SVM4	10	0.95	92.7 %
SVM5	1000	0.95	95.9 %

Cuadro 11: Configuraciones de C y Gamma para cada clasificador

Como se puede ver, la mayoría de clasificadores obtienen los mejores resultados en las zonas donde el parámetro c es 10 y gamma es 0.95. Es bastante interesante la gran tasa de aciertos del segundo clasificador (SVM2). Eso es debido a que se intenta clasificar cabeza contra torso y esto son objetos que no son para nada parecidos. En la mayoría de ejemplos de test los predecía correctamente. Sobre estos rangos, se realizó una segunda prueba para ver si se podía obtener aún mayor tasa de acierto. Los resultados para esta segunda fase de búsqueda son los siguientes:

Clasificador	Valor de C	Valor de Gamma	Probabilidad de Acierto
SVM1	10	0.99	96.8 %
SVM2	10	0.95	99.2 %
SVM3	10	1	93.7 %
SVM4	10	1	92.8 %
SVM5	1000	0.95	96.2 %

Cuadro 12: Valores de C y Gamma optimizados para cada clasificador

En esta segunda prueba que se realizó, se puede ver como la mejoría es muy poca. Por ejemplo, el mismo clasificador que hemos analizado antes (SVM2) no conseguía mejorar el ratio de acierto. Los demás han mejorado algunas décimas, variando el parámetro de gamma muy cerca de donde se había obtenido en la primera fase el mejor resultado. En principio, ya no se ha podido optimizar más estos resultados.

4.8 Resultados visuales de la salida de los clasificadores SVM

A continuación se van a mostrar una serie de ejemplos donde se puede ver, sin aplicar la técnica ECOC, las detecciones que nos devuelven los clasificadores. En nuestro caso, tenemos una imagen de entrada (Fig. 29), la cual se ha recorrido con sliding window a distintas escalas y que el resultado de las detecciones se han obtenido incrementando en 1 toda la región de la ventana deslizante que está afectada por la detección.

Las imágenes de la Fig. 30 representan un mapa de probabilidad donde se indican las predicciones de los clasificadores SVM. En este caso, la columna referente a SVM1, como clase +1 tiene todo el conjunto de datos referentes a la detección de torso y cabeza, mientras que la clase -1 tiene todo el conjunto referente a antebrazo, brazo, muslo y pierna. Si analizamos la salida del clasificador SVM2 se pueden ver varios cúmulos de puntos referentes a la clasificación entre torso (clase +1) y cabeza (clase -1). Si tomamos el clasificador SVM3, se puede ver una clara distinción entre el conjunto de datos de brazo y antebrazo (clase +1) contra muslo y pierna (clase +1). Hay que comentar que estos resultados son dependientes de la imagen de entrada. En algunas ocasiones los resultados no son tan buenos como podrían parecer. Esto puede ser debido a la falta de ejemplos de una clase concreta, el proceso de clasificación, etc.

Figura 29: Imagen de entrada

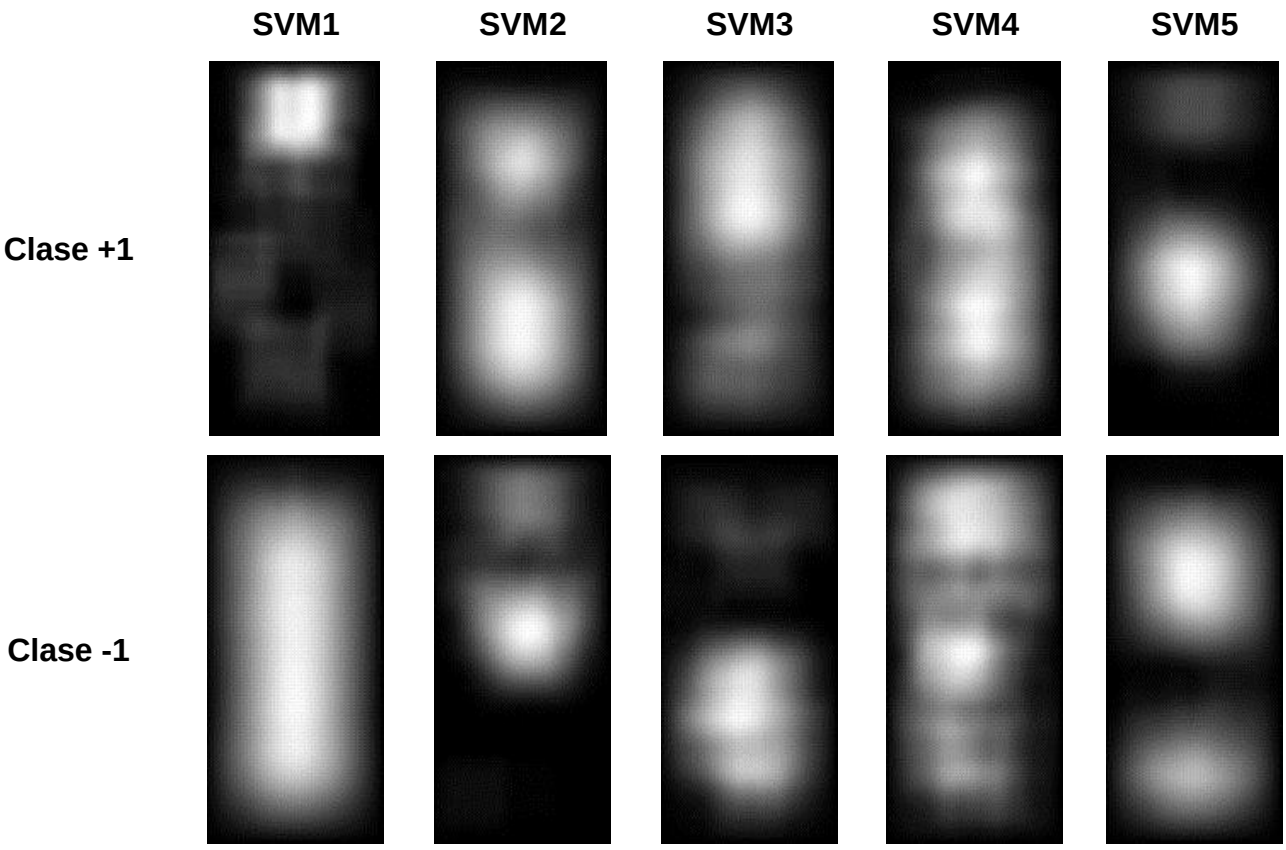


Figura 30: Resultados SVM

4.9 Aplicando ECOC a la salida de SVM

Lo que se ha mostrado anteriormente, son las salidas directas de los clasificadores. El problema es unir un problema donde tenemos varias extremidades, es decir, solucionar el problema de la multi-clasificación. Aquí es donde se vuelve a utilizar la técnica ECOC. De esta manera se obtiene un resultado a partir de un conjunto de información compuesto por múltiples clases.

Se aplicará ECOC directamente a las salidas de los clasificadores. Para ello se utiliza el resultado combinado de todos los clasificadores para obtener una salida conjunta. En este caso se utiliza el cálculo de la distancia Loss-Weighted y no se usa ningún tipo de peso dentro del sistema ECOC. Este cálculo nos devolverá aquella fila de la matriz ECOC que esté a menor distancia de la predicción realizada por los clasificadores. Una vez obtenida la clase a la que pertenece, finalmente se construye igual que antes una imagen de probabilidad, pero incrementando la probabilidad de la zona detectada por su extremidad calculada por ECOC.

El resultado que obtenemos será una imagen por extremidad, donde se indicará con más intensidad aquellas zonas donde se ha detectado la extremidad en cuestión. A continuación, se muestran algunas imágenes sobre esta segunda fase de procesado de los resultados de SVM en conjunción con ECOC.

4.10 Resultados visuales de la salida de SVM + ECOC

A continuación se muestran una serie de imágenes correspondientes a la probabilidad de que una extremidad se encuentre en una posición de la imagen. Estos resultados se aplicarán posteriormente a una última fase que consiste en el cálculo mediante GraphCut, donde se aplicará una segmentación de manera que se obtendrá la persona con sus extremidades pintadas de un color, aunque eso lo veremos más adelante.

Figura 31: Imagen inicial obtenida de GrabCut

Si pasamos la Fig. 31 por los clasificadores SVM y ECOC, los resultados que obtenemos por extremidad se encuentran reflejados en la Fig. 32:

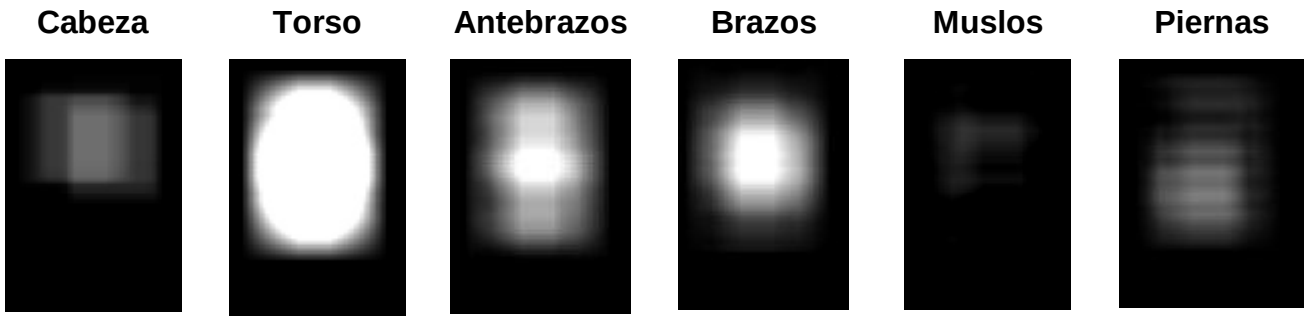


Figura 32: Detección por extremidad de SVM + ECOC

Como se puede observar, hay varias respuestas en función del clasificador. Por ejemplo, vemos que los muslos no los detecta muy bien, mientras que el torso lo marca claramente. En función de la imagen y de cómo esté entrenado el clasificador, se obtienen unos resultados un poco diferentes en cada caso. Por último, el conjunto de estas 6 imágenes se envían a la última fase del procesado, que es el GraphCut.

4.11 Resultados visuales del proceso de Graph Cut

En este punto se aplica el proceso GraphCut que consiste en obtener un resultado multi-clase a partir de unas entradas por extremidad con unos pesos. Para una explicación más extensa, consultar el PFC del Sr. Daniel Sánchez Abril, donde se detalla en profundidad la explicación del algoritmo.

Básicamente lo que queremos es unir las 6 imágenes de probabilidad que obtenemos desde los clasificadores SVM una vez hemos aplicado ECOC. Para realizar esta unión de la forma más óptima, se usa GraphCut, con el cual conseguimos unir las imágenes de probabilidad por extremidad y transformarla en una única donde se encuentran segmentadas de distinto color todas las extremidades detectadas.

Se han realizado varias pruebas con distintos sujetos, utilizando las salidas de SVM y ECOC. A cada extremidad se le ha asignado un color que la distingue de las demás y por lo tanto se puede ver claramente la segmentación.

Extremidad	Color asociado
Cabeza	Rojo
Torso	Verde
Antebrazos	Amarillo
Brazos	Azul
Piernas	Violeta
Muslos	Cian

Cuadro 13: Código de colores para Graph Cut

A continuación se muestran varias imágenes donde se puede observar la efectividad del método. Este proceso se realiza manualmente, ya que se deben ajustar una serie de parámetros que afectan directamente al resultado.

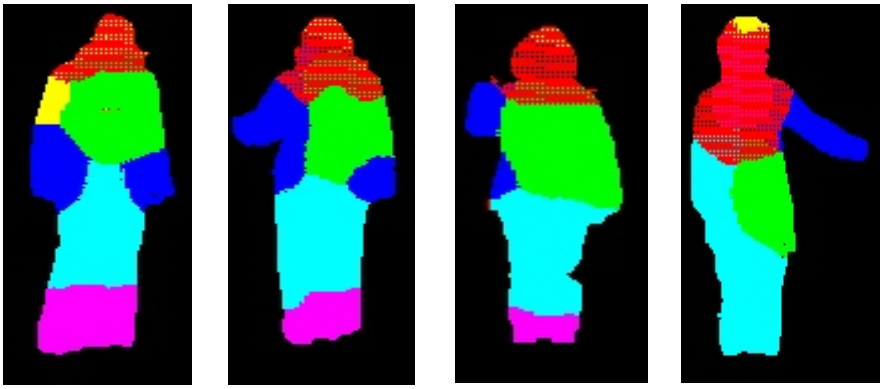


Figura 33: Resultado obtenidos desde Graph Cut

Como se puede ver en las imágenes de la Fig. 33, las tres primeras tienen mejores detecciones que la última. Se puede observar que los antebrazos son bastante más difíciles de detectar y casi siempre se solapa con la extremidad del brazo y el torso. En la última imagen se puede ver como los resultados no son tan buenos.

Se puede ver como detecta antebrazos en la región de la cabeza, y el torso no lo sitúa correctamente. Lo mismo pasa con la región de la cabeza, donde se expande hasta el torso y para finalizar, la no aparición de detecciones de piernas.

Como se puede ver, estos resultados dependen de los parámetros que se especifiquen, por lo que la segmentación es un proceso que tiene una gran parte manual de configuración de parámetros. Finalmente se ha podido ver el resultado de todas las fases del proyecto en profundidad y los resultados asociados a cada fase. Para finalizar, el último punto consiste en un resumen rápido donde se especifican todas las fases por las que pasan las imágenes de entrada hasta que tenemos los resultados finales.

4.12 Resumen de todo el proceso de testeo

Para aclarar finalmente todo el funcionamiento conjunto del sistema, vamos a describir todo el proceso desde el inicio hasta el final. Para ello vamos a ver rápidamente todas las fases por las que pasan los resultados y la forma en la que se van a tratar.

En el inicio de todo el proceso, tenemos una imagen que vamos a testear. Sobre esta imagen inicial, se va a realizar un sliding window a distintas escalas con el fin de detectar las extremidades independientemente del tamaño que estas tengan en la imagen. Esta primera detección se realiza mediante las cascadas Adaboost y los descriptores Haar. A los resultados de las cascadas Adabost se aplica una fase de ECOC con los pesos de las cascadas, con la finalidad de mejorar la detección añadiendo un sistema de corrección de errores. Estos pesos nos sirven para dar más importancia o menos a los clasificadores en función de la tasa de aciertos que estos tengan, es decir, de su fiabilidad.

De esta fase obtenemos una imagen de probabilidad en la que se indica la región donde hay más probabilidades de que exista una persona. Mediante el uso del sistema GrabGut, se obtiene aquella región donde la probabilidad es más alta, y por lo tanto se desestiman las demás (ya que se consideran falsos positivos). El resultado de esta fase será una imagen de negro donde se encontrará la persona correctamente segmentada, es decir, tendremos una imagen donde no hay fondo, simplemente la persona.

Una vez que tenemos a la persona segmentada, aplicamos la segunda fase de clasificación mediante SVM y HOG. En este punto, volvemos a realizar sliding window sobre la imagen segmentada. Al resultado de SVM incluimos una segunda etapa de ECOC, donde no hay pesos asociados, y se generan las 6 imágenes de probabilidad, una por extremidad.

Por último, estas 6 imágenes de probabilidad pasan a Graph Cut donde finalmente se realiza una segmentación de las extremidades y se obtienen los resultados expuestos anteriormente. La conjunción de todas las fases requiere tiempo de cálculo, y de momento no se podría realizar en tiempo real. Finalmente, en la Fig. 34 se muestra un diagrama donde se muestran todas las etapas más gráficamente

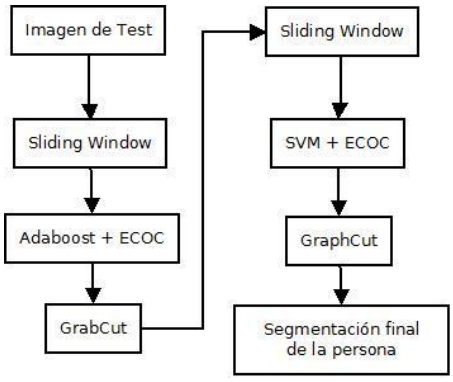


Figura 34: Proceso completo

5. Conclusiones y trabajo futuro

El campo de la Visión e Inteligencia Artificial está en constante progreso y día a día van apareciendo nuevas técnicas que mejoran todos los sistemas anteriores. El hecho de que sea un campo que no se ha explorado completamente, hace que haya grandes oportunidades para encontrar nuevos métodos que sirvan para resolver problemas complejos.

En nuestro caso, la segmentación de la persona ha sido compleja, dado que hemos pasado por muchas fases, y finalmente se han obtenido unos resultados bastante aceptables. En la mayoría de las imágenes resultantes que hemos obtenido, las predicciones se ajustaban bastante bien a la persona que estaba representada en la imagen de entrada. Quizás en este punto, nuestro gran hándicap sea el tiempo de ejecución. El hecho de entrenar las cascadas con tantos ejemplos, entrenar los clasificadores SVM, el testeo posterior, y que todo el sistema funcione correctamente ha sido una tarea bastante dura. En nuestro caso, estos resultados son válidos para el conjunto de las personas que forman los vídeos, es decir, los 9 actores. Si tuviéramos que contemplar el mundo real, tendríamos que entrenar los clasificadores con todo tipo de combinaciones de ropa, vestido… que pueden aparecer, por lo que esto da una idea de la complejidad del problema que se está tratando en este proyecto.

Las personas pueden ir vestidas de destina forma y esto nos afecta en el proceso de detección, ya que tendríamos que tener todas las posibles extremidades con todas las posibles combinaciones para tener un conjunto de datos que nos sirviera para detectar la mayoría de problemas. Por lo tanto, la cuestión aquí es la gran cantidad de potencia de cálculo que hace falta para hacer aprender a los clasificadores a distinguir ejemplos que aunque sean muy parecidos entre ellos, son distintos.

Después de realizar la primera etapa de los clasificadores SVM, una optimización que podría dar mejores resultados aún es el aprendizaje contextual. Este consiste en dadas las salidas de SVM, recorrer alrededor de las zonas etiquetadas como ground truth las etiquetas que más probabilidad nos han dado el primer nivel de SVM. Un ejemplo de estas metodologías son los métodos basados en Stacked Sequential Leraning.

El aprendizaje contextual puede ayudar al primer clasificador local a mejorar su poder de generalización. Esto es debido a que si en la primera fase existen errores o previsiones que no son del todo válidas, al intentar entrenar la segunda fase de SVM con estos datos, podríamos llegar a obtener unos mejores resultados ya que hemos aprendido el funcionamiento de la primera etapa.

La idea consiste en montar un correlograma en todas esas regiones donde se especifique para una región, un histograma que indique la probabilidad que hay de que sea una extremidad u otra. Esto al fin y al cabo se resume en un gran histograma donde se codifica para cada región la distribución de las detecciones. Si le asignamos la etiqueta del ground truth y lo entrenamos con SVM, podremos llegar a detectar las zonas mejor aún que como lo hace la primera fase de SVM.

En el marco de multi-clasificación ECOC también se podrían hacer pruebas con distintos o incluso combinaciones de clasificadores. Se podría haber realizado una estrategia 1-vs-1 en vez de montar un árbol binario, para poder simplificar el problema, aunque se a costa de aumentar el número de clasificadores que tienen que funcionar de forma conjunta.

En definitiva, este es un campo en el que hay mucho que investigar, y en el que se están realizando avances constantemente. Como se ha visto, se han utilizado distintas técnicas para realizar las pruebas de nuestro sistema, con unos resultados aceptables. La cuestión es ir mejorando y optimizando los métodos actuales para hacer que todo el proceso se ejecute más rápidamente. Ahora mismo no se puede ejecutar en tiempo real, dada la gran capacidad computacional que hace falta. Lo mejor es investigar y comprobar los resultados que se obtienen con la combinación de distintos métodos e intentar mejorar poco a poco todo el sistema.

6. Referencias

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

[2] http://bcnpcl.wordpress.com/research/error-correcting-output-codes/

[3] A. Hidalgo Chaparro, “Reconocimiento de Objetos Multi-clase Basado en Descriptores de Forma”, PFC UAB, 2008

[4] S. Escalera Guerrero, “Coding and Decoding Desing of Ecocs for Multi-class Pattern and Object Recognition, Tesis UAB, 2008

[5] S. Escalera, O. Pujol, P. Radeva, “Error-Correcting Output Codes Library”, 2010

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

[7] http://opencv-users.1802565.n2.nabble.com/gradients-amp-angles-td2191671.html

[8] http://blog.timmlinder.com/2011/07/opencv-equivalent-to-matlabs-conv2-function/#comment-267

[9] http://note.sonots.com/SciSoftware/haartraining/document.html

[10] http://www.computer-vision-software.com/blog/2009/11/faq-opencv-haartraining/

[11] http://gigadom.wordpress.com/category/haartraining/

[12] http://opencv.itseez.com/trunk/doc/user_guide/ug_traincascade.html

[13] http://se.cs.ait.ac.th/cvwiki/opencv:tutorial:haartraining

[14] http://vanderlin.cc/2008/12/haartraining-exploration/

[15] http://note.sonots.com/SciSoftware/haartraining.html

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

[17] http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

[18] http://en.wikipedia.org/wiki/Support_vector_machine

[19] http://www.dtreg.com/svm.htm

[20] http://www.cognotics.com/opencv/servo_2007_series/part_2/sidebar.html

[21] http://en.wikipedia.org/wiki/Haar-like_features

A. Contenido del DVD

En el DVD adjuntado con la memoria, se puede encontrar el código fuente de todas las aplicaciones que se han desarrollado en OpenCV, los datos utilizados para entrenar y testear y la memoria digitalizada en formato PDF. Las rutas donde encontrar esta información son las siguientes:

- **/MemoriaPFC.pdf**: Archivo con una copia de la memoria en formato PDF.

- **/Datos**: Incluye todas las imágenes procesadas para cada fase.

- **/CodigoFuente**: Incluye todo el código fuente de las aplicaciones que se han desarrollado. Incluyen los ejecutables para x64.

B. Instalación del Entorno

El desarrollo de los programas utilizados en el proyecto se ha hecho sobre la distribución Ubuntu 12.04. Básicamente hacen falta instalar los siguientes packages, mediante el gestor de paquetes apt-get:

- **Apt-get install libcv-dev**
- **Apt-get install libcv2.3**
- **Apt-get install opencv-doc**
- **Apt-get install eclipse**
- **Apt-get install eclipse-cdt**

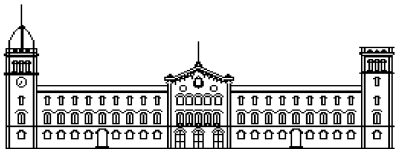
Posteriormente hace falta una configuración de Eclipse para poder usar las librerías en la extensión CDT. Para ello, hay que agregar en las rutas de inclusión de librerías y en las librerías las siguientes rutas:

Rutas de archivos de inclusión:

- /usr/include/opencv
- /usr/include/opencv2
- /usr/lib

Librerías OpenCV:

- Opencv_core
- Opencv_flann
- Opencv_legacy
- Opencv_ml
- Opencv_video
- Opencv_features2d
- Opencv_calib3d
- Opencv_objdetect
- Opencv_contrib
- Opencv_imgproc
- Opencv_highgui



GRAU D'ENGINYERIA INFORMÀTICA

UNIVERSITAT DE BARCELONA

Treball fi de carrera presentat el dia de de 200
a la Facultat de Matemàtiques de la Universitat de Barcelona,
amb el següent tribunal:

Dr. President

Dr. Vocal 1

Dr. Secretari

Amb la qualificació de: