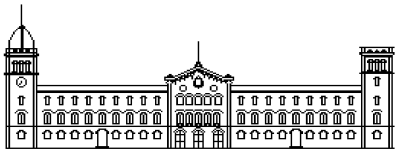

Trabajo final de grado

GRADO EN INGENIERÍA INFORMÁTICA

Facultad de Matemáticas
Universidad de Barcelona

RECONOCIMIENTO AUTOMÁTICO DE MODELOS DE AVIÓN EN ENTORNOS AEROPORTUARIOS

Adrián Infiesta Aguilá

Director: Sergio Escalera Guerrero
Realizado en: Departamento de Matemática Aplicada y Análisis. UB

Barcelona, 16 de septiembre de 2011

Agradecimientos

Quiero dar las gracias a todos aquellos que han estado a mi lado animándome a acabar este proyecto y otros muchos. En especial a Sergio Escalera, mi tutor del proyecto, que ha tenido una paciencia de santo y ha demostrado ser un gran profesional y mejor persona. También a mi familia que siempre han confiado en mi pese a que algunas veces yo no encontraba el momento de acabar. Y a todos aquellos amigos, ya sean de la universidad o amigos de toda la vida, que han entendido el trabajo que comporta un proyecto de este tipo.

Gracias a todos.

Resumen
En pleno siglo XXI la cantidad y diversidad de información con la que trata el ser humano es gigantesca. Gran parte de esta información es captada por sistemas automáticos de control, tales como cámaras fotográficas o dispositivos de video. Con el fin de agilizar la gestión de todos estos datos, surgen herramientas para el tratamiento y procesamiento de imágenes.

En este proyecto pondremos en práctica algunas de las más modernas herramientas para estos fines y orientaremos su uso al reconocimiento, identificación y clasificación de aviones comerciales que actualmente operan en aeropuertos de todo el mundo. Así mismo, analizaremos los resultados obtenidos y veremos como la implementación de diferentes algoritmos mejora la potencia de cálculo además de proporcionar resultados sorprendentes dejando abierta la puerta a futuras ampliaciones y mejoras.

Resum
En ple segle XXI la quantitat i la diversitat d'informació amb la que tracta l'ésser humà és gegantesca. Gran part d'aquesta informació és captada per sistemes automàtics de control, com càmeres fotogràfiques o dispositius de vídeo. A fi d'agilitzar la gestió de totes aquestes dades, sorgeixen eines pel tractament i processament d'imatges.

En aquest projecte posarem en pràctica algunes de les més modernes eines per aquestes finalitats i orientarem el seu ús al reconeixement, identificació i classificació d'avions comercials que actualment operen en aeroports de tot el món. Tanmateix, analitzarem els resultats obtinguts i veurem com la implementació de diferents algoritmes millora la potència de càlcul a més de proporcionar resultats sorprenents deixant oberta la porta per a futures ampliacions i millores.

Abstract
In the XXI century the number and diversity of information to manage is enormous. Much of this information is captured by means of automated systems, such as cameras or video devices. In order to streamline the management of this data, tools for image processing have been recently designed.

In this project we will put in practice some of the most modern tools and guide their use for recognition, identification and classification of commercial aircraft currently operating at airports around the world. We will also analyze the obtained results and will see how the implementation of different algorithms improves computational power while providing amazing results leaving the door open to future extensions and improvements.

Índice

1. - Introducción

El tratamiento de imágenes por computador es una técnica muy extendida en el mundo informático, y actualmente se aplica en campos muy amplios a la vez que diferentes. Las aplicaciones que se obtienen van desde las más sencillas técnicas de retoque fotográfico orientadas a mejorar las fotografías de revistas de moda, hasta la detección de puntos, zonas de interés e identificación de objetos concretos. Es por estos motivos, que cada vez existen más herramientas específicamente creadas para facilitar nuestros objetivos.

En este trabajo analizaremos, argumentaremos y resolveremos un problema de detección y clasificación de objetos aprovechando las herramientas destinadas al tratamiento de imágenes por computador. Más en concreto nos centraremos en un objeto en particular: aviones comerciales.

Actualmente en los aeropuertos españoles y en la gran mayoría de aeropuertos del resto del mundo, la tarea de distribución de pistas de aterrizaje y pistas tanto de aparcamiento como de puerta de enlace con las terminales, las efectúan manualmente los operarios del centro de control. Cada vez que se aproxima un vuelo con la intención de aterrizar, éste solicita permiso y espera respuesta. La autorización dependerá de varios factores, entre ellos de la disponibilidad de pistas en ese momento. Como cada modelo de avión dispone de una medidas y características diferentes, los operarios han de verificar la posibilidad de aceptar ese nuevo vuelo entrante para evitar colapsos. El problema surge cuando hay demasiados vuelos entrantes simultáneos y en tierra surgen problemas para despegar. Además, es común que las aerolíneas que operan estos vuelos utilicen aparatos diferentes al habitual, ya sea por motivos de mantenimiento o por imprevistos de última hora. Todos estos factores sobrecargan la capacidad de los operarios del centro de control, superando muchas veces sus límites de trabajo. La figura 1 muestra una vista aérea de varios aviones estacionados de manera ordenada en un aeropuerto.

Figura 1: aviones estacionados

Nosotros proponemos implementar un software capaz de analizar imágenes, reconocer objetos que sean de interés, en nuestro caso aviones, y de generar una salida determinando de qué modelo se trata. La idea final es que el programa sea autónomo y capaz de agilizar la gestión efectuada por los operarios, ayudando en el desarrollo de su cometido, aportando datos y verificando los introducidos manualmente.

Si hablamos de procesamiento de imágenes, podemos hacerlo en términos generales y divagar entre los diferentes ámbitos de aplicación. Si concretamos un poco más, podemos trazar líneas que separan un tipo de trabajos de otros. Como ya hemos comentado, existen muchas maneras de enfocar un problema y resolverlo. Así pues, puede que necesitemos la ayuda de unas librerías concretas para aplicar filtros sobre imágenes. Filtros de degradado, erosionado, o simplemente para quitar o disimular el ruido filtrado en las imágenes. Además, si añadimos un entorno gráfico para manipular estos filtros, podríamos hablar de un programa de retoque fotográfico, tal como el conocido Photoshop. Evidentemente, estos programas incluyen herramientas mucho más complejas, pero lo que el usuario final tiene a la vista, dista mucho de la implementación interna.

Por otro lado, podemos hablar de reconocimiento de objetos, reconocimiento de personas o incluso reconocimiento de rostros faciales y de sus partes más significativas. Para todas estas tareas se han creado librerías muy concretas y específicas. Los lenguajes de programación que las soportan también son de lo más variado, pero todos ellos trabajan mirando hacia el mismo horizonte: dotar al programador de las herramientas más útiles y sencillas de manejar, a la vez que potentes y rápidas.
En este contexto nacen aplicaciones como el reconocimiento de personas, para implementar, por ejemplo, sistemas de seguridad. La distinción entre un ser humano o un animal que deambula por un recinto puede ser un requisito fundamental para ciertas actividades de vigilancia. De igual manera, existen potentes softwares orientados a reconocer personas por su aspecto físico, en concreto su cara. Y si profundizamos más, reconocimientos oculares. Todos ellos trabajan en condiciones muy exigentes, y por ello necesitan las herramientas adecuadas.

Un ejemplo más sencillo, pero fácil de comprender, es el reconocimiento automático de matrículas de coches, por ejemplo, en la entrada de un parking. Disponer del software adecuado facilita una tarea algo engorrosa en caso de aglutinamiento del flujo de vehículos y, por consiguiente, de datos.

Nosotros nos basaremos en las herramientas de software gráfico y de visión por computador ya existentes. En concreto trabajaremos con las librerías de OpenCV, las cuales nos proporcionan potentes herramientas para entrenar clasificadores para la detección de objetos. Nuestra aportación será la de un nuevo software capaz de trabajar en un entorno real, aportando una solución justificada al problema que existe en lo relevante a la clasificación de objetos de unas características similares.

OpenCV son un conjunto de librerías que proporcionan funciones y programas completos relacionados con la visión por computador, siglas de las cuales deriva su nombre (Open Computer Vision). OpenCV dispone de tipos datos concretos adaptados para una mejor manipulación en el ámbito gráfico. Sus características nos facilitan la obtención de información y el tratamiento que le daremos. Es uno de los mayores conjuntos de librerías destinados a tal propósito. Además cuenta con una gran cantidad de ejemplos ya implementados de los cuales podemos obtener ideas y modificarlos para nuestros propósitos más concretos. También cuenta con una muy buena documentación donde consultar dudas.

OpenCV está escrito en lenguaje C y C++, un lenguaje robusto al mismo tiempo que potente. Gracias a este lenguaje podemos acceder a niveles muy cercanos al funcionamiento del hardware, y

al mismo tiempo lograr un nivel de abstracción propio de los mejores lenguajes orientados a objetos. Por todos estos motivos y muchos más, hemos escogido las librerías de OpenCV para tratar y desarrollar nuestro proyecto.

A lo largo de la memoria profundizaremos en los aspectos más técnicos e interesantes sobre todo el proceso. Esta memoria se divide en capítulos, cada uno de los cuales con sub-apartados. En el capítulo 2 trataremos el análisis del proyecto, pero sin llegar a la parte de implementación. Analizaremos la planificación y los costes del proyecto, los métodos empleados, tanto de hardware como de software necesarios. Efectuaremos un breve repaso a los requerimientos funcionales y a los módulos que vamos a implementar. En el capítulo 2 encontraremos el análisis de los procesos de adquisición de imágenes, detección de aviones, y otras actividades relacionadas con todo el proceso de aprendizaje y clasificación.

En el capítulo 3 trataremos el apartado de diseño de nuestra arquitectura y software. Facilitaremos los detalles de la implementación de los módulos más relevantes. Podremos ver algunos diagramas interesantes que faciliten la comprensión del funcionamiento del software.

Para finalizar presentaremos los resultados obtenidos con imágenes reales y las conclusiones a las que llegamos. Validaremos nuestra metodología seguida y compararemos los resultados esperados con los resultados obtenidos. También dispondremos de manuales de usuarios en los anexos, así como las guías de configuración del software necesario e implementado.

2. - Análisis

2.1 - Introducción

A lo largo del proyecto podemos identificar diferentes fases. A primera vista, podemos separar el trabajo en cuatro módulos principales:

- Aprendizaje
- Detección
- Descripción
- Clasificación

En la fase de aprendizaje trataremos la manera en que adquirimos imágenes adecuadas para el proceso de entrenamiento o propiamente dicho, de aprendizaje. Tal y como explicaremos en su debido apartado, veremos que son necesarios unos conjuntos de imágenes tanto positivas como negativas, y en una determinada proporción para optimizar los resultados.

En la parte de detección obtendremos la región del espacio donde se encuentra el avión, y adecuaremos las imágenes para poder trabajar con ellas.

Una vez localizado el avión, necesitaremos describir matemáticamente su presencia y características.

Finalmente clasificaremos una serie de aviones según ciertos criterios a partir de los datos obtenidos en los apartados anteriores.

A continuación efectuaremos un análisis de los métodos empleados, justificando el cómo y el por qué de su uso, así como mostrando ejemplos prácticos de los casos surgidos, y veremos como solventar algunos problemas de diseño. Incluye un repaso a los requerimientos de hardware y software necesarios.

2.2 - Requerimientos de hardware

Para desarrollar este proyecto no será necesario un gran equipo. Bastará con poder trabajar en un ordenador de sobremesa como el que podemos encontrar en cualquier despacho o laboratorio. Los requisitos mínimos los comentamos a continuación:

- Procesador Intel Pentium 4 @ 3,0 Ghz o superior. Aunque es posible funcionar con procesadores inferiores, hemos fijado el mínimo en uno a 3,0 Ghz para establecer unos tiempos de cómputo aceptables. No es necesario utilizar procesadores multi-núcleo, ya que el programa no ha sido diseñado para ello. Sin embargo, es posible realizar cambios y trabajar de manera concurrente para aprovechar este tipo de procesadores si se desea. Sobretodo seria interesante en proyectos con mayor número de imágenes y donde pueda darse el caso de trabajar con múltiples detecciones simultaneas.

- 2 Gb Memoria Ram o superior. Este tipo de memoria influye sobretodo en el momento del entrenamiento del clasificador. Según la documentación técnica [2], necesitaremos como mínimo 200Mb para funcionar correctamente, aunque es posible aumentar este parámetro.

Así pues, disponer de una cantidad sobrada de memoria ram es importante tanto para el aprendizaje como para los posteriores cálculos.

- Espacio en el disco duro. No fijaremos una cantidad mínima de espacio libre en el disco duro, ya que esto depende de cada usuario. Simplemente hemos de ser conscientes de tener libre tanto espacio como necesitemos para recopilar y guardar imágenes empleadas en el aprendizaje del clasificador.

- 500 Mb de tarjeta gráfica. Esta parte del hardware no influye en nuestra tarea más que en el momento de visualizar las imágenes. Por este motivo establecemos dicha cantidad de memoria como la recomendación mínima para trabajar adecuadamente.

2.3 - Requerimientos de software

En este apartado comentaremos que tipo de requisitos de software necesitamos tanto para realizar, como para ejecutar nuestro programa. En el apartado de anexos encontraremos información detallada de como instalar y configurar algunos de estos programas.

- Sistema Operativo Windows XP o superior. En nuestro caso hemos probado en trabajo en diferentes sistemas operativos, y en todos ellos funciona correctamente. También sería posible utilizar un sistema operativo basado en Unix, aunque complicaría un poco las configuraciones del entorno de programación.

- Visual Studio 2008 ha sido el entorno de programación elegido. Sus herramientas nos han parecido las más adecuadas, y su entorno cuenta con una gran cantidad guías y tutoriales que facilitan su comprensión.

- Librerías OpenCV 2.0 o superiores. En nuestro caso hemos trabajado con las versiones 1.8, 2.0, 2.1 y 2.2. Las diferencias consisten en las actualizaciones de sus bibliotecas. En el momento de la configuración es necesario saber con cuales se está trabajando. Se recomienda utilizar la versión 2.2 ya que por el momento se ha probado y es estable.

- Cmake 2.8. Es un conjunto de herramientas crear, compilar y testear software. Nuestra elección ha sido la versión 2.8.5 para las pruebas finales.

- Cygwin. Es una colección de herramientas que proporcionan un aspecto de un entorno Unix, incluyendo poder ejecutar comandos de este tipo, para plataformas Windows. La elección de la versión la dejamos a cargo del usuario final, para que pueda escoger la que mejor se adapte a su sistema operativo. De igual modo, recomendamos las instalación de todos los paquetes básicos más los relacionados con temas gráficos.

2.4 - Planificación

El objetivo de este apartado es proporcionar una rápida descripción sobre la metodología y la manera en la que se ha estructurado el proyecto, tanto desde un punto de vista de asignación de tareas como de previsiones sobre los trabajos que tenemos pensado realizar. No entraremos en detalles sobre los diferentes módulos que aparecen, ya que existen apartados que abordarán estos temas en profundidad. Hemos acompañado la lectura con tablas y gráficos que facilitan la comprensión de los objetivos que se describen.

2.4.1 - Planificación proyecto

A continuación mostramos una tabla en la figura 2 en la que podemos observar la planificación inicial hecha antes de comenzar el proyecto. También figura el tiempo estimado para cada tarea. Algunas de las tareas, se han podido hacer en menos tiempo del previsto, aunque ha sido compensado por otros contratiempos que han ido surgiendo a lo largo del trabajo.

Etapa	Sub etapa	Tareas	Dias
Análisis requerimientos			
	Requerimientos hardware		0
	Requerimientos software		
		Instalación software	1
		Verificación software	1
Diseño lógico			
	Especificaciones globales		2
	Reunión con director proyecto		
		Explicación de finalidades	1
	Contrastar datos y requisitos		
		Borrador definitivo	1
Diseño de la aplicación			
	Diseño de la aplicación		
		Diagrama de flujo	1
		Diagrama de clase	1
		Diagrama de secuencia	1
		Casos de uso	1
		Diseño entorno usuario	2
Implementar aplicación			
	Implementar código fuente		30
Extraer resultados			
	Obtener resultados		
		Recopilar resultados	3
		Comparar resultados	1
		Contrastar resultados	1
		Extraer conclusiones	1
	Presentar resultados		1
Redacción memoria proyecto			
	Redactar memoria proyecto		25
	Reunión con director proyecto		
		Explicación memoria	1
	Reescribir memoria definitiva		10
Presentación proyecto			
	Exposición proyecto ante tribunal		1

Figura 2: planificación del proyecto

2.4.2 - Planificación e implementación

Las figuras 3 y 4 muestran en los siguientes diagramas como ha quedado distribuido el trabajo general del proyecto según diferentes módulos en los que hemos separado las fases más destacadas. A lo largo del desarrollo de esta memoria se tocarán en profundidad muchos de estos bloques de trabajo, y explicaremos detalladamente como han quedado implementados.

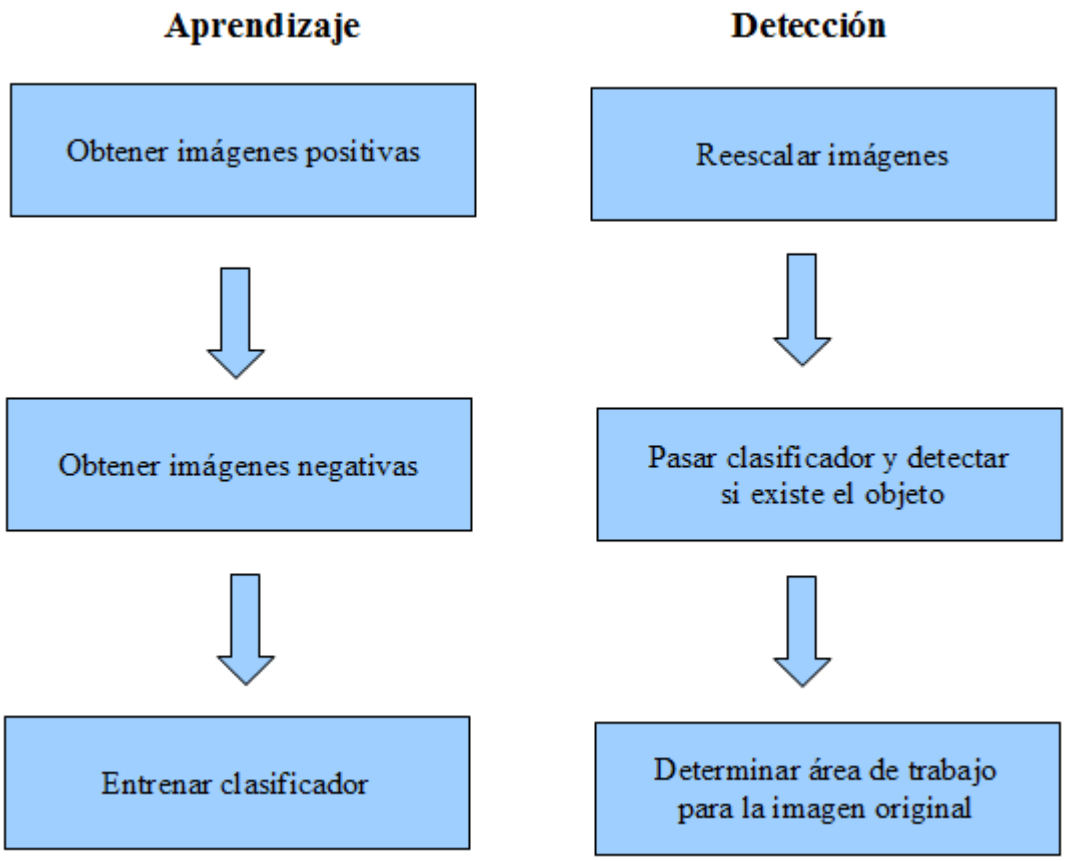

Figura 3: diagrama aprendizaje y detección

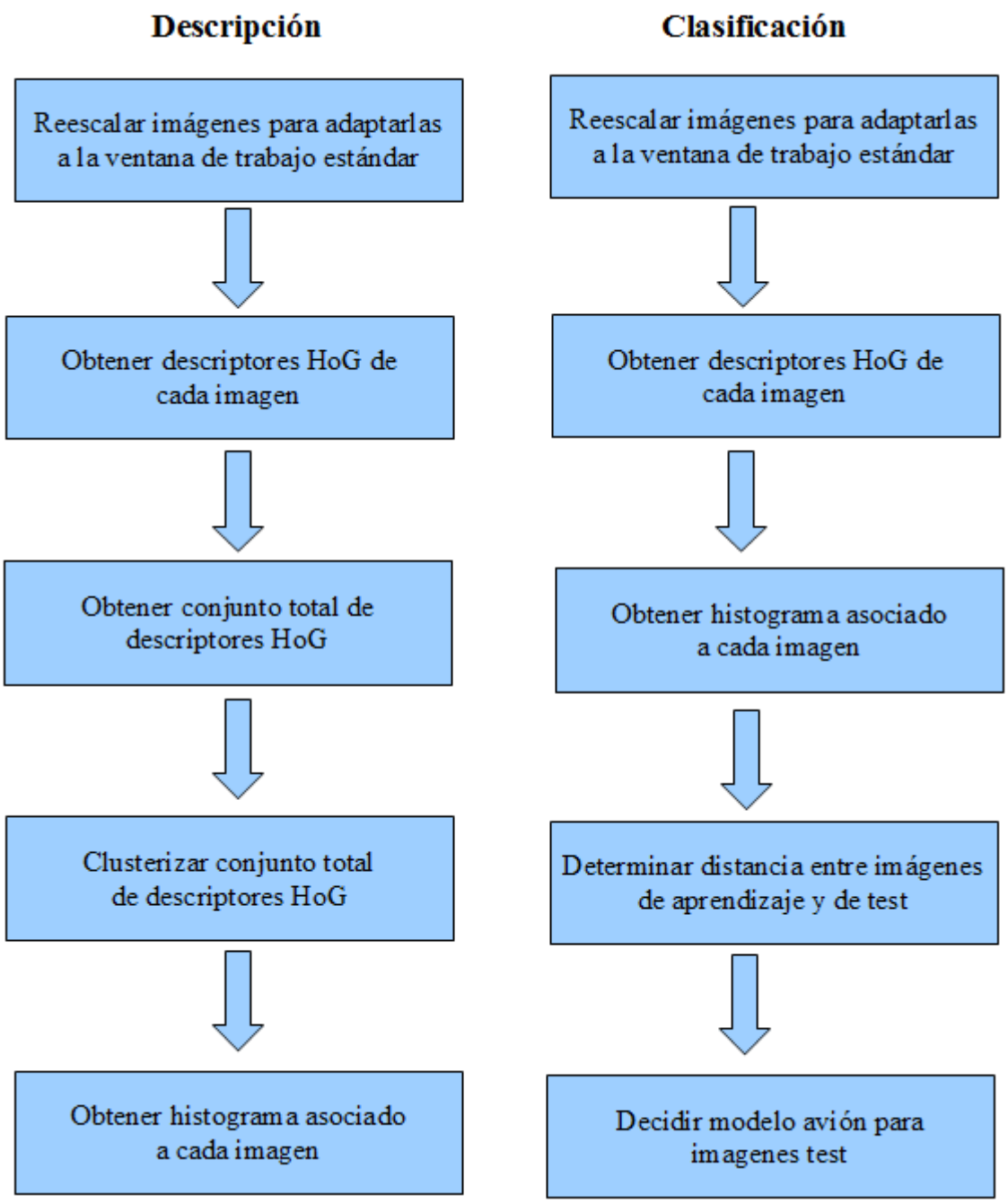

Figura 4: diagrama descripción y clasificación

2.5 - Aprendizaje

En este apartado explicaremos todo lo relativo a este proceso así como en que se fundamenta. El objetivo es aprender a utilizar las librerías de OpenCV que nos proporcionan potentes programas o funciones que se utilizan para el entrenamiento de clasificadores, en particular para sistemas de detección facial, que llaman *'HaarTraining'*.

La detección facial está muy extendida en el mundo del procesado de imágenes, y se aplica con diversas finalidades. La ventaja que obtenemos usando las librerías de OpenCV es que gracias a los recientes y constantes aportes, disponemos de librerías especializadas en esta tarea. Además, tenemos algunos ejemplos o samples sobre los que probar algunos pequeños programas. No obstante, no podemos conocer exactamente los parámetros que han utilizado los creadores de los clasificadores que nos proporcionan, ya que no nos facilitan información sobre dichos valores. Así pues, nuestro objetivo, es seguir sus pasos para crearnos un detector a medida, un clasificador preparado para reconocer objetos concretos, que para nuestro interés se tratara de aviones comerciales tal y como explicábamos anteriormente. Intentaremos dar los pasos lo más ajustados posibles a la realidad y para que se muestren de manera general, posibilitando la re-utilización para la creación de otros clasificadores distintos.

En la figura 5 podemos ver un ejemplo de detección facial aplicado a una imagen con múltiples objetivos a reconocer. Los cuadrados rojos son generados a partir de un clasificador que es capaz de reconocer patrones, en este caso rostros humanos.

Para la correcta comprensión de este apartado es necesario puntualizar los entornos en los que trabajaremos. Como ya hemos dicho, utilizaremos las librerías OpenCV, además de trabajar en un entorno con Visual Studio, acompañado de Cygwin, sobre un sistema operativo de Windows 7. Necesitaremos el programa Cygwin ya que haremos uso de múltiples comandos del tipo UNIX, no solo para la creación del clasificador, sino también para otros muchos propósitos. En los anexos explicaremos en detalle como instalar Visual Studio y las configuraciones apropiadas. Sobre Cygwin cabe comentar que se trata de una colección de herramientas desarrolladas para proporcionar un comportamiento similar a los sistemas Unix en Windows. Su objetivo es portar software.

También debemos puntualizar que normalmente un buen aprendizaje sobre un conjunto de imágenes utilizando el método de “haartraining” puede durar días dado el gran volumen de información que se ha de procesar. En nuestro ejemplo no será éste el caso, ya que el objetivo es conseguir un detector fiable, aunque no necesariamente perfecto. No obstante, se recomienda trabajar concurrentemente si se desea mejorar este proceso, a fin de evitar tiempos muertos de espera entre la generación del clasificador y la obtención de resultados.

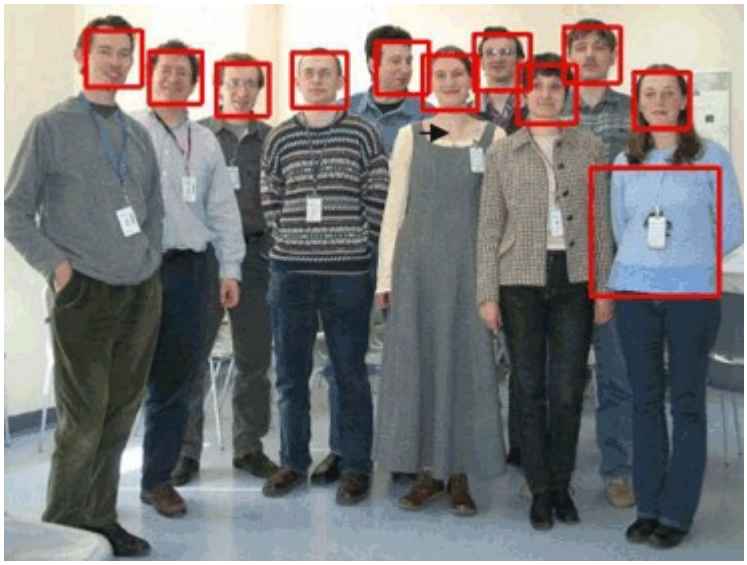

Figura 5: ejemplo de detección facial

2.6 - Preparación de la información

En este apartado hablaremos de como hemos de recopilar y almacenar el conjunto de datos y de información a tratar. Es muy importante hacer correctamente los pasos que se detallan a continuación para lograr un buen funcionamiento de las aplicaciones que surgen posteriormente. Hablaremos de todo lo referente al tipo de datos que se utilizarán y de sus características. Para facilitar la comprensión hemos dividido esta parte en dos sub-apartados, distinguiendo entre dos tipos de imágenes.

2.6.1 - Imágenes positivas

En primer lugar necesitamos recoger lo que llamaremos 'imágenes positivas'. Estas imágenes serán las que contengan únicamente los objetos de interés, en nuestro caso aviones. Kuranov et. al. [1] menciona que se suelen utilizar 5000 imágenes positivas, las cuales derivan de un conjunto de 1000 imágenes diferentes con patrones del objeto a reconocer. Para obtener este número de imágenes debemos recorrer a bases de datos existentes. En internet podemos encontrar diferentes bases de datos, sobretodo en cuanto a patrones de caras humanas. En nuestro caso, las imágenes positivas han sido proporcionadas por colaboradores de la universidad que trabajan en el aeropuerto del prat, imágenes facilitadas también por AENA.

Como ya hemos comentado, nuestro clasificador ha de ser suficientemente robusto, aunque no perfecto. Por este motivo únicamente hemos utilizado 207 imágenes positivas, correspondientes a 5 modelos distintos de aviones. En cualquier caso, si deseamos utilizar más imágenes positivas y no disponemos de ellas, podemos generarlas a partir de las que ya tenemos. Aunque no es nuestro caso, dejamos indicado la manera de hacerlo para futuras ampliaciones. Esto se consigue aplicando distorsiones sobre una misma imagen. Podemos utilizar esta función si las opciones -img, -bg y -vec son especificadas.

-img <una imagen positiva>
-bg <colección de negativos>
-vec <nombre del fichero contenedor de las muestras generadas>

Podemos ver un ejemplo a continuación:

```
$ createsamples -img face.png -num 10 -bg negatives.dat -vec samples.vec -maxxangle 0.6 -maxyangle 0 -maxzangle 0.3 -maxidev 100 -bgcolor 0 -bgthresh 0 -w 20 -h 20
```

En el caso de disponer de suficientes imágenes positivas, debemos crear un archivo .vec contenedor de dichas imágenes Debemos ejecutar las siguientes opciones:

```
-info <archivo de positivos>
-vec <nombre del archivo de salida generado>
```

el formato del archivo de positivos debe ser como sigue:

```
[filename] [# of objects] [[x y width height] [... 2nd object] ...]
[filename] [# of objects] [[x y width height] [... 2nd object] ...]
[filename] [# of objects] [[x y width height] [... 2nd object] ...]
…
```

donde (x,y) son las coordenadas de la esquina izquierda superior del objeto, y la esquina superior izquierda de la imagen tiene como coordenadas (0,0), tal que un ejemplo seria:

```
img/img1.jpg 1 140 100 45 45
img/img2.jpg 2 100 200 50 50 50 30 25 25
img/img3.jpg 1 0 0 20 20
```

Entonces podríamos ejecutar el comando:

```
$ createsamples -info samples.dat -vec samples.vec -w 20 -h 20
```

En la figura 6 podemos ver como son nuestras imágenes positivas, contenedoras del objeto deseado de aprender, un avión. Además, si nos fijamos, veremos como los aviones aunque parecidos, son algo distintos, motivo que deberemos trabajar más adelante en el apartado de clasificación.

Figura 6: ejemplo de imágenes positivas

2.6.2 - Imágenes negativas

Las imágenes negativas que necesitaremos serán imágenes aleatorias, de temática diversa, pero asegurándonos de que no contengan ningún objeto de interés como el que estamos preparando para ser reconocido. Kuranov [1] menciona que ellos utilizaron 3000 imágenes negativas. En nuestro caso, también disponemos de un gran número de imágenes negativas. Utilizaremos unas 2000.

Necesitamos tener las imágenes guardadas en un directorio, o en varios, pero lo suficientemente simple como para crear un archivo del tipo .dat que contenga dicha información. Podemos crearlo fácilmente con los siguientes comandos:

```
$ cd [nuestro directorio de trabajo]
$ find [image dir] -name '*.[image ext]' > [description file]
```

En la figura 7 observamos un par de imágenes negativas completamente aleatorias.

Figura 7: ejemplo de imágenes negativas

2.7 - Entrenamiento

Una vez disponemos del set de imágenes positivas y negativas podemos dar paso a la creación del fichero .xml como resultado de entrenar a nuestro clasificador. Para este cometido, disponemos de la utilidad 'haarTraining'. Se trata de la implementación de un algoritmo de aprendizaje conocido como AdaBoost en cascada de clasificadores, cuyo nombre proviene del inglés "Adaptative Boosting", basado en el método formulado por Yoav Freund y Robert E. Schapire [4]. Es un meta-algoritmo que puede ser usado juntamente con muchos otros algoritmos de aprendizaje para mejorar su rendimiento.

El funcionamiento de AdaBoost se basa en la combinación de clasificadores débiles para acabar obteniendo un clasificador fuerte. AdaBoost utiliza un modelo de aprendizaje llamado "decision stump", que consiste en árboles de decisión de un nivel [5]. El árbol de este tipo tiene un nodo raíz que conecta con dos nodos terminales. En cada etapa de decisión, se hace una predicción basada en el valor del dato de entrada. Dependiendo del valor de este dato, se construye los dos nodos hijos, uno de los cuales corresponde a la categoría elegida correctamente, y el otro al resto de categorías posibles. AdaBoost utiliza este procedimiento para crear clasificadores débiles, a partir de los cuales acabará resultando en clasificador fuerte final.

Para cada etapa se llama repetidamente a una serie de clasificadores débiles del total de clasificadores. En cada llamada se actualiza los pesos de los clasificadores débiles indicando la importancia de los ejemplos de datos a clasificar, a partir de los cuales se crearán los nuevos clasificadores también débiles. En cada ronda, los pesos de los ejemplos incorrectamente clasificados se incrementan, o alternativamente, los pesos de los ejemplos correctamente clasificados se decrementan, logrando así que los nuevos clasificadores se centren más en estos ejemplos.

En la figura 8 podemos ver como los clasificadores h1 y h2 no son perfectos, pero AdaBoost los combina para obtener un buen clasificador. AdaBoost es adaptativo en el sentido que los clasificadores construidos posteriormente se ajustan a favor de los casos mal clasificados anteriormente. Igual que muchos otros algoritmos, AdaBoost es sensible al ruido en los datos y a los valores atípicos.

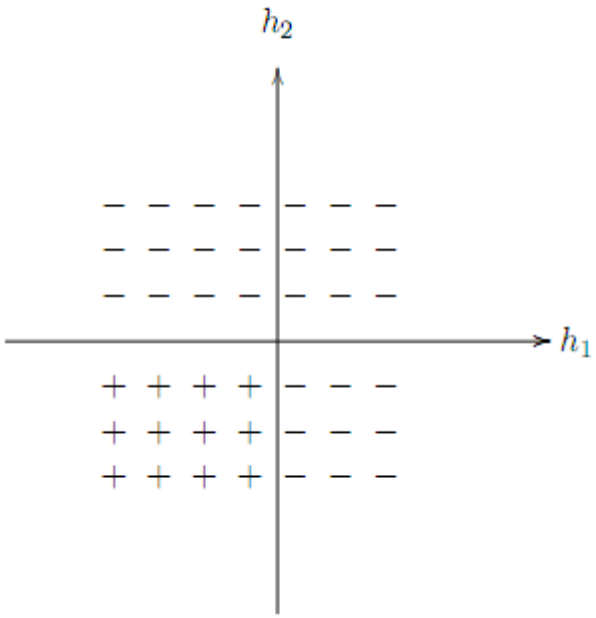

Figura 8: h1 y h2, clasificadores débiles de AdaBoost

A continuación explicamos algunos de los parámetros más interesantes a tener en cuenta:

-nstages indica el número de estados en el que será entrenado el clasificador. Como ejemplo, asumiendo que el set de imágenes es representativo para el aprendizaje, podemos esperar un ratio de falsa alarma sobre $0.5^{20} \approx 9.6e{-}07$ y un hit rate de $0.999^{20} \approx 0.98$.

Entonces podemos utilizar ejemplos de 20x20, nsplit =2, nstages = 20, minhitrate = 0.9999, maxfalsealarm = 0.5 y weighttrimming = 0.95.

-sym se utiliza cuando el objeto no tiene simetría vertical. En nuestro caso, dado que las fotos de los aviones siempre serán des de el mismo punto de vista, enfocándolos frontalmente, podemos contar con que sí que hay simetría vertical, aunque no en todos los casos sea 100% cierto. Si activamos esta opción, se agiliza el proceso de aprendizaje, ya que solo trabajara en una mitad de la imagen.

-mem es la memoria disponible para el precálculo especificado en Mb. Por defecto son 200Mb [2], así que es bueno incrementar el parámetro si se dispone de más memoria.

También hemos de tener en cuenta, que aunque incrementemos el número de 'stages', el aprendizaje puede acabar en un estado intermedio cuando exceda el número deseado de hit rate mínimo. Igualmente puede suceder que finalice por que todas las imágenes de ejemplo son rechazadas. En este caso hay que incrementar en número de imágenes.

Una vez conocemos los parámetros que nos interesan, mostramos un ejemplo de como ejecutar el entreno:

```
$ haartraining -data haarcascade -vec samples.vec -bg negatives.dat -nstages 20 -nsplits 2 -minhitrate 0.999 -maxfalsealarm 0.5 -npos 7000 -nneg 3019 -w 20 -h 20 -nonsym -mem 512 -mode ALL
```

A continuación listamos todas las opciones disponibles:

```
Usage: ./haartraining
  -data <dir_name>
  -vec <vec_file_name>
  -bg <background_file_name>
  [-npos <number_of_positive_samples = 2000>]
  [-nneg <number_of_negative_samples = 2000>]
  [-nstages <number_of_stages = 14>]
  [-nsplits <number_of_splits = 1>]
  [-mem <memory_in_MB = 200>]
  [-sym (default)] [-nonsym]
  [-minhitrate <min_hit_rate = 0.995000>]
  [-maxfalsealarm <max_false_alarm_rate = 0.500000>]
  [-weighttrimming <weight_trimming = 0.950000>]
  [-eqw]
  [-mode <BASIC (default) | CORE | ALL>]
  [-w <sample_width = 24>]
  [-h <sample_height = 24>]
  [-bt <DAB | RAB | LB | GAB (default)>]
  [-err <misclass (default) | gini | entropy>]
  [-maxtreesplits <max_number_of_splits_in_tree_cascade = 0>]
  [-minpos <min_number_of_positive_samples_per_cluster = 500>]
```

Cabe decir que el fichero de salida .xml es generado cuando el proceso se completa al 100%. Si deseamos convertir una lista de ficheros de salida de pasos intermedios en el fichero .xml antes de que termine, podemos hacerlo con el software proporcionado, y con el siguiente comando:

```
$ convert_cascade --size="<sample_width>x<sampe_height>" <haartraining_ouput_dir> <ouput_file>
```

Por ejemplo, de la siguiente manera:

```
$ convert_cascade --size="20x20" haarcascade haarcascade.xml
```

2.8 - Test

Una vez disponemos del fichero .xml de aprendizaje generado, es un buen momento para ver su rendimiento. Podemos hacerlo usando la utilidad 'performance'. Esta utilidad trabaja durante la detección de una manera sencilla:

una ventana se va moviendo píxel a píxel por toda la imagen a diferentes escalas. Las escalas varían cada vez hasta que exceden las dimensiones de la imagen en al menos una dimensión. Las detecciones de múltiples objetos cercanos (si los hay) se mezclan. Entonces se crea lo que se llama ROCS (Receiver Operating Curves) variando en número de objetos detectados por imagen antes de mezclarlos en un resultado final de detección. Durante el experimento solo se cambia un parámetro al mismo tiempo. El mejor parámetro encontrado se usa en los subsiguientes experimentos [1].

Pero lo que nos interesa es poder ejecutarlo y ver los resultados. Lo hacemos así:

```
$ performance -data haarcascade -w 20 -h 20 -info tests.dat -ni

$ performance -data haarcascade.xml -info tests.dat -ni
```

Obtendremos una salida parecida a la siguiente, donde encontramos el número de hits en imágenes correctamente detectadas, el número de missed en imágenes incorrectamente detectadas, así como las falsas alarmas o falsos positivos.

```
+==================================+======+======+======+
|          File Name          | Hits |Missed| False|
+==================================+======+======+======+
|tests/01/img01.bmp/0001_0153_005|    0|    1|    0|
+------------------------------+------+------+------+
....
+------------------------------+------+------+------+
|                   Total|   874|   554|    72|
+==================================+======+======+======+
Number of stages: 15
Number of weak classifiers: 68
Total time: 115.000000
15
         874    72     0.612045       0.050420
         874    72     0.612045       0.050420
         360    2      0.252101       0.001401
         115    0      0.080532       0.000000
         26     0      0.018207       0.000000
         8      0      0.005602       0.000000
         4      0      0.002801       0.000000
         1      0      0.000700       0.000000
         ....
```

2.9 - Dectección aviones

Una vez hemos finalizado el 'training' o entrenamiento del clasificador de objetos, ahora es momento de ver como lo podemos utilizar. Las librerías de OpenCV nos proporcionan algunos clasificadores ya implementados, la mayoría de ellos para reconocer caras. Una prueba interesante es comprobar como actúa un clasificador ya hecho con una imagen cualquiera. Para ello hemos tomado la conocida imagen “lena.jpg”, la cual muestra el rostro de una bella chica. El resultado es el que se muestra en la figura 9, que como podemos ver, acierta en localizar concretamente la cara de Lena.

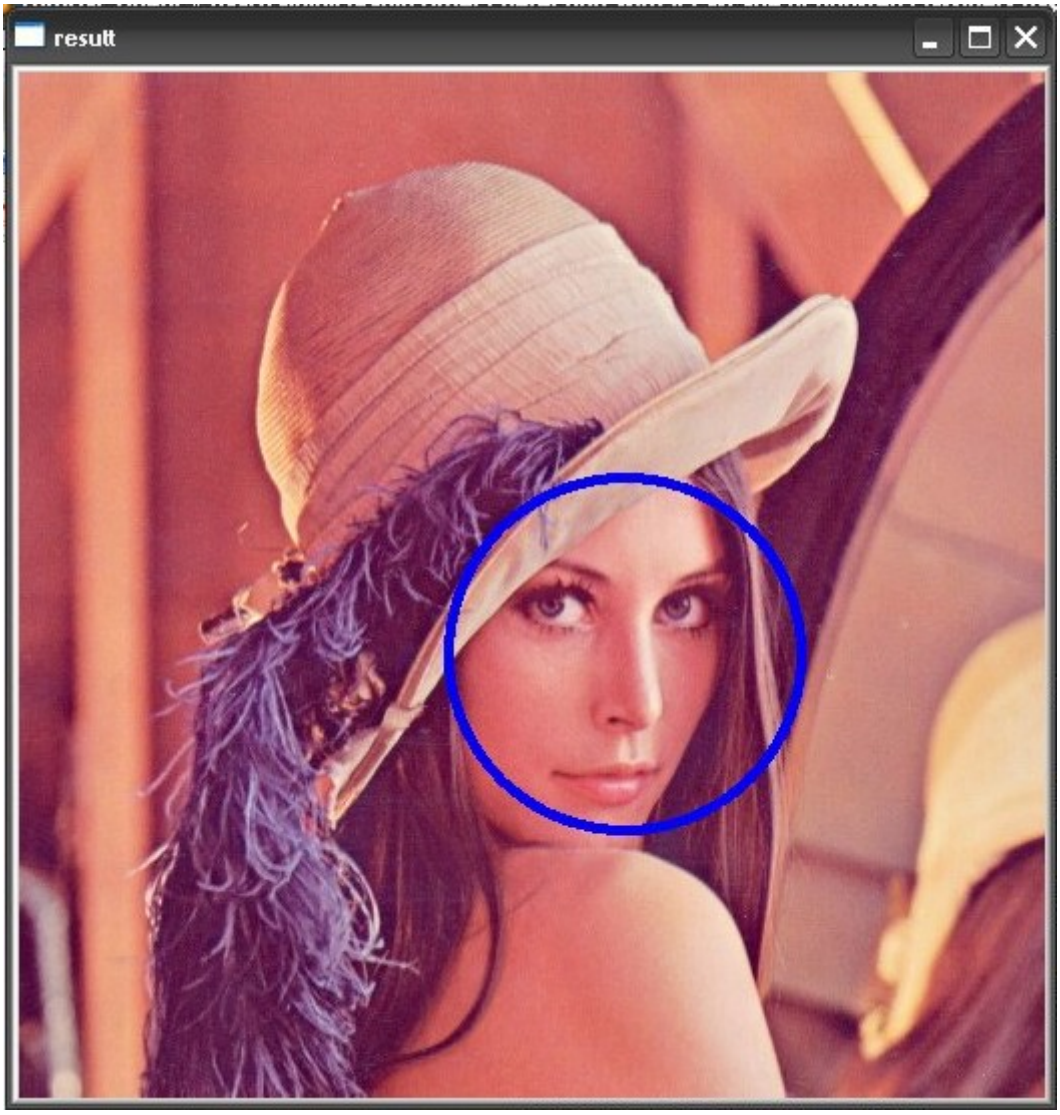

Figura 9: Detección facial sobre Lena

De esta manera, nuestro objetivo será emular el proceso ahora aplicado a la detección de aviones, utilizando nuestro clasificador previamente entrenado. La utilidad “facedetect” que se ha utilizado en la imagen de Lena, se basa en lo que conocemos como “haar-like features”. La idea de estas interpretaciones viene relacionada con el tipo de clasificadores de tipo “cascada” o débiles que se han utilizado en el aprendizaje, para generar un clasificador fuerte que nos permita una detección ajustada.

“Haar-like” tiene como principio trabajar en áreas de la imagen tales como rectángulos de un determinado tamaño, buscando similitudes o patrones previamente entrenados en el clasificador que indiquen la existencia de una parte conocida del objeto a detectar. “Haar-like” considera las regiones adyacentes a los rectángulos de una región especifica, suma la intensidad de los píxeles y calcula de diferencia entre ellas. Esta diferencia sirve para categorizar las diferentes secciones de la imagen. En la figura 10 vemos un ejemplo de rectángulos de trabajo de estas interpretaciones. De esta manera, encontraremos patrones en áreas conocidas como por ejemplo los ojos, la nariz o la boca. La clave de usar “haar-like features” es la velocidad de cómputo respecto a otros sistemas.

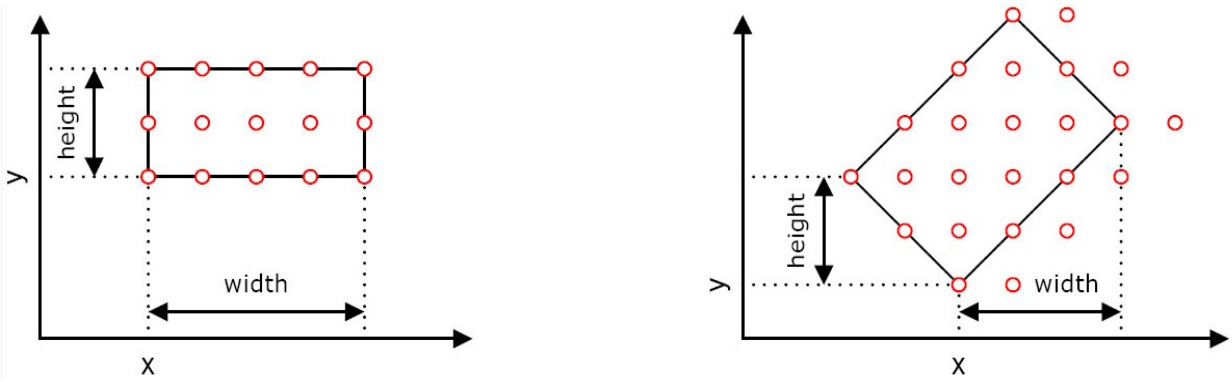

Figura 10: Haar-like features

Primeramente, para enmarcar un avión, necesitaremos una ventana rectangular, y no circular, como la que aparece en la imagen. Esto no es problema, ya que, tal y como veremos más en profundidad en el correspondiente apartado de diseño, únicamente cambiando algún parámetro obtenemos una ventana adecuada. El problema que surge, es que un avión, a no ser que sea un modelo muy particular, tiene unas proporciones rectangulares, y eso también ocurre en nuestra vista frontal. El algoritmo de detección está orientado a describir objetos dentro de una ventana de dimensiones circulares, como en el ejemplo de la figura 9, o bien cuadradas, como podemos ver en ejemplo de la figura 5. Si aplicamos la utilidad 'facedetect' junto a nuestro clasificador, podemos observar como se forma un problema de múltiple detección, a la par que no encaja el objeto en el área detectada.

En la figura 11 se observa como el avión es detectado correctamente por el círculo rosa, pero no encajado completamente, dado que el círculo ha de mantener sus propiedades. A la vez, debido a que estamos utilizando un algoritmo de multi-detección, debido a los posibles errores existentes, ya sea por falta de imágenes positivas, o por demasiado ruido en la imagen, se detectan incorrectamente zonas que no son del avión, como muestran los círculos de la gama de colores azul.

Por otra parte, los círculos rojo, calabaza y amarillo, muestran que reconocen las partes de los motores y de las alas como propias de un avión. Ahora el problema radica en juntar todo esto y obtener una única ventana de detección.

Una solución es activar un flag dentro de las opciones que indica detectar "únicamente el objeto más grande", una opción completamente válida en nuestro trabajo, ya que sabemos con certeza que trataremos imágenes en las que aparecerán aviones de forma independiente. Ahora bien, para solventar en tema de la ventana cuadrada, hemos de efectuar un pequeño rodeo. Es posible obtener una ventana rectangular, pero hay que actuar paso a paso.

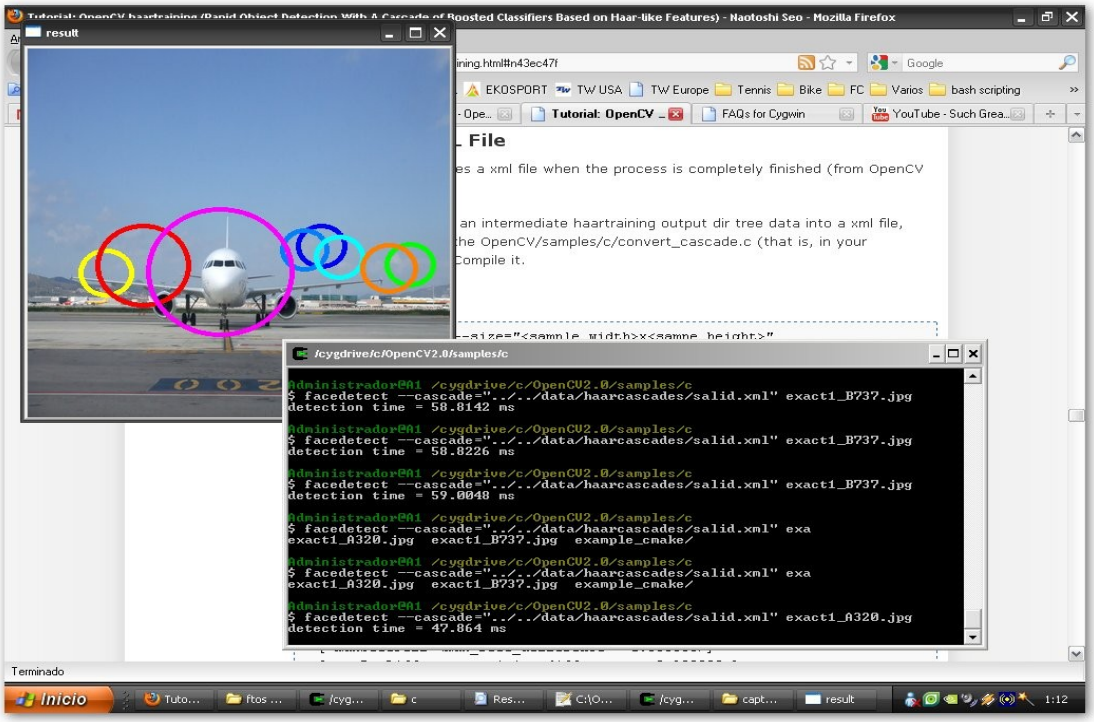

Figura 11: detección incorrecta

Así pues, la solución adoptada pasa por conseguir imágenes en las que el avión aparezca en proporciones cuadradas. Para tal fin, necesitamos herramientas sencillas tratamiento de imágenes. Para ello nos serviremos otra vez de los comandos Unix.

Primeramente determinaremos las proporciones del objeto, ya que éstas variarán en función de cada fotografía. Seguidamente calcularemos el tamaño de la nueva imagen para que conserve dichas proporciones. Este método se explicará en profundidad en el apartado de diseño. Una vez tengamos esa imagen, ahora sí que podremos detectar correctamente sobre ella una ventana cuadrada. Además, como conoceremos las proporciones reescaladas, podemos invertir el proceso, y reescalar la ventana para trabajar con la imagen original. Véase un ejemplo de imagen reescalada para contener proporciones cuadradas según el avión en la figura 12.

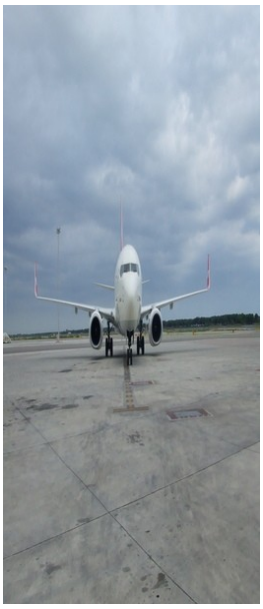

Figura 12: imagen avión redimensionado

2.10 - Describir aviones

En este apartado profundizaremos en que métodos utilizamos para describir una imagen en la que aparece un avión, y más concretamente, la región en la que se sitúa el avión. Emplearemos métodos matemáticos para generar datos que posteriormente puedan ser interpretados.

Hemos de partir de la idea base de que ya disponemos de un clasificador previamente entrenado, y que hemos creado un set de imágenes para ser procesadas de acuerdo a unos criterios fundamentales, como la comentada proporción cuadrada que deben tener los aviones en las imágenes redimensionadas para poder ser detectados satisfactoriamente. Una vez tenemos claro la forma en la que se presentaran los datos de entrada, debemos buscar una manera eficiente de interpretarlos. Y de hecho, con unas librerías tan amplias como las de OpenCV surgen varias opciones.

Una primera idea nos remite a la descripción mediante tablas de datos, como por ejemplo, histogramas según los colores de cada imagen. Histogramas basados en el color que adopta cada píxel. Determinar las cantidades acumuladas en imágenes conocidas y establecer un patrón de como debe de ser para imágenes positivas que contengan un avión. Pero aparece el problema de poder confundir imágenes que no tengan ninguna similitud. Fotografías que contengan exactamente la

misma cantidad de un cierto color que otra, pero que su representación visual no se parezca en nada a la del objeto deseado. Ocurre lo mismo si trabajamos con histogramas de intensidades. Por este motivo descartamos esta técnica.

La necesidad de encontrar un método que describa la imagen teniendo en cuenta la correlación entre los píxeles nos lleva a la idea propuesta por Navneet Dalal y Bill Triggs [3] de utilizar Histogramas de Gradientes Orientados para la detección de personas. La idea es extraer los descriptores HoG (Histogram of Oriented Gradients) de una región concreta, y contar el número de ocurrencias de una magnitud en una porción localizada de la imagen. El método se parece mucho a otros, como por ejemplo el SIFT (Scale Invariant Feature Transform), o el EOH (Edge Orientation Histograms), pero se diferencia por que se calcula en una densa parrilla de celdas espaciadas de forma uniforme y utiliza overlapping local en la normalización del contraste a fin de mejorar su efectividad.

La idea básica que se esconde detrás de la teoría es que un objeto dentro de una imagen puede ser descrito por la distribución de la intensidad de sus gradientes o la dirección de los bordes. La zona en la que aplicar el cálculo se divide en pequeñas regiones conectadas llamadas celdas, y para cada celda se calcula un histograma de las direcciones de gradientes para los píxeles de la celda. La combinación de estos histogramas representa el descriptor. Para mejorar la precisión, se puede aplicar otra vez la misma metodología en regiones de celdas, llamadas bloques, y normalizando los valores obtenidos para cada bloque. En la figura 13 podemos ver un ejemplo visual representando los gradientes y como extraerlos para cada píxel y sus celdas. La implementación la comentaremos en profundidad en el apartado oportuno de diseño.

Ahora que ya tenemos el método apropiado para extraer información sobre nuestras imágenes, sólo falta comentar cómo procederemos. Sin entrar en detalles de la implementación de las funciones que actúan, debemos explicar que para un buen funcionamiento del proceso, es necesario disponer de las imágenes tanto a proporción, como las originales, y conocer sus factores de reescalado. Además, para que la descripción del total de aviones sea coherente, hemos decidido describirlos a todos dentro en una misma ventana o marco de trabajo, con el mismo tamaño para todas las fotografías. Esto significa que deberemos volver a calcular y reescalar las imágenes, cada una a unas dimensiones particulares. Así, para cada objeto detectado, tendremos el mismo número de 'features' o descriptores. Entonces ya podemos decir que cada avión lo describimos a base de descriptores HoG, y como ya veremos, nos servirán para tratar la información sobre ellos.

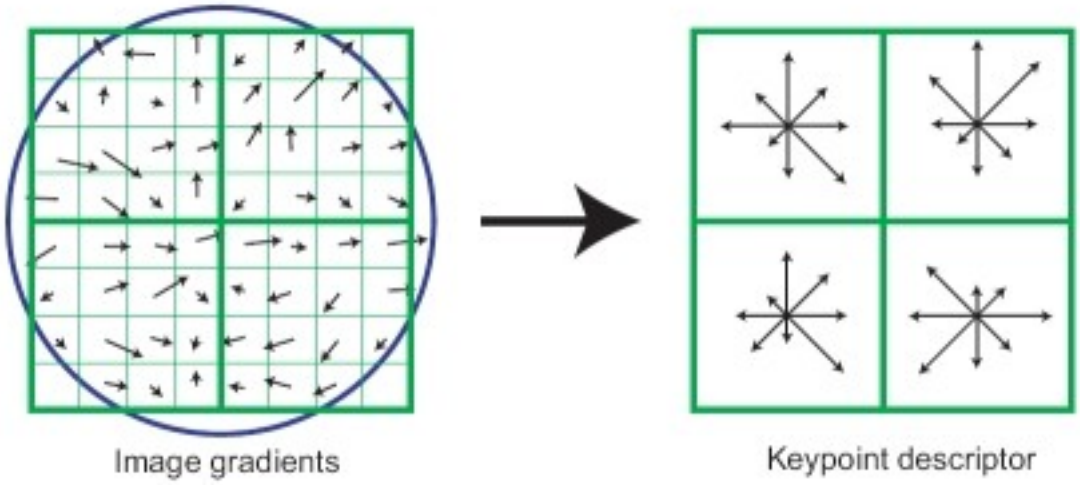

Figura 13: HoG features para una región

Los descriptores HoG nos proporciona información tal como los cambios de intensidad debido a los contornos o bordes de una imagen. Al tener en cuenta la relación con sus zonas vecinas y colindantes, es posible reconocer cuando existe una frontera entre un objeto y otro. De esta manera, podremos identificar objetos de siluetas más suaves o más pronunciadas. Los descriptores serán una buena manera de gestionar la información particular de cada modelo de avión, independientemente de su tamaño y sus colores, y fijándonos más en su relación con el entorno, distinguiendo los cambios más pronunciados.

2.11 - Clasificar aviones

Uno de los objetivos de este trabajo, es precisamente lograr clasificar con éxito nuestros objetos de interés. De esta manera, debemos prestar especial atención al diseño de este apartado.

Una vez que hemos descrito los aviones, el siguiente paso es clasificarlos. Podemos distinguir entre los términos de 'reconocer' aviones, o 'clasificar' aviones. Cuando hablamos de reconocer, lo hacemos refiriéndonos al acto de detectar un avión dentro de una imagen, y validar esa imagen como positiva. Además, cuando reconocemos un avión, hemos de ser capaces de determinar las coordenadas en las que se encuentra dentro de la fotografía.

Si hablamos de clasificar aviones, nos referimos a distinguir entre varios tipos de aviones, en determinar que modelo de avión es concretamente. Se trata de distinguir entre diferentes clases de un mismo objeto, en nuestro caso sobre aeronaves.

Como ya hemos comentado en el apartado de describir aviones, hemos creado un módulo capaz de extraer información sobre las regiones que contengan los objetos de interés. La información se obtiene en forma de descriptores HoG, y sabemos que para cada imagen obtendremos el mismo número de descriptores. Si somos capaces de extraer los descriptores de una imagen, seremos capaces de iterar esta función y extraer los descriptores para un conjunto de fotografías. Tomaremos como referencia un número concreto de imágenes positivas correctamente detectadas. A partir de ese conjunto, confeccionaremos la información relevante a todos ellos en forma de descriptores HoG siguiendo las técnicas antes mencionadas. Al finalizar el proceso, obtendremos una gran matriz conteniendo toda esta cantidad de información.

Una vez disponemos de la información relevante de un cierto número de aviones, podemos coger todos esos datos y agruparlos de algún modo. Aunque los datos por si solos ya describen a cada avión, ahora necesitamos sacar otro tipo de información. Partiendo de la base teórica en la que se fundamenta los descriptores HoG, podemos asegurar que imágenes muy parecidas visualmente, deberían tener una cierta similitud en cuanto a sus descriptores. Por lo tanto, si tenemos aviones muy parecidos, es decir, el mismo modelo de avión, estos deberían aportar descriptores parecidos. Entonces, una buena idea es agruparlos por tipos. En este caso, podemos utilizar una técnica común de clusterización.

Un algoritmo de agrupamiento (*clustering* en inglés) es un procedimiento de agrupación de una serie de vectores de acuerdo con un criterio de cercanía. Esta cercanía se define en términos de una determinada función de distancia, como por ejemplo la euclídea, aunque existen otras más robustas o que permiten extenderla a variables discretas.

En nuestro caso, un simple ejercicio de clustering será suficiente para agrupar los descriptores según su parecido. En la figura 14 podemos ver un ejemplo de agrupamiento realizado según una técnica de agrupamiento, como por ejemplo “k-means”. Los diferentes colores indican a que clúster o centro pertenece cada valor.

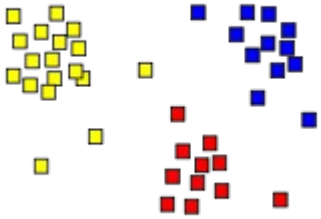

Figura 14: clustering con 3 centros

Conociendo los descriptores de cada imagen, seremos capaces de extraer información referente a qué cantidad de centros diferentes acumula cada avión. Teóricamente los aviones parecidos según su geometría deberían acumular cantidades de centros semejantes. Esta información sobre las cantidades de centros la podemos procesar en forma de histogramas. Crearemos tablas asociadas a cada fotografía en la que se incluirá la información del número de descriptores asociados a un determinado centro que aparece para un cierto modelo de avión. En la figura 15 se muestra un ejemplo de histograma que contiene la frecuencia de aparición de una serie de valores.

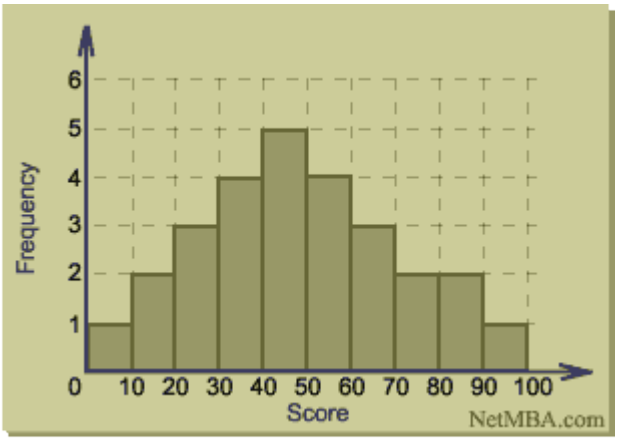

Figura 15: ejemplo de histograma

Retomando en proceso de clasificación, volvemos ahora sobre las imágenes que aun no han sido clasificadas. Las nuevas imágenes de entrada, en las que, en primera instancia, valoraremos que sean positivas. Para cada imagen positiva que deseemos clasificar, aplicaremos un procedimiento parecido al anterior.

Primeramente detectaremos el avión y reconoceremos el área en el que se encuentra. Teniendo en cuenta las dimensiones de esa región, efectuaremos un proceso de reescalado a fin de poder trabajar en una ventana del mismo tamaño de la que hemos empleado en el proceso de aprendizaje.

Seguidamente extraeremos los descriptores HoG para la región en la que determinemos que se encuentra el avión. Estos descriptores los guardaremos para posteriormente ser comparados.

A continuación, compararemos los descriptores obtenidos con los diferentes centros que previamente hemos calculado. Entonces será posible determinar a que centro se acerca más cada uno de esos descriptores. De este modo, crearemos también un histograma asociado a la imagen que estamos tratando.

Para finalizar, compararemos el histograma de la imagen a validar con los histogramas de las imágenes de entrenamiento. Para compararlos, utilizaremos métodos matemáticos, tales como la distancia euclídea. Su explicación podrá encontrarse en detalle en el apartado de diseño.

Si el proceso de clustering ha sido exitoso, la distancia entre dos histogramas debería reflejar el grado de parecido de sus respectivas imágenes a las que están asociados. Por lo tanto, concluiremos nuestra clasificación de aviones en función de los resultados obtenidos, que determinaran teóricamente que modelo concreto de avión aparece en las imágenes de test. Los resultados obtenidos los podremos comprobar en el capítulo dedicado a resultados.

3. - Diseño

3.1 - Introducción

En este capítulo centraremos nuestra trabajo en analizar los aspectos más técnicos de los principales módulos que ya hemos explicado anteriormente. Estudiaremos como han sido diseñados y como trabajan comunicándose entre ellos. También comentaremos la implementación de algunas funciones de cálculo, para lograr explicar con más facilidad su utilidad. A lo largo del capítulo mostraremos esquejas y diagramas de todo lo que se irá explicando, y que servirán para sintetizar la información de manera visual.

La documentación referente a como abordar los diferentes problemas que han ido surgiendo la encontraremos en el capítulo anterior de "Análisis". En este capítulo encontraremos información ligada a dicho capítulo, pero sin extendernos en explicar el por que se ha elegido un método en concreto para resolver cierto problema; la información que encontraremos estará orientada a explicar como se ha construido un método en particular.

También abordaremos los aspectos del diseño general del sistema. Veremos como el usuario puede actuar con el sistema, y que libertades y limitaciones tiene. Además plantearemos diferentes opciones de uso, y por qué nos centramos en algunas más que en otras.

3.2 - Arquitectura

Este apartado está destinado a explicar cómo está construido físicamente el entorno de trabajo de nuestro sistema. Veremos las posibles relaciones entre el usuario y la máquina. Cabe decir que existen múltiples combinaciones y maneras de implementar nuestro sistema para que el funcionamiento final sea el mismo, así que nosotros únicamente explicaremos los más relevantes.

En nuestro proyecto hemos trabajado montando un único sistema que solo necesitaba la interacción con una persona física. Se ha construido de esta manera ya que se trata de un trabajo de campo destinado a obtener unos resultados y medir su rendimiento, y no es necesario llevar a los extremos que requiere una implantación real. Todos los datos son introducidos por el usuario, y el ordenador sencillamente ejecuta los cálculos que se le piden. Por este motivo se simplifica mucho la arquitectura del sistema. La figura 16 muestra como se representa esta arquitectura.

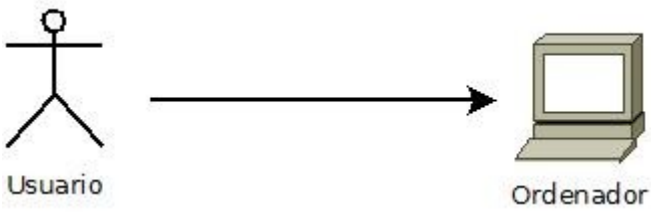

Figura 16: arquitectura simple

Sin embargo, también debemos considerar el supuesto caso para un uso final en un aeropuerto o otro tipo de empresa. En esta ocasión, podemos contar con más de un usuario y con más de un terminal en el que trabajar. Por ejemplo, partimos de la hipotética situación de que cada empleado utilice un terminal, ya sea suyo o uno dispuesto por la empresa. Estos terminales se conectarían a un servidor central encargado de distribuir el trabajo a los diferentes miembros del equipo evitando repetir el trabajo sobre un mismo caso. La gestión de la información de nuevos datos de entrada la efectuaría en ordenador principal, automatizando el proceso de captación de imágenes procedentes de la pista de aterrizaje. El servidor central tendría conexión a una base de datos, en la que se almacenaría toda la información que los empleados crean relevante, así como todo el conjunto de datos asociado a cada imagen susceptible de ser procesada. La base de datos también puede albergar fotografías de aviones, tanto nuevos como antiguos, para en un futuro actualizar los clasificadores y mejorar su funcionamiento. La figura 17 muestra como quedaría implementada esta nueva distribución.

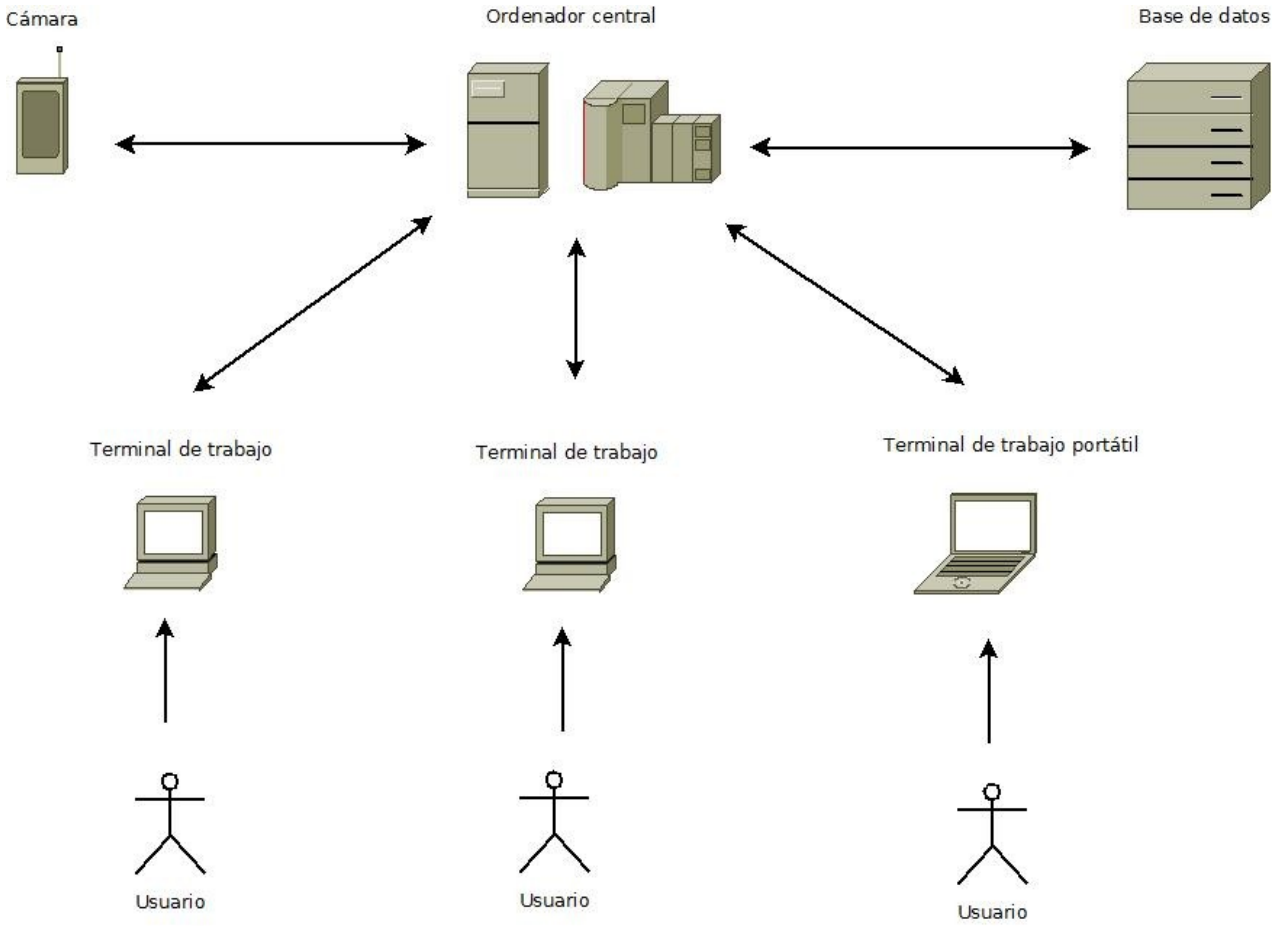

Figura 17: arquitectura compleja

La diferencia principal entre esta arquitectura y la anterior, además de poder trabajar con varias fotografías al mismo tiempo en diferentes terminales, reside en la ubicación de las funciones de cálculo. Esto significa dónde se encuentra el programa principal. En el caso de una arquitectura simple, es obvio que el programa reside en el único ordenador que existe, y que es éste el que lleva a cabo todas las tareas del programa. Sin embargo, en el caso de una arquitectura compleja, tenemos dos opciones bien distintas. Por un lado, podemos proceder a instalar el programa en cada terminal, y que sean estos los que ejecuten el programa de detección y clasificación. De este modo el ordenador central únicamente serviría de enlace entre las máquinas de los trabajadores y la base de datos, desde donde gestionaría las imágenes para que los empleados las analicen. Esta configuración tiene la ventaja de disponer de tantos equipos de cálculo como sean necesarios. En contra tenemos que cada equipo está limitado por la potencia de que dispone originalmente, y no todos tendrán una misma configuración, sobretodo si se trata de los equipos particulares de cada empleado. Esto podría influir en el rendimiento del trabajador, así como en la eficacia del programa.

Otra configuración posible seria una arquitectura compleja, en la que el ordenador central se encargue de efectuar los cálculos solicitados por los usuarios. Este ordenador solicitaría datos al servidor y los enviaría a los usuarios, que a su vez, solicitarían efectuar cálculos concretos sobre esos datos. Finalmente los resultados extraídos los podríamos guardar en la base de datos. A favor de este método se puede decir que conseguimos un reparto equitativo del trabajo, ya que todos los empleados deberán trabajar con las características propias del ordenador central. Además, para llevar a cabo unos cálculos sobre un gran volumen de datos necesitaremos un equipo con una gran potencia. Será mucho más fácil reunir ciertas características en un único ordenador, que en todos los pequeños equipos de los empleados. La gran desventaja viene asociada a dicha característica, y es que como en toda unidad de trabajo centralizada, siempre dependeremos del ordenador central para trabajar. La capacidad de trabajo viene limitada por la potencia del ordenador central, y un exceso de peticiones podría llegar a saturar el sistema. Igualmente, un posible error en éste ordenador repercutiría en el trabajo del resto de terminales. Para solucionarlo se propone trabajar concurrentemente con dos o varios ordenadores centrales, de manera que se minimice el riesgo de fallo del sistema por avería, y al mismo tiempo incrementamos las capacidades y el rendimiento de toda la arquitectura. Este tipo de configuración será posiblemente la elegida en caso de implantación en una empresa.

3.2 - Casos de uso

Este apartado tiene como objetivo explicar los diferentes casos de uso que ofrece el programa y mostrar las opciones de que dispone el usuario cuando ejecuta la aplicación. De la misma manera que en apartado anterior, haremos una distinción entre casos de uso sencillos o complejos, y los más habituales o frecuentes además de otros más específicos. Comentaremos que parámetros son los más adecuados para cada caso de uso y por que se ofrecen diferentes posibilidades y opciones.

Evidentemente cada usuario presenta unas características distintas, y sus peticiones pueden variar dependiendo de cual sea su labor. Entonces es necesario crear un software lo suficientemente flexible como para poder ofrecer múltiples posibilidades dentro de un rango lógico que previamente ha sido pactado con el cliente. Otro objetivo es diseñar y construir un código que sea apto para ser modificado, mejorado y ampliado en un futuro. Es así como surgen casos de uso distintos entre ellos, y que dejan abiertas las puertas a nuevas opciones que pueden nacer a partir de las ya implementadas.

3.2.1 - Caso de uso simple

El primer caso de uso que podemos encontrar es el relacionado con las utilidades destinadas a un único usuario que desee probar el software en su casa. Quizás el propósito del usuario sea simplemente aprender como funcionan ciertos algoritmos e iniciarse en el mundo de la visión por computador. En este supuesto, la necesidad de una interfaz clara y funcional primará por encima de las múltiples opciones. La finalidad didáctica configurará un entorno simple para facilitar al usuario la comprensión de todos los procesos que se llevan a cabo. La figura 18 muestra las opciones de que dispone el usuario.

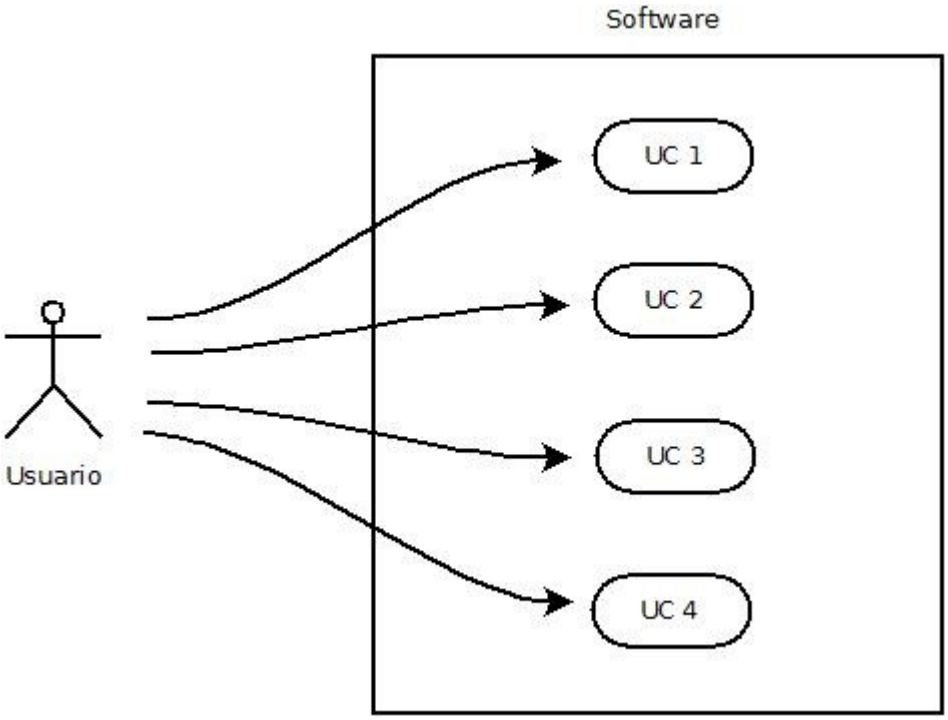

Figura 18: uso simple

Entre las opciones podemos distinguir las siguientes:

- UC1. Elegir imagen: el usuario escoge la imagen que desea procesar, y sobre la cual se efectuarán los cálculos de detección y clasificación si son necesarios.

- UC2. Visualizar localización objeto: en caso de tratarse de una imagen positiva, existe la opción de visualizar donde queda localizado el objeto reconocido dentro de la fotografía.

- UC3. Visualizar imagen redimensionada: como ya hemos comentado en el capítulo anterior, las imágenes son redimensionadas para ser tratadas. El usuario dispone de la opción de ver la imagen en su nuevo tamaño para así hacerse una idea de cuanto ha cambiado el formato sobre el que trabajará.

- UC4. Mostrar tipo: se trata de la opción final, en la que el software ha finalizado los cálculos y ha clasificado nuestra imagen de entrada. Nos indicará en que categoría de las aprendidas se encuentra nuestro objeto. En el caso de nuestro programa, nos indica a que modelo de avión pertenece el objeto que aparece en la fotografía que le pasamos como entrada.

Todas estas opciones que hemos descrito, se activarán a medida que se ejecuta el código. Aparecerá una pregunta en la consola en la que el usuario deberá elegir entre las opciones “si” y “no”. La ruta de la imagen a tratar deberá pasarse adecuadamente por linea de comandos cuando se le requiera al iniciar el programa. En caso de introducir alguna opción incorrecta, surgirá un mensaje de error avisando de la situación.

3.2.2 - Caso de uso complejo

Este tipo de caso de uso esta destinado a una labor más compleja, la cual requiera de múltiples pruebas, y donde sea oportuno efectuar pequeños ajustes particulares para cada imagen. Puede ser tanto un uso de laboratorio y con fines de investigación, como para intentar mejorar los algoritmos descritos, o simplemente para probar en mayor cantidad la eficiencia del código implementado. Dado estas situaciones, se han ampliado las funcionalidades respecto al caso de uso simple. En la figura 19 se muestran los nuevos casos de uso que comentaremos a continuación.

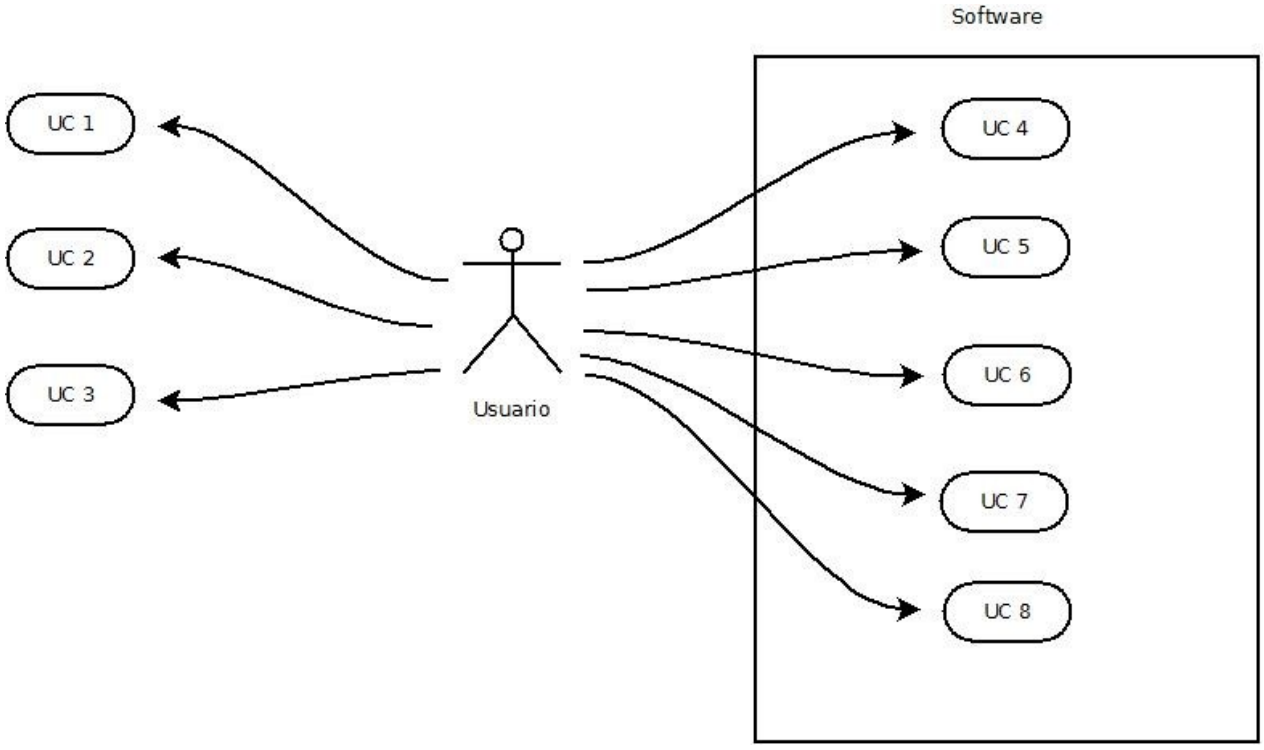

Figura 19: uso complejo

Esta vez vemos que existen casos de uso mucho más complejos. Para empezar, las opciones no se limitan al apartado de cálculos referente a detección y clasificación. Por ejemplo el usuario puede decidir construir un nuevo clasificador y después, elegir que clasificador utiliza en el momento de ejecutar el código. Detallamos las opciones a continuación:

- UC1. Crear nueva colección de imágenes positivas: si el usuario tiene acceso a una variada colección de imágenes positivas y cree que serán mejor para entrenar un futuro clasificador que las que se han proporcionado, dispone de la libertad de crear un fichero apropiado para este cometido siguiendo los pasos proporcionados en capítulo 2.

- UC2. Crear nueva colección de imágenes negativas: de la misma manera que con las imágenes positivas, el usuario puede crear un nuevo fichero conteniendo las imágenes negativas para posteriormente utilizarlo en el entreno de un nuevo clasificador.

- UC3. Crear clasificador: en esta ocasión el usuario puede decidir si utilizar el clasificador que se le proporciona por defecto o crear uno nuevo. Para construirlo solo necesita seguir los pasos que se describen en el capítulo 2.

- UC4. Seleccionar clasificador: es una de las consecuencias lógicas de la utilidad anterior. El usuario solo necesita especificar correctamente la ruta en la que se encuentra el clasificador deseado.

- UC5. Aprender aviones: esta vez es posible facilitar una colección de imágenes que contengan aviones para que el programa los utilice posteriormente para identificar los diferentes tipos de avión que existen. Deberemos indicar en cada secuencia de imágenes de que modelo se trata.

- UC6. Describir aviones: utilizaremos esta funcionalidad para describir matemáticamente los aviones reconocidos en las imágenes y generar un histograma asociado a cada imagen.

- UC7. Visualizar localización objeto: si activamos esta opción se desplegará una imagen en la que aparezca el avión localizado en cada una de las fotografías que le pasemos.

- UC8. Reconocer aviones: con esta opción es posible pasar imágenes para que el software ejecute sus cálculos y decida de que tipo de avión se trata. Siempre trabajaremos en baso a los modelos aprendidos por defecto a menos que indiquemos lo contrario en el caso de uso correspondiente, donde podremos aprender una nueva lista de modelos de aviones.

Tal como se observa, se han añadido algunas nuevas utilidades y se han suprimido otras. Por ejemplo, la opción de visualizar las imágenes redimensionadas para que adapten al tamaño apropiado de cálculo se ha eliminado principalmente por que puede resultar molesto y engorroso si se dispone de una colección muy grande de fotografías. Además, la opción se había implementado con fines puramente didácticos y para facilitar un seguimiento de todo el proceso al usuario. Consideramos que las opciones simples y de estas características son redundantes en un ámbito más técnico, así que hemos decidido suprimirlas. No obstante, siguen implementadas en el código fuente para que sea posible recuperarlas.

3.2.3 - Caso de uso final

En los dos subapartados anteriores hemos explicado dos tipos de casos de uso, con diferentes características según a quién estén orientados. El caso de uso final que se ha implementado en este trabajo reúne todas las opciones antes explicadas. La diferencia consiste en que hemos automatizado muchos procesos, para que la versión de prueba se ejecute de una manera sencilla. La figura 20 muestra como se distribuyen estas opciones.

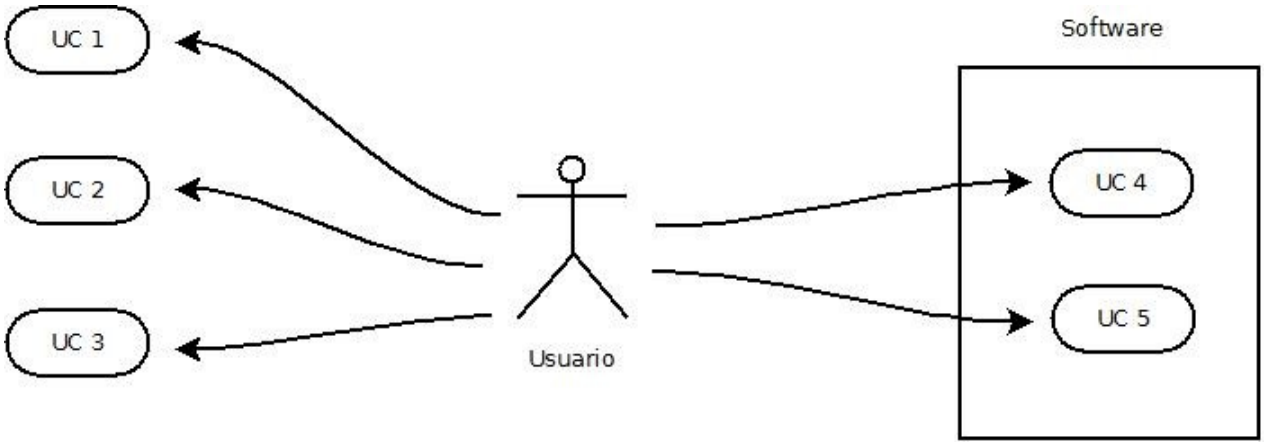

Figura 20: uso final

A continuación enumeramos los casos de uso ofrecidos:

- UC1. Crear nueva colección de imágenes positivas: si el usuario tiene acceso a una variada colección de imágenes positivas y cree que serán mejor para entrenar un futuro clasificador que las que se han proporcionado, dispone de la libertad de crear un fichero apropiado para este cometido siguiendo los pasos proporcionados en capítulo 2.

- UC2. Crear nueva colección de imágenes negativas: de la misma manera que con las imágenes positivas, el usuario puede crear un nuevo fichero conteniendo las imágenes negativas para posteriormente utilizarlo en el entreno de un nuevo clasificador.

- UC3. Crear clasificador: el usuario puede decidir si utilizar el clasificador que se le proporciona por defecto o crear uno nuevo. Si decide crearse su propio clasificador solo necesita seguir los pasos que se describen en el capítulo 2 en el apartado correspondiente.

- UC4. Seleccionar clasificador: es posible especificar una ruta que contenga un clasificador diferente al proporcionado por defecto.

- UC5. Ejecutar programa: en este caso, ejecutaremos todo el programa con los datos definidos por defecto, excepto el clasificador que se utilizará en caso de haberse especificado uno de nuevo.

De este modo, solo necesitaremos seleccionar que clasificador utilizamos. Si no queremos crear un nuevo clasificador, podemos utilizar el que proporcionamos por defecto. Para evitar complicaciones en el momento de introducir la ruta del clasificador, disponemos de un archivo situado junto a las carpetas de código proporcionadas, la ruta del cual ya se encuentra especificada en el inicio del código. En caso de querer cambiarlo, solo seria necesario pegar nuestro nuevo fichero en la misma dirección y editar la dirección por defecto, cambiando únicamente el nombre del fichero. También es posible pasarle la ruta por linea de comandos, y en caso de no hacerlo se utilizará el clasificador por defecto. El resto de opciones se mantienen, pero se ejecutarán de forma ordenada, indicando en cada momento que proceso se está llevando a cabo.

Tanto los modelos de aviones para aprender como los destinados a comprobar el funcionamiento se encuentran incrustados en el código de una manera parecida a la descrita para el clasificador. De esta manera, es posible y sencillo cambiar algún parámetro de un avión en particular, o modificar todos los aviones que sea necesario. De cualquier modo, el software está automatizado para actuar de manera inteligente y mostrar los resultados de una forma clara y concisa al usuario final. Tal y como ya hemos comentado, el código fuente mantiene toda la potencia de los otros casos de uso, pero adaptado a un entorno de trabajo más sencillo.

3.3 - Implementación de módulos

Una vez en este punto del trabajo, podemos empezar a profundizar en la implementación de los diferentes módulos que componen el trabajo. Nos centraremos en los algoritmos que utilizan las funciones más relevantes del código, y comentaremos aquellos apartados que contengan información interesante para el lector. Dividiremos este apartado en diferentes subapartados para facilitar la localización de las funciones comentadas. En algunos casos acompañaremos las explicaciones con pseudocódigo o incluso código puro si es necesario para comprender su funcionamiento.

3.3.1 - Módulo aprendizaje y entrenamiento

Este módulo, tal y como ya hemos comentado en el anterior capítulo, está ligado al uso de la aplicación “HaarTraining” y los algoritmos que implementa para entrenar y crear clasificadores. Obviaremos la parte que introduce como crear un clasificador, dado que ya ha sido explicado anteriormente, y nos centraremos en el algoritmo que utiliza, que concretamente se trata de AdaBoost.

Este algoritmo se basa en la creación de clasificadores débiles a partir de los cuales acabará creando un clasificador fuerte final. Supongamos que empieza utilizando dos clasificadores, *h1* y *h2*. Inicialmente elige el clasificador que aprende más información correctamente. En el siguiente paso, el peso de la información se vuelve a calcular para incrementar la “importancia” de los ejemplos mal clasificados. Este proceso continua y en cada paso el peso de los clasificadores débiles respecto al resto de clasificadores se determina. Entonces, encontramos un algoritmo como el siguiente:

1. Poner el peso de todos los ejemplos igual, y encontrar *h1* que maximice $\sum_i y_i h(x_i)$.

2. Realizar el balanceamiento de pesos para incrementar el peso de los ejemplos mal clasificados.

3. Encontrar el siguiente *h* que maximice $\sum_i y_i h(x_i)$. Encontrar el peso de este clasificador, α.

4. Volver al paso 2.

5. El clasificador final será:

$$sgn(\sum_{i=1}^{T} \alpha_t h_t(X))$$

Para calcular correctamente el peso de cada nuevo clasificador débil necesitamos una función objetivo y encontrar α y minimizarlo. Existen diversas funciones posibles, y todas ellas son complejas. Dado que en nuestro caso no conocemos que función han utilizado los creadores de la utilidad "HaarTraining", no podemos desarrollarla, así que nos limitamos a comentar su existencia y su importancia en el algoritmo que hemos expuesto.

3.3.2 - Módulo detección

En este apartado centraremos nuestra atención en las funciones que tienen como objetivo dibujar la ubicación de los aviones sobre las fotografías que le pasemos. Existen diversas funciones que trabajan de manera parecida y partiendo de la misma base, aunque después devuelven resultados diferentes.

Un ejemplo es la función `step1()`. Esta función devuelve un objeto del tipo `CvRect` que contiene la localización de nuestro avión para una imagen. Básicamente acepta dos imágenes, la de tamaño estándar y la imagen previamente reescalada a proporciones cuadradas, tal como se comentaba en el capítulo 2. Como parámetros acepta el valor de la proporción de reescalado sobre la imagen cuadrada, y las correcciones manuales sobre las desviaciones que puedan haber surgido en la localización sobre dicha imagen. De su código podemos destacar como carga el clasificador indicado según estas lineas:

```
static CvHaarClassifierCascade* cascade = 0;
cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name5, 0, 0, 0
);
```

En caso de no cargar un clasificador correctamente lanza un mensaje de error. También es importante conocer como detecta los objetos a partir de una función proporcionada en las OpenCV:

```
CvSeq* faces = cvHaarDetectObjects( img, cascade, storage, 1.1, 2,
CV_HAAR_FIND_BIGGEST_OBJECT, cvSize(40, 40) );
```

Notemos que carga los objetos detectados a partir del clasificador proporcionado en una secuencia. Trabajaremos sobre la imagen pequeña *img*. Si nos fijamos bien, hemos utilizado la opción *Cv_Haar_Find_Biggest_Object*. Esto significa que simplemente localizará el más grande que aparezca, y de esta manera eliminamos la posibilidad de interferencias con falsos objetos pequeños detectados. Si existiesen varios objetos de interés en la misma imagen podríamos variar este parámetro según nuestra intención. Para ver las otras opciones disponibles podemos buscar en el manual de OpenCV que proporcionan dichas librerías. El resto de cálculos simplemente cumplen con el objetivo de la función.

De la parte de detección esta es la función más relevante. Tal como ya hemos dicho, existen otras funciones que trabajan con una base similar, por ejemplo para mostrar la imagen reescalada, o el objeto enmarcado dentro de su ubicación, pero no contienen algoritmos de ejecución importantes. Además, esta función reúne la detección mediante el clasificador previamente creado y la localización gracias a la imagen adaptada correctamente para este fin.

3.3.3 - Módulo descripción

Esta sección contiene las funciones destinadas a describir matemáticamente los objetos identificados correctamente en las fotografías. En nuestro caso particular describiremos aviones y crearemos datos asociados a cada avión para posteriormente facilitar su reconocimiento. Listaremos las funciones más importantes y explicaremos su manera de trabajar internamente.

Comenzaremos repasando el proceso de descripción que hemos seguido y que se ha comentado en su correspondiente apartado del capítulo 2. Una vez hemos localizado el avión dentro de la imagen y conocemos su ubicación, crearemos una nueva imagen reescalada para que el objeto se encuentre dentro de un área de trabajo siempre de la misma medida. A esta área nos referiremos como ventana de trabajo. Seguiremos el proceso de descripción aplicando ciertas funciones sobre nuestra ventana para obtener los descriptores HoG del total de la ventana. Este proceso se hará mediante iteraciones sobre ventanas de trabajo más pequeñas, que se irán moviendo a lo largo y ancho de nuestra ventana de trabajo de tamaño estándar. Una vez finalizado el proceso obtendremos la descripción del avión mediante una serie de interpretaciones matemáticas que serán los comentados descriptores HoG. Además, podemos crear un histograma asociado a cada imagen según una distribución de centros determinada por una función de clustering.

La función `calculateIntegralHOG_v2()` es la primera que encontramos que interviene en el proceso que hemos comentado. Su cometido es obtener un histograma integral o de integrales para una imagen. El histograma obtenido se puede usar después para un cálculo mucho más rápido de los “HoG features” sobre una región arbitraria de dicha imagen. La idea de un histograma integral es análoga a la idea de una imagen integral, usada por Viola y Jones [6] para el cálculo de las “haar features” para reconocimiento facial. Matemáticamente obtenemos que:

$$ih(x, y, b) = \sum_{x' \leqslant x} \sum_{y' \leqslant y} h(x', y', b)$$

donde b representa el número de particiones del histograma. De esta manera el cálculo de las particiones sobre un rectángulo arbitrario de la imagen requiere sólo 4 por número de particiones de referencias a la matriz. Para más detalles podemos referirnos a [7].

Así pues, la función calcula el histograma integral de una imagen que recibe por parámetro. La función crea en nuestro caso un array del tamaño de 9 imágenes por que tenemos particiones de 20 grados y sin signo, por lo tanto 180/20 = 9. A partir de estas imágenes se calcularán las imágenes integrales, que juntas constituirán el histograma integral. De este modo, la función también recibe por parámetro un doble puntero a un objeto del tipo `IplImage` que tiene el tamaño concreto para almacenar la información necesaria de estas 9 imágenes. En la función se calculan las magnitudes y la orientación de los gradientes para cada píxel según la relación matemática siguiente:

```
{Magnitude = sqrt(sq(xsobel) + sq(ysobel) ), gradient = itan
(ysobel/xsobel) }
```

La función itera tantas veces como sea necesario hasta rellenar el array de 9 imágenes que constituyen el histograma integral.

La siguiente función importante que aparece es `calculateHOG_TOTAL_v2()`, que como se puede adivinar por su nombre, está destinada a calcular los descriptores HoG para una imagen. Los parámetros que admite son el histograma integral de la imagen, el tamaño de la ventana de trabajo, una matriz donde almacenar los valores de los descriptores, y el tipo de normalización que deseamos hacer.

Esta función se ayuda de otras funciones pequeñas a las que llama y que comentaremos a continuación por que están muy relacionadas. Lo que la función hace es llamar `calculateHOG_area_param()`, indicándole una pequeña ventana sobre la que calcular los descriptores HoG. Cuando los ha obtenido, la ventana se desplaza ligeramente, y se vuelve a llamar a la función, y se repite el proceso hasta recorrer toda nuestra región de interés (ROI). Al acabar habrá almacenado los descriptores HoG para toda la ventana de trabajo que le hemos indicado. La idea es repetir este proceso con todas las imágenes para después aplicar una función de clustering al conjunto de descriptores de las diferentes imágenes. La figura 21 ilustra el proceso. Podemos ver lo que seria la pequeña ventana de trabajo llamada *Patch* sobre la que deberá trabajar `calculateHOG_area_param()`, y cuyo centro se irá desplazando muy lentamente, obteniendo incluso solapamiento de regiones ya calculadas.

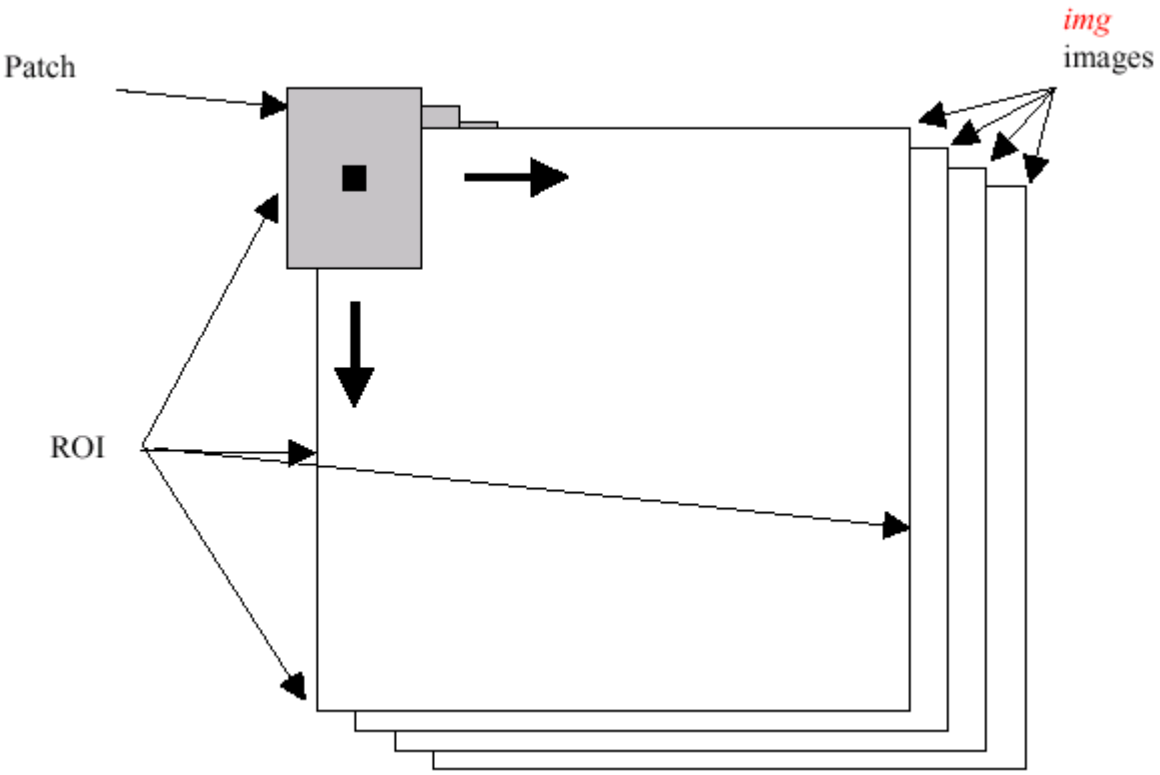

Figura 21: esquema de trabajo de la función calculateHOG_TOTAL_v2()

Ahora que ya conocemos como trabaja la función más externa, podemos explicar la similitud de la función `calculateHOG_area_param()`, que calculara los descriptores HoG para una región llamada "bloque", que a su vez dividirá en "celdas". Los parámetros que recibe son la matriz para almacenar el valor de los descriptores HoG del bloque, el tamaño del bloque, el tamaño de la celda, el histograma integral, y la normalización a efectuar. La función itera sobre la nueva región de interés llamando a `calculateHOG_rect()`. Esta función recibe como parámetros la celda para la cual se ha de los descriptores HoG, la matriz donde almacenarlos, el histograma integral, y la normalización a efectuar. Esta vez si que ya calcula definitivamente los valores para cada una de las particiones de cada imagen del histograma uno por uno.

Llegado a este punto, podemos resumir que `calculateHOG_TOTAL_v2()` trabaja dentro de una ventana o ROI que le indicamos subdividiéndolo en bloques; `calculateHOG_area_param()` trabaja dentro de una región llamada bloque subdividiéndola en celdas, y `calculateHOG_rect()` trabaja en una región pequeña llamada celda para la cual obtiene los descriptores HoG. Son funciones que se llaman recursivamente unas a otras en una serie de iteraciones para mejorar el rendimiento y los valores calculados.

Una vez finalizada la ejecución de estas funciones ya disponemos de los descriptores HoG para una imagen, y si ejecutamos en proceso de manera iterativa como en el caso de nuestro ejemplo, para todo el conjunto de imágenes de aviones que le pasamos como entrada. Ahora ya hemos descrito cada imagen y cada avión según un modelo matemático.

3.3.4 - Módulo clasificación

El último de los módulos en los que se ha dividido el trabajo consiste en clasificar los aviones según los diferentes modelos aprendidos que existen. Después de las opciones planteadas en el apartado correspondiente del capítulo 2, hemos decidido clasificarlos mediante la creación de histogramas asociados a la cantidad de tipo de descriptores HoG que posee cada avión. A continuación explicaremos las funciones más importantes para esta tarea.

En primer lugar hemos de realizar una clasificación del tipo de descriptores que existen. Para ello hemos usado la función de clustering `cvKMeans2()` que implementa un algoritmo "k-means" que encuentra el número centros de clústers que le indiquemos agrupando la colección de muestras de entrada alrededor de estos centros. En nuestro caso ha quedado definido así:

```
cvKMeans2(mat_total,40,jarl,cvTermCriteria(  CV_TERMCRIT_EPS+CV_TER
MCRIT_ITER, 10, 1.0 ),1,0,0, salida,0);
```

donde `mat_total` es la matriz con los datos de entrada que queremos agrupar en clústers, `40` son el número de clústers deseados, `jarl` es el vector de salida que contiene el número de clúster asociado a cada dato de entrada y `salida` contiene el valor de los centros encontrados. El resto de opciones responden modos de calcular los centros.

El algoritmo que más comúnmente implemente un proceso de clustering del tipo k-means usa una técnica de refinamiento iterativa. Dado un conjunto de k interpretaciones $m_1^{(1)}, \ldots, m_k^{(1)}$, el algoritmo procede alternando entre estos dos pasos:

- Etapa de asignación: asigna cada observación al clúster con la interpretación más cercana.

$$S_i^{(t)} = \left\{ \mathbf{x}_j : \left\| \mathbf{x}_j - \mathbf{m}_i^{(t)} \right\| \leq \left\| \mathbf{x}_j - \mathbf{m}_{i^*}^{(t)} \right\| \text{ for all } i^* = 1, \ldots, k \right\}$$

- Etapa de actualización: calcula las nuevas interpretaciones para ser el centroide de las observaciones en el clúster.

$$\mathbf{m}_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{\mathbf{x}_j \in S_i^{(t)}} \mathbf{x}_j$$

El algoritmo se inicializa escogiendo centros aleatorios y se itera hasta conseguir converger, momento en el que la asignación a los centros no varia. La figura 22 nos muestra un ejemplo de ejecución del algoritmo. En la primera imagen se plantean 3 interpretaciones o centros aleatorios. Seguidamente se crean *k* clústers asociando cada observación al centro más cercano. A continuación el centroide de cada clúster se convierte en el nuevo centro. Finalmente repetimos los pasos 2 y 3 hasta alcanzar la convergencia.

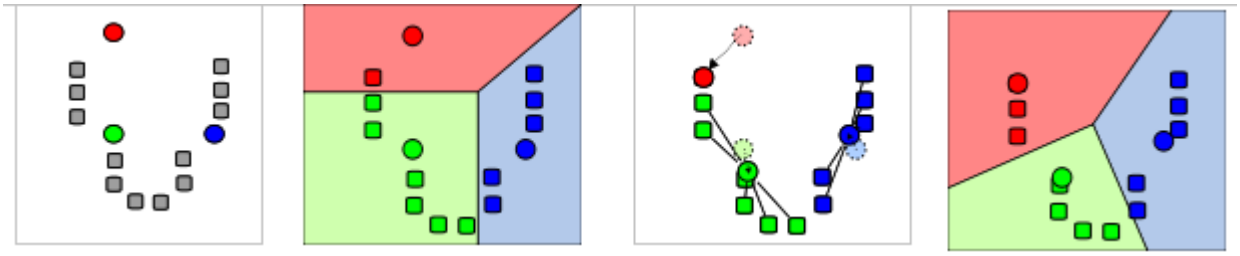

Figura 22: ejecución de k-means

Una vez hemos obtenido los centros del clustering, hemos de crear un histograma asociado a cada imagen, ya sea para las imágenes que sirven de aprendizaje como para las que sirven de prueba. Hemos implementado pequeñas funciones que ayudan. `obtener_hist_test()` calcula el histograma para una imagen que recibe por parámetro. Se utiliza para calcular los histogramas de las nuevas imágenes. Primero calcula sus descriptores HoG y después los compara con los centros obtenido anteriormente, que también le hemos de pasar por parámetro.

También tenemos otras funciones que directamente aprovechan el vector de etiquetas generado por la función de clustering para crear los histogramas asociados a las imágenes de entreno, ya que nos ayudamos de que conocemos el orden en que las hemos procesado. Un ejemplo es `rellenar_hist()`, que efectúa la tarea anterior basándose en la posición de cálculo de la imagen. Necesita como parámetros la matriz de indices, el histograma a rellenar, y las posición inicial y final de los indices de dicha imagen.

Para acabar, la función que compara los histogramas de la lista de imágenes de test con los de las aprendidas es `todos_hist_test()` que para cada imagen de test llamará a la creación de su histograma y a la comparación con los de las imágenes de entreno. Mostrará por pantalla la similitud con cada tipo de avión.

Para el cálculo de la distancia entre histogramas hemos hecho servir la distancia euclidiana. En general, la distancia entre los puntos $P=(p_1, p_2, \ldots, p_n)$ y $Q=(q_1, q_2, \ldots, q_n)$ en un espacio euclidiano n-dimensional se define como $\sqrt{(p_1-q_1)^2+(p_2-q_2)^2+\ldots+(p_n-q_n)^2}$, o en forma de sumatorio tal que:

$$\sqrt{\sum_{i=1}^{n}(p_i-q_i)^2}$$

Existen otra cantidad de funciones importantes pero que carecen de relevancia en el aspecto técnico. Son simplemente funciones de cálculo y de comparación y por ello no aparecen comentadas en este apartado.

3.4 - Otras funciones

Tal y como es de suponer existen muchas más funciones que las que hemos comentando hasta el momento. Todas ellas cumplen algún objetivo y son importantes para el funcionamiento del software. No obstante, debido a motivos como la simplicidad que tienen o la poca importancia técnica, hemos decidido no incluirlas en este capítulo de diseño. Incluir el total de funciones hubiese convertido los anteriores apartados en explicaciones demasiado densas y le restaría importancia a las funciones verdaderamente significantes. Si el lector tiene especial interés en conocer el código fuente a fondo, puede explorar los ficheros que se facilitan en este trabajo. Además cada función está debidamente comentada y contiene anotaciones e indicaciones para ayudar a entender su funcionamiento.

4. - Resultados

4.1 - Introducción

En este capítulo hablaremos sobre los resultados obtenidos. Explicaremos las funciones que actúan para realizar los cálculos y comentaremos en profundidad como afectan los cambios de algunos datos a los resultados finales. También discutiremos por qué logramos buenos resultados en muchos casos y por qué no tan buenos en otros. Documentaremos las explicaciones con imágenes y gráficos para generar una idea visual de la situación que estamos tratando. El capítulo lo estructuraremos según los resultados obtenidos en relación a diferentes procesos hasta llegar a las comparaciones y clasificaciones finales.

En cuanto al tipo de material empleado para obtener y calcular los resultados podemos encontrar de diferentes características tales como:

- **Datos**: utilizaremos imágenes positivas y negativas para crear el clasificador, de tamaño 1600 x 1200 píxeles. Las imágenes positivas destinadas a ser analizadas las redimensionaremos a 800 x 600 píxeles, y al mismo tiempo a dimensiones particulares que tengan una relación del tamaño del objeto cuadrada. Utilizaremos imágenes de 5 modelos distintos de avión.

- **Hardware**: emplearemos dos equipos distintos, cada uno con potencias diferentes.

- **Software**: implementaremos diferentes algoritmos de cálculo para detectar, identificar, describir y clasificar aviones.

- **Métricas de evaluación**: los criterios de evaluación vendrán determinados por diferentes comparaciones en los campos antes nombrados. Compararemos el rendimiento de los equipos hardware en cuanto a tiempo empleado para generar el mismo resultado. Mediremos la potencia de cálculo de los algoritmos basándonos en su eficacia. Finalmente obtendremos el porcentaje de aciertos del software sobre una cantidad limitada de los datos proporcionados.

4.2 - Creación del clasificador

Las primeras conclusiones sobre el funcionamiento del software las podemos encontrar en una de las primeras fases de ejecución. Cuando creamos el clasificador, el tiempo de ejecución viene determinado según el valor de los parámetros que le pasemos. Tal y como describíamos en el capítulo 2, existen múltiples opciones que podemos activar, todas ellas explicadas en dicho capítulo. La figura 23 muestra las opciones introducidas y como quedan el resto de opciones por defecto. Los valores que hemos usado en concreto son los siguientes:

```
$ opencv_haartraining -data haarcascade -vec positivos.vec -bg negativos.dat -nstages 20 -nsplits 2 -minhitrate 0.999 -maxfalsealarm 0.4 -npos 270 -nneg 2007 -w 40 -h 40 -sym
```

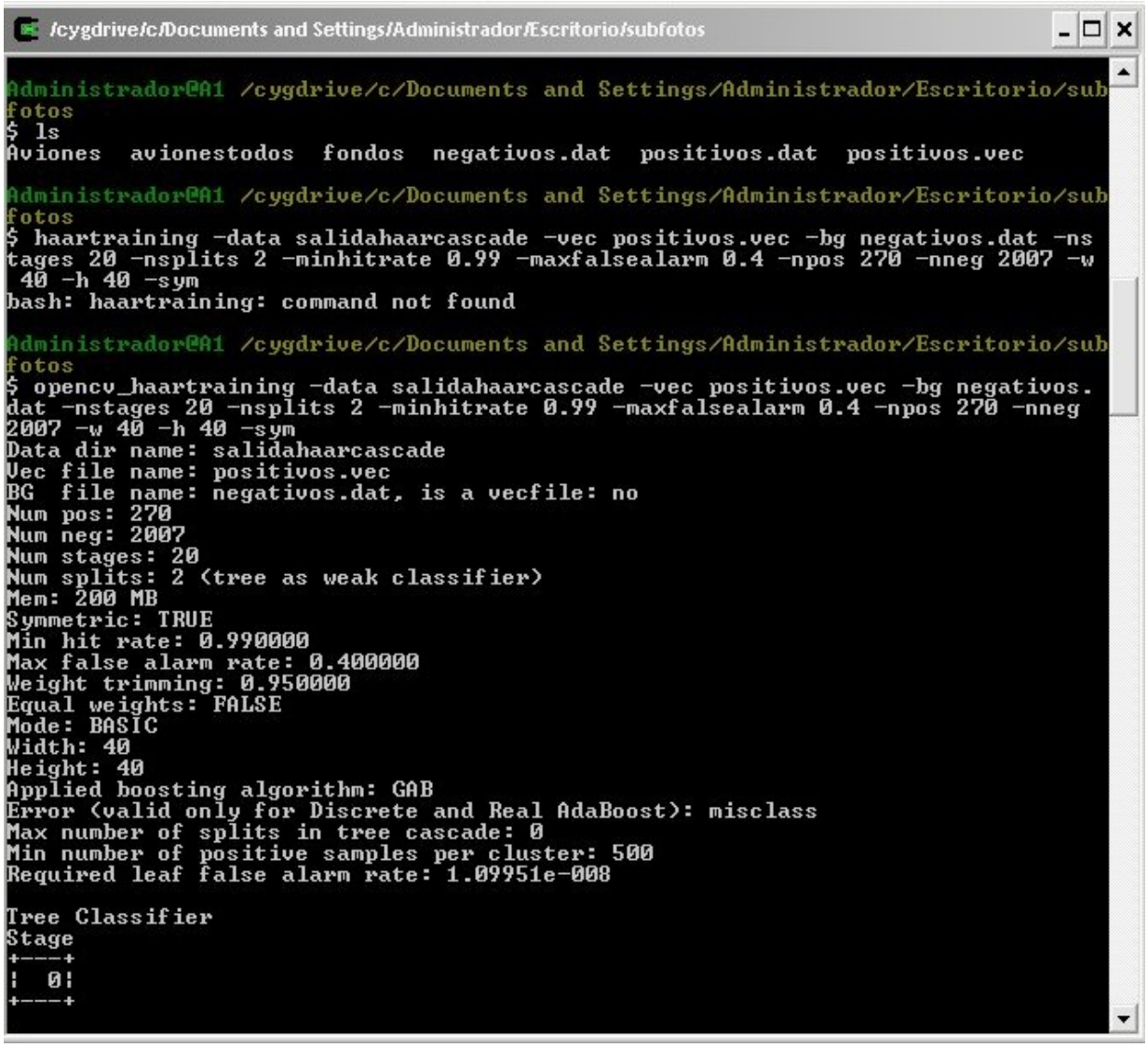

Figura 23: ejecución de haartraining

Una vez ejecutado haartraining, es interesante comparar el tiempo que consume dependiendo del procesador del equipo en el que se trabaja y de la cantidad de memoria ram que se permite usar. En nuestro caso hemos hecho algunas pruebas cambiando la máquina en la que se ejecutaba. En este caso en particular, el tiempo de ejecución superó las 4h de trabajo. En concreto, el tiempo para cada "stage" de un clasificador se muestra en segundos, y la suma total ascendió a más de 15000 segundos. En la figura 24 vemos un ejemplo de tiempo consumido en el cálculo de un stage intermedio, expresado en segundos, tal como indica *"state training time: 5651.86"*.

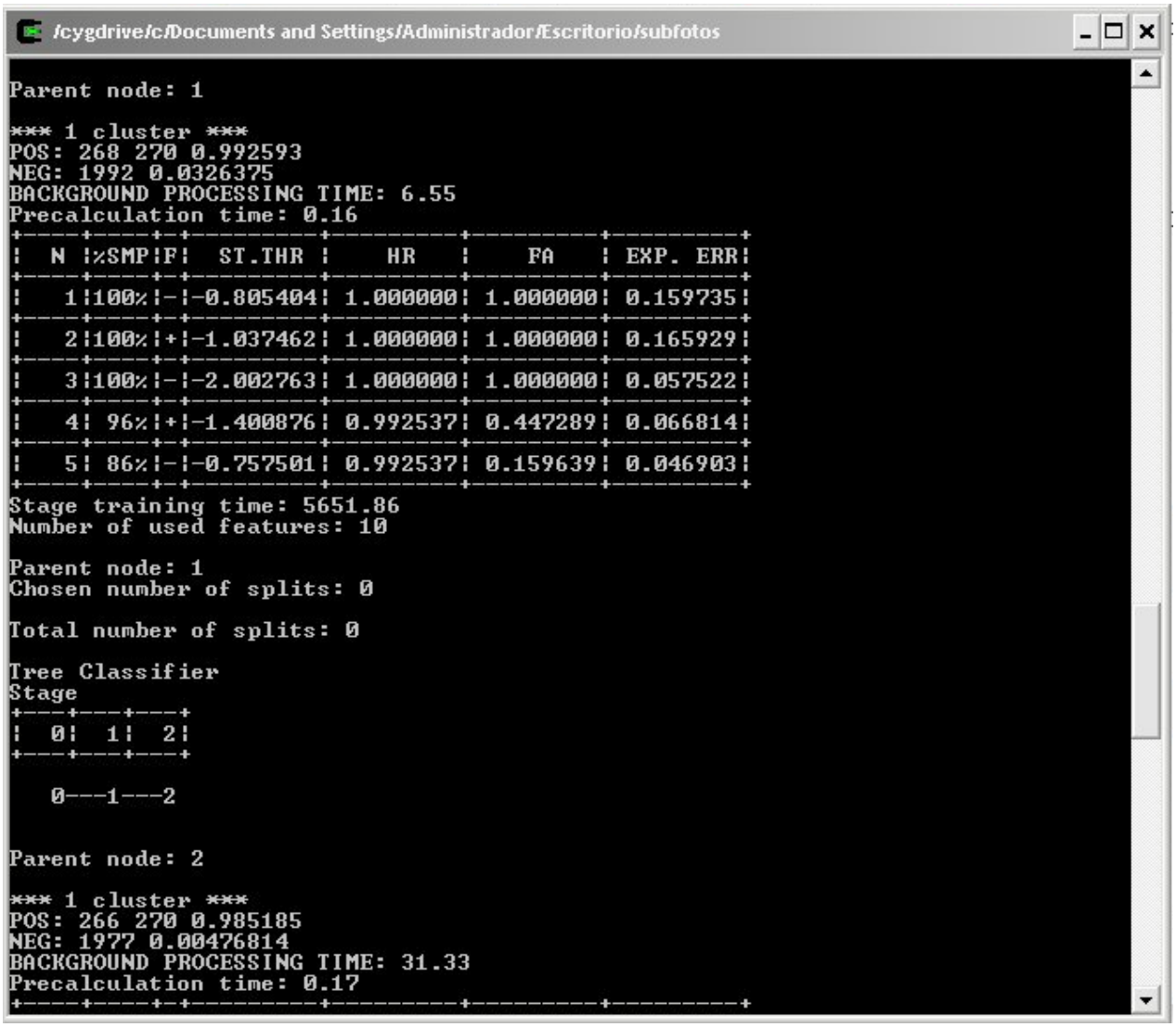

Figura 24: stage de un clasificador

Seguidamente hemos realizado los mismos cálculos en otra máquina para generar exactamente el mismo clasificador. Únicamente hemos cambiado los recursos de ram asignados, pasando de 200Mb a 1024Mb. El procesador también era diferente, siendo el primero un procesador con un núcleo a 3,0 Ghz, y en la segunda prueba un doble núcleo a 3,10 Ghz. El tiempo transcurrido disminuye notablemente, siendo ahora de poco más de 1 hora de trabajo. Por lo tanto, hay que tener en cuenta los factores de hardware. Además estos tiempos también variarán en función de la cantidad de imágenes que procesemos, y tal y como hemos comentado, nosotros estamos usando un número menor al recomendado. En un caso real de implantación en una empresa deberíamos plantearnos muy detenidamente qué máquinas tendremos a nuestra disposición para trabajar, ya que en función de sus características podremos ofrecer una mayor calidad y rendimiento en casos como el explicado.

4.3 - Detección de aviones

En este subapartado estudiaremos los resultados de la fase de detección de aviones. El objetivo es comprobar qué efectividad tiene el clasificador que hemos creado anteriormente. Como ya hemos comentado en el capítulo 3, disponemos de ciertas funciones que basan su uso en ejecutar el clasificador para identificar los aviones dentro de la imagen. En este caso tenemos una función de prueba llamada `captura()`, que se dedica a analizar la imagen que le pasemos y a mostrar si ha sido correctamente detectado el avión. La función tiene la siguiente cabecera:

```
void captura( IplImage* img, IplImage* img_grande2, float factor_x, float factor_y, int izq, int der, int arr, int aba );
```

Los parámetros que acepta son:

- img: imagen de proporciones cuadradas.
- img_grande2: imagen normal de tamaño 800x600.
- factor_x: factor de reescalado en el eje horizontal.
- factor_y: factor de reescalado en el eje vertical.
- izq: píxeles que aplicamos como corrección hacia la izquierda.
- der: píxeles que aplicamos como corrección hacia la derecha.
- arr: píxeles que aplicamos como corrección hacia arriba.
- píxeles que aplicamos como corrección hacia abajo.

Para cada imagen, si la detección que se muestra no se ajusta correctamente al avión, podemos aplicar una pequeña corrección según los parámetros citados. La corrección se expresa en píxeles, y hay que tener en cuenta que el sistema parte de una coordenadas situadas en la esquina superior izquierda. De este modo si queremos avanzar hacia la derecha o hacia abajo, indicaremos un número positivo, mientras que entraremos una cantidad negativa para mover los parámetros hacia la izquierda o hacia arriba. La figura 25 muestra un avión detectado en su tamaño de proporciones cuadradas. En la figura 26 vemos el resultado de recalcular la aplicación según los parámetros siguientes `captura(img, img_grande, 3.76, 1.0, 15, -15, 0, 0)`:

Figura 25: detección correcta

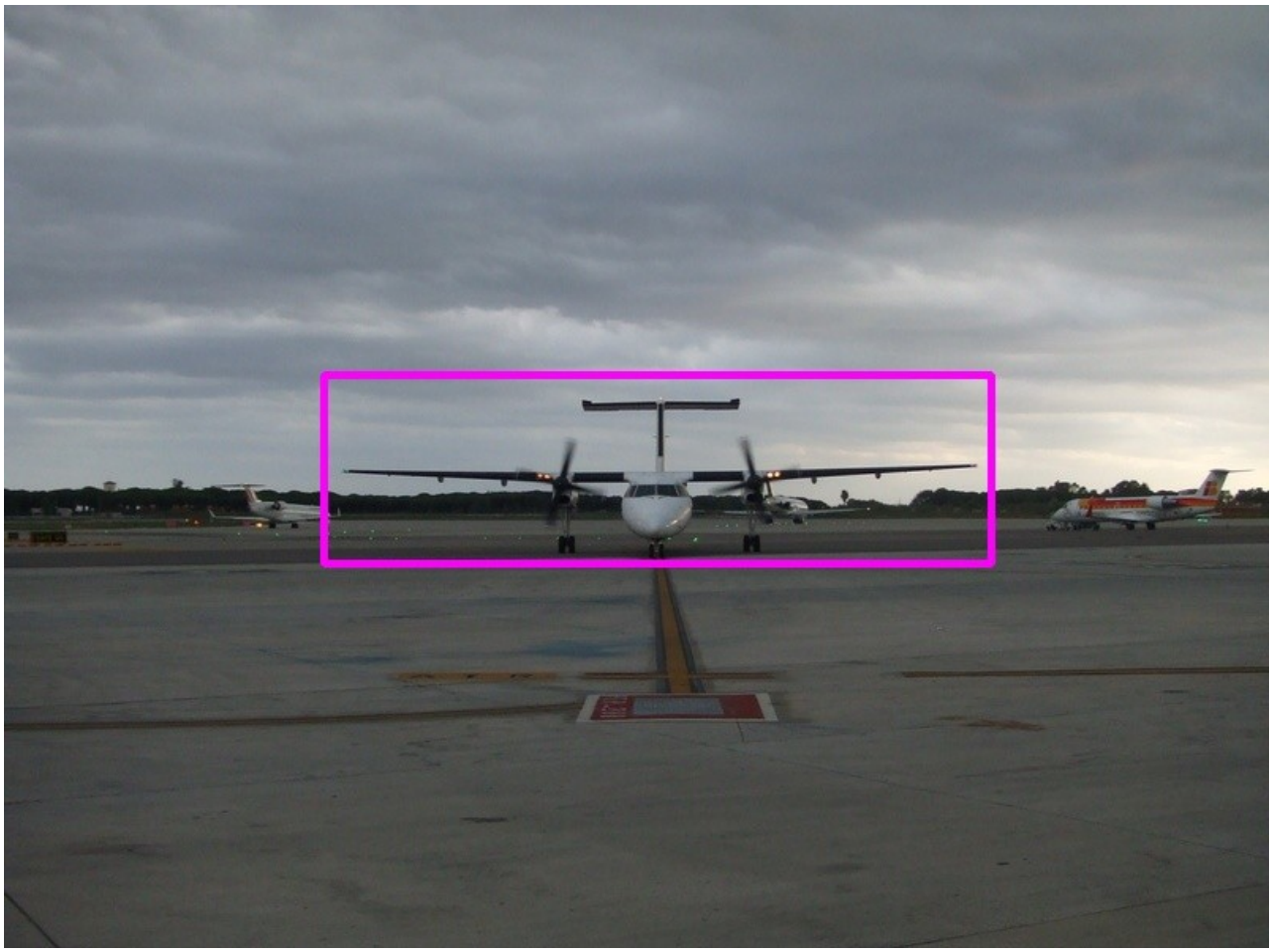

Figura 26: corrección sobre detección correcta

Tal y como se puede apreciar, las correcciones aplicadas son mínimas, y carecen de importancia en este caso en concreto. Se han aplicado para mostrar visualmente su efecto. No obstante, la función comentada sirve para ver si se ha efectuado una correcta detección sobre la imagen. En caso de necesidad disponemos de los parámetros necesarios para corregir un eventual error. Estos parámetros han de ser tratados con sumo cuidado para no desvirtuar las propiedades de la aplicación.

En el caso de nuestras imágenes hemos ejecutado la función de manera iterativa para verificar el buen funcionamiento. Como era de esperar, no todas la imágenes han sido detectadas correctamente, aunque éstas han sido una minoría. El ratio de acierto se ha situado cerca del 95% de casos correctamente detectados.

De este número de aviones correctamente localizados, hemos seleccionado un grupo para confeccionar la colección de aviones de entrenamiento y de prueba. Tal como comentaremos en los siguientes subapartados, a una pequeña cantidad de ellos se les ha aplicado alguna corrección en los parámetros de detección, aunque ha sido mínima tal y como hemos visto en el ejemplo.

En el caso de los aviones detectados incorrectamente podemos decir que han sido una pequeña cantidad. Además, el fallo en la localización suele ser grande, como muestra la figura 27. Sin embargo, entre fotografías muy similares, puede existir error en una de ellas y una correcta localización en el resto, motivo que nos lleva a pensar que es un error eventual. Probablemente mejorando la cantidad de imágenes utilizadas al crear el clasificador podríamos eliminar este tipo de errores.

Figura 27: error de detección sobre imagen reescalada

4.4 - Clasificación de aviones

Llegados a este punto sólo nos falta explicar que técnicas hemos utilizado para la clasificación de aviones y en que porcentaje hemos acertado. Primeramente es importante recordar que tipo de datos hemos hecho servir. El clasificador ha sido creado con 207 imágenes positivas y 2007 negativas. Del conjunto de fotografías positivas, hemos reescalado todas ellas al tamaño de 800 x 600 píxeles para ser tratadas. Además, de para cada una de ellas hemos creado su versión "cuadrada" en la que las proporciones del avión eran las mismas para el eje horizontal como para el eje vertical.

Entre todas las imágenes positivas existen 5 tipos de avión distintos. En un mismo tipo de avión podemos encontrar fotografías muy parecidas, y algunas que nos parecerían de otro modelo distinto. Un mismo modelo de avión puede estar pintado de maneras muy diferentes según sean las exigencias de las compañías que los compran. Esta es una dificultad añadida a la hora de diferenciar visualmente estos aparatos. A continuación, en las figuras 28, 29, 30, 31 y 32 podemos ver ejemplos de los 5 modelos de avión que hemos utilizado y su nombre de modelo.

Figura 29: avión A320

Figura 28: avión B717

Figura 30: avión B737

Figura 31: avión BA146

Figura 32: avión DH8

En nuestra fase de clasificación de aviones necesitamos dos conjuntos de imágenes. Un conjunto de aprendizaje y un conjunto de pruebas. En nuestro caso hemos creado cada conjunto con un total de 20 imágenes. Debido a los márgenes de error en algunas imágenes del modelo BA146 y a que solo disponemos de unas pocas fotografías y muy parecidas de este tipo de avión, hemos decidido apartarlo de los conjuntos que crearemos.

En cada conjunto de 20 imágenes existen 5 fotografías de cada tipo de avión, ninguna igual, representando así cuatro tipos de avión, que son el A320, el B717, el B737 y el DH8.

Para determinar que de que tipo de avión, previamente debemos efectuar los cálculos explicados en el capítulo 3 para cada una del total de las 40 imágenes de que disponemos. Una vez disponemos de los descriptores HoG de cada avión crearemos su histograma asociado. El proceso de comparación de histogramas lo efectuará la función `todos_hist_test()`, que se encargará de comparar el histograma de cada imagen de test con los histogramas de todas las imágenes del conjunto de aprendizaje, y mostrará en pantalla el tipo de avión al que más se parece. La función admite los siguientes parámetros:

```
void todos_hist_test(int filas, CvMat* indices, CvMat* centros)
```

donde `filas` es el número de filas de la matriz de descriptores HoG de una imagen cualquiera, `indices` es la matriz con etiquetas con los centros asociados a cada una de las imágenes del conjunto de aprendizaje, y `centros` es la matriz con el valor de los diferentes centros calculados en el proceso de clustering.

La función tiene implementado por defecto el conjunto de imágenes de prueba con sus parámetros sobre las que trabajar, así que si deseamos cambiar alguna imagen lo podemos hacer directamente sobre el propio código del programa. En una versión más avanzada y menos automatizada se pasarían las imágenes por parámetros, pero en nuestro caso solo ralentizaría el proceso de analizar resultados.

Otro aspecto importante a tener en cuenta, es que la función comentada no se limita decidir a que tipo de avión se parece más otro avión cualquiera, sino que nos muestra la similitud con cada uno de los aviones aprendidos. De esta manera, en los aviones que no acierta a clasificar claramente, podemos ver si ha fallado por mucho o si se ha quedado cerca de su verdadero grupo. En la figura 33 podemos ver un ejemplo de los datos comentados y la distancia a varios tipos de avión. La distancia se expresa en unidades según la distancia euclidiana entre histogramas.

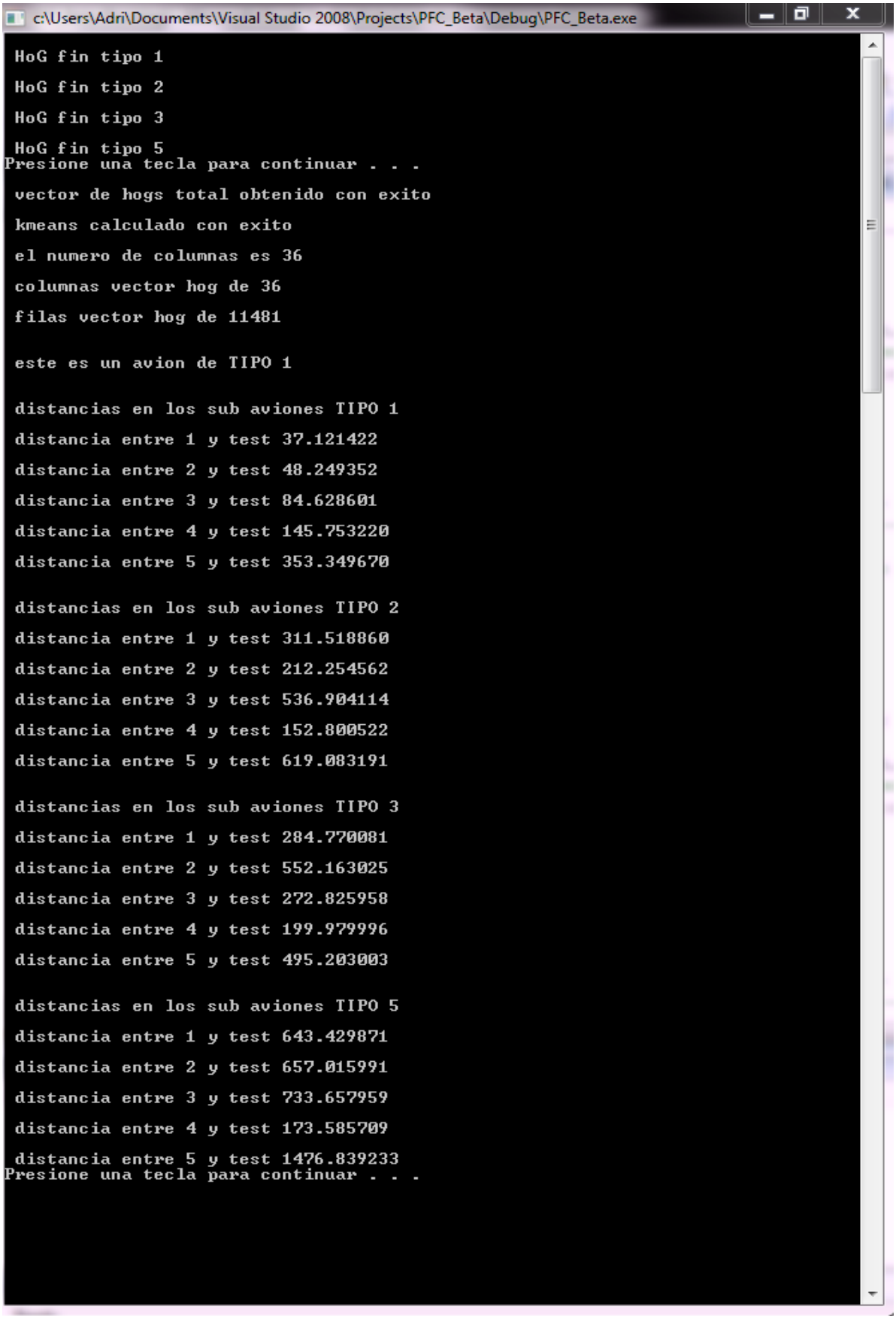

Figura 33: distancia de un avión de tipo 1 al resto de modelos

4.5 - Análisis de resultados

En este apartado analizaremos numéricamente los resultados asociados a las diferentes etapas que hemos visto hasta ahora. Las explicaciones se acompañaran de tablas que mostrarán gráficamente los porcentajes, y a partir de las cuales podremos discutir los resultados finales obtenidos.

4.5.1 - Resultados de hardware

En cuanto a los resultados probando el diferente hardware del que dispones, hemos podido comprobar como la potencia del equipo en el que trabajemos influye en gran medida sobre el tiempo de cómputo. Si tenemos en cuenta el volumen de datos tratados, podemos hacernos una idea de que forzosamente necesitamos disponer de un equipo de alto rendimiento para modelar casos más complejos. La figura 34 muestra la tabla de comparativas entre los dos equipos.

Equipo	Procesador (Ghz)	Núcleos	Memoria Ram empleada (Mb)	Tiempo empleado (segundos)
Ordenador 1	3	1	200	15000
Ordenador 2	3,1	2	1024	3800

Figura 34: comparativa de hardware creando clasificador

También podemos diferenciar entre el tiempo que han tardado en calcular los descriptores HoG para 20 imágenes cada uno de los equipos, y ver como influye el hardware empleado. La figura 35 muestra esta segunda prueba.

Equipo	Procesador (Ghz)	Núcleos	Memoria Ram empleada (Mb)	Tiempo empleado (segundos)
Ordenador 1	3	1	512	600
Ordenador 2	3,1	2	4096	90

Figura 35: comparativa de hardware calculando HoG features

4.5.2 - Resultados de software

En el caso de los algoritmos utilizados podemos decir que cumplen con los resultados esperados. El uso de histogramas integrales para calcular de manera rápida los descriptores HoG ha funcionado, dado que los resultados que se obtienen son correctos. Además hemos variado las parámetros de la función de clustering, como por ejemplo el número de centros a crear, variando con valores entre 30 y 50, y también el número de veces que colocaba aleatoriamente los primeros centros entre 1 y 4, y podemos afirmar que no ha influido en el tiempo de ejecución.

El tiempo de cómputo es válido, tardando aproximadamente 90 segundos en ejecutar el conjunto de cálculos sobre 20 imágenes y de aplicar k-means sobre estos datos. Dado que no hemos podido comparar este tiempo de ejecución con otros métodos, y que consideramos los 90 segundos como una buena marca, daremos por correctos estos resultados.

4.5.3 - Resultados de detección

Tal como ya hemos comentado en su debido apartado, hemos probado la función de detección sobre el conjunto de imágenes positivas para ver si el clasificador trabajaba correctamente y así ha sido. De 207 imágenes se ha detectado correctamente sin necesitar apenas correcciones de posición 195 imágenes, lo que significa un 94% de precisión. La figura 36 muestra el porcentaje logrado. Estos datos están en linea con los esperados, y podremos utilizarlos para trabajar adecuadamente.

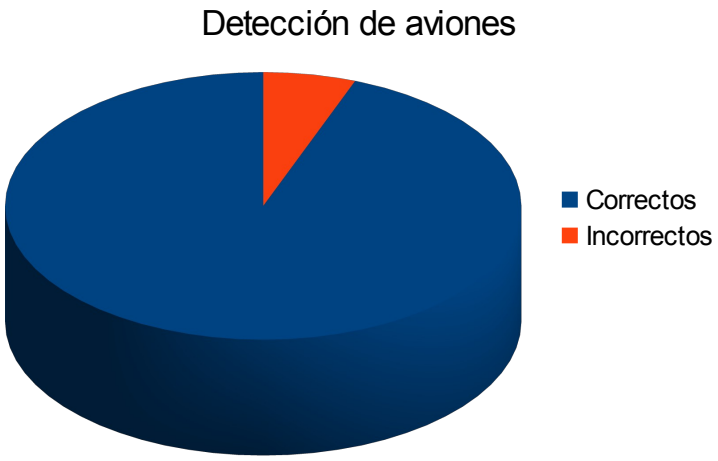

Figura 36: porcentaje de aviones detectados

4.5.4 - Resultados de clasificación

Los resultados que presentamos a continuación corresponden a la última fase del proyecto, y puede que la más importante a la hora de evaluar resultados. Se trata de evaluar el número de detecciones y clasificaciones hechas correctamente. Partiendo de la base antes expuesta, en la que se han usado dos conjuntos de 20 imágenes cada uno, con 4 tipos diferentes de aviones, hemos obtenido los resultados que se muestran en la tabla de la figura 37:

Modelo	Avión número	Clasificación correcta	Total %
A320	1	Sí	
A320	2	Sí	
A320	3	Sí	
A320	4	Sí	
A320	5	No	
			80,00%
B717	1	Sí	
B717	2	Sí	
B717	3	Sí	
B717	4	Sí	
B717	5	Sí	
			100,00%
B737	1	Sí	
B737	2	Sí	
B737	3	No	
B737	4	Sí	
B737	5	Sí	
			80,00%
DH8	1	No	
DH8	2	Sí	
DH8	3	Sí	
DH8	4	Sí	
DH8	5	Sí	
			80,00%

Hemos de puntualizar que para un correcto funcionamiento y para evitar que los resultados extraídos pudiesen ser distorsionados por factores externos, el conjunto total de las 40 imágenes ha sido seleccionado entre todas las imágenes positivas que eran detectadas correctamente. Así pues, no se ha verificado que no se incluyan imágenes incorrectamente detectadas, ya que el porcentaje de éstas es muy bajo y la inclusión en un grupo tan reducido de pruebas podría falsear el resultado final.

Los datos expresados en la tabla verifican un buen funcionamiento del software y aparecen en concordancia con los resultados esperados. La valoración global de todos los resultados hasta ahora comentados la resumiremos en el siguiente apartado.

4.6 - Valoración global

Una vez vistos y analizados los resultados obtenidos en las diferentes pruebas realizadas podemos hacer una valoración global del proyecto. Hemos visto que en cada apartado en particular los resultados obtenidos han sido buenos. Si miramos en proyecto en términos generales podemos hacer una valoración positiva también de los resultados obtenidos, ya que cumplen nuestras expectativas si enlazamos todos los módulos de trabajo. Además, los resultados finales obtienen un porcentaje de acierto muy grande, alcanzando un promedio del 85% de aciertos, y esto nos hace creer aun más en la correcta elección e implementación de los diferentes algoritmos a lo largo de todo el código.

Según los resultados obtenidos podemos afirmar que la cantidad de datos con los que hemos trabajado han sido suficientes para simular he implementar soluciones a un problema de visión por computador. Aunque el ratio de aviones mal detectados es bajo, las imágenes sobre las que se produce el error no muestran un patrón concreto. Dada la similitud entre imágenes bien detectadas y mal detectadas únicamente podemos pensar que se trata de un error ajeno a los algoritmos de cálculo y que es debido errores del clasificador, producidos probablemente por una cantidad pequeña de imágenes positivas. Además, las imágenes en las que el clasificador no acierta en la detección se suelen corresponder con aquellos aviones en los que el fondo de la fotografía contiene elementos destacados, como edificios o partes de otros aviones, y no simplemente una pista de aterrizaje limpia con el horizonte como escenario de fondo. También cabe destacar que en los casos en que no se consigue clasificar correctamente, si observamos los datos proporcionados veremos que en realidad la diferencia entre el valor correcto y el incorrecto es muy pequeña, y en la mayoría de casos la distancia al histograma equivocado es sólo un 0,05% menor que a los histogramas del modelo correcto.

En resumen y una vez vistos todos los valores obtenidos, podemos decir que el funcionamiento es el adecuado, y aun siendo exigentes, los resultados son los esperados.

5. - Conclusiones y trabajo futuro

5.1 - Conclusiones

Una vez llegado a este capítulo podemos repasar todos los puntos propuestos a lo largo del proyecto y valorar su elaboración. No obstante, en el capítulo 4 de análisis de resultados ya se han comentado la mayoría de resultados obtenidos. Entonces no queda más que puntualizar algún dato que nos faltaba por conocer. Por ejemplo, una vez que hemos acabado el trabajo somos capaces de elaborar un presupuesto para una posible implantación en una empresa real, por ejemplo en un aeropuerto. La figura 38 muestra esta tabla.

Concepto	Cantidad	Precio €	Iva €	Precio final €
Visual Studio	1	800	144	944
Base de datos	1	2000	360	2360
Equipo central	1	4000	720	4720
Equipo de trabajo	10	1000	180	11800
Horas de trabajo	700	30	3780	24780
			Total	**44.604,00 €**

Después de realizar el trabajo hemos aprendido los aspectos más técnicos de realizar un proyecto serio, con posibilidades de implantarse en una empresa real. Una de las dificultades más grandes ha sido el simple hecho aprender a distribuirse adecuadamente el tiempo, y dividir el trabajo en fases con objetivos razonables. La planificación es fundamental.

Por otra parte, en cuanto a dificultades técnicas hemos tenido especial trabajo en el diseño de las funciones y algoritmos complejos, procurando optimizar el gasto de recursos que generan, como por ejemplo, la memoria que consumen. El hecho de trabajar bajo el lenguaje C++ ha mostrado la gran importancia de administrar a la perfección la gestión de memoria.

En lineas más generales, una conclusión que sacamos al finalizar el proyecto y ver sus resultados es que podemos afirmar que las herramientas que actualmente existen destinadas al procesamiento de datos y a la visión por computador son de gran calidad, y su potencia es muy grande. Además, tal y como hemos podido comprobar, existe mucha actividad relacionada con las tareas de mejora y crecimiento de las plataformas que dan soporte a esta herramientas, motivo que augura un futuro prometedor en este campo.

Otra conclusión que deducimos después de recoger los datos observados en los test, es que para trabajar y aprender iniciándose a nivel de investigación no es necesario un gran equipo ni grandes recursos tanto de infraestructura como de financiación. La enorme cantidad de información que proporciona internet junto a publicaciones científicas que podemos encontrar en las librerías, supondrán una herramienta indispensable y al alcance de muchos para desarrollar nuestros conocimientos en esta materia. Un recurso fundamental serán las ganas y el empeño que pongamos en nuestra labor.

5.2 - Trabajo futuro

Después de ver el proyecto finalizado, podemos hablar sobre las posibles mejores que se podrían llevar a cabo en un futuro cercano. La primera que intuimos es la optimización de los recursos utilizados. Sería bueno investigar en que proporción deberíamos hacer crecer nuestras colecciones de imágenes para mejorar el rendimiento de la aplicación. Por otro lado, la inclusión de nuevos algoritmos de cálculo y la mezcla de algunos de ellos podría ser una alternativa más eficiente a los que hemos planteado en este trabajo.

Finalmente, la mejora de la interfaz de trabajo es una de las opciones más claras. Implementar un modelo de uso en el que se facilite el manejo de la aplicación, y se flexibilice las opciones de uso pueden constituir una mejora notable a un bajo coste.

Cualquier otro trabajo y mejoras relacionados con el tema que hemos desarrollado serán siempre bienvenidos.

6. - Referencias

[1] Alexander Kuranov, Rainer Lienhart, and Vadim Pisarevsky. “An Empirical Analysis of Boosting Algorithms for Rapid Objects With an Extended Set of Haar-like Features.” Intel Technical Report MRL-TR-July02-01, 2002.

[2] HaarTraining doc. Este documento puede encontrarse en OpenCV/apps/HaarTraining/doc en el directorio de instalación de OpenCV.

[3] Navneet Dalal, Bill Triggs, "Histograms of Oriented Gradients for Human Detection," cvpr, vol. 1, pp.886-893, 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1, 2005.

[4] Yoav Freund, Robert E. Schapire, “A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting” (1995).

[5] Wayne Iba and Pat Langley. (1992), “Introduction of One-Level Decision Trees.”

[6] Paul Viola and Michael J. Jones. “Rapid Object Detection using a Boosted Cascade of Simple Features.” IEEE CVPR, 2001.

[7] Fatih Porikli. “Integral Histogram: A Fast Way to Extract Histograms in Cartesian Spaces.” Mitsubishi Electric Research Laboratories, TR2005-057 December 2005.

[8] Gary Bradski, Adrian Kaehler. “Learning OpenCV: Computer Vision with the OpenCV Library”.

[9] http://opencv.willowgarage.com

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

[11] http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2006/slides/dalal.pdf

[12] www.wikipedia.com

7. - Apéndice

7.1 - Contenido CD

El contenido del CD que se facilita junto con la memoria es el siguiente:

- Carpeta raíz "PFC", dentro de la cual encontramos la memoria en formato .pdf, una carpeta llamada "Imágenes" y otra carpeta llamada "Código".

- Dentro de "Imágenes" encontramos las carpetas "Positivas" y "Negativas", que contienen las imágenes a partir de las cuales se ha generado el clasificador.

- Dentro de la carpeta "Código", encontramos una carpeta llamada "PFC_Beta", que contiene todo el proyecto generado por Visual Studio.

En la carpeta "PFC_Beta" encontramos las carpetas "Debug" y "Release", donde cada una contiene el ejecutable propio de la aplicación. También existe otra carpeta con el nombre "PFC_Beta", y dentro de la cual se ubican los archivos .h, los archivos de código .cpp, el clasificador utilizado con el nombre "salidahaarcascade3.xml" y las diferentes carpetas con los contenedoras de los modelos de avión redimensionados y listos para ser tratados. Se recomienda que para probar la aplicación se edite el proyecto con Visual Studio, y una vez dentro del entorno de trabajo, generemos la solución y encendamos la aplicación.

7.2 - Instalación Visual Studio

Para instalar Visual Studio sólo necesitamos el programa y instalarlo en la ubicación que deseemos. Recomendamos instalar Visual Studio 2008 o superior.

7.3 - Instalación de las OpenCV

A continuación explicamos los pasos a seguir para instalar las OpenCV. Damos por hecho que el usuario ha instalado previamente el entorno de trabajo Visual Studio.

- Descargar las librerías desde http://sourceforge.net/projects/opencvlibrary. Recomendamos instalar la versión 2.0 o superior. En nuestro proyecto hemos trabajado la versión final con las librerías 2.3.
- Durante la instalación elegir la opción de añadir las OpenCV al path para todos los usuarios.
- Descargar e instalar "cmake" desde http//:www.cmake.org.
- Reiniciar el ordenador.
- Crear el proyecto y compilar las OpenCV. Desde la carpeta donde están instaladas podemos ejecutar la instrucción: `$ cmake . -G "Visual Studio 2008"`
- Abrir el proyecto que se haya creado en la carpeta OpenCV.
- Crear la solución en modo Debug y Release.
- Añadir a las variables de sistema las rutas indicadas y reiniciar el ordenador:
- `C:\OpenCV2.3\bin\Release`
- `C:\OpenCV2.3\bin\Debug`

- Abrir el compilador y añadir en **Tools->Options->Projects->VC++ Directories->Include files** el path:
- `C:\OpenCV2.3\include\opencv`
- Añadir en **Tools->Options->Projects->VC++ Directories->Library files** los path:
- `C:\OpenCV2.3\lib\Release y C:\OpenCV2.3\lib\Debug`
- Para cada proyecto que se cree, añadir en **Project->Properties->Configuration Properties->Linker->Input->Additional dependencies:**
- `opencv_core230d.lib`
- `opencv_highgui230d.lib`
- `opencv_ml230d.lib`
- para la configuración Debug, y para la configuración Release:
- `opencv_core230.lib`
- `opencv_highgui230.lib`
- `opencv_ml230.lib`
- Añadir los siguientes includes al proyecto:
- `#include "cv.h"`
- `#include "cxcore.h"`
- `#include "highgui.h"`

Con todo esto ya tenemos el equipo configurado para funcionar y utilizar las librerías OpenCV.

7.4 - Instalación Cygwin

La herramienta de trabajo Cygwin podemos descargarla desde www.cygwin.com. Instalaremos la versión más moderna y cada usuario escogerá que paquetes desea incorporar, recomendando el uso de todos los relacionados con el procesamiento de imágenes.