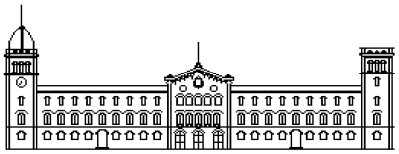


Treball fi de carrera

ENGINYERIA TÈCNICA EN INFORMÀTICA DE SISTEMES

Facultat de Matemàtiques
Universitat de Barcelona

RECONOCIMIENTO DE OBJETOS POR DESCRIPTORES DE FORMA

Julián Sanz Cuenca

Director: Sergio Escalera Guerrero
Realitzat a: Departament de Matemàtica Aplicada i Anàlisi. UB

Barcelona, 10 de juny de 2008

1. INDICE

2. RESUMEN

2.1 Resumen

El objetivo de los sistemas diseñados para reconocer objetos se basa en la capacidad de responder de forma correcta ante una nueva imagen de entrada semejante a aquellas con las que el sistema ha sido entrenado. Para ello, se realiza una primera fase de entrenamiento que consiste en la extracción de las principales características de un conjunto de imágenes de muestra, y en una segunda fase la información es clasificada y almacenada en diferentes clases. En este contexto, presentamos en este proyecto un sistema capaz de reconocer objetos formado por un descriptor de forma basado en algoritmos como K-NN y distancia euclídea y por un clasificador ECOC basado en el algoritmo Gentle Adaboost y distancia euclídea. El sistema propuesto ha sido validado en bases de datos públicas realizando una comparativa entre diferentes sistemas existentes mostrando un alto rendimiento.

2.2 Resum

L'objectiu dels sistemes dissenyats per al reconeixement d'objectes es basen en la capacitat de respondre de forma correcta enfront una nova imatge d'entrada semblant a d'altres amb que el sistema ha sigut entrenat. Per fer-ho, cal realitzar un primera fase d'entrenament que consisteix en la extracció de les principals característiques d'un conjunt de imatges de mostra, y una segona fase en que la informació es classificada y emmagatzemada en diferents classes. En aquest context, presentem en aquest projecte un sistema capacitat per al reconèixer objectes format per un descriptor de forma basat en els algorismes K-NN i Distància Euclídea i per un classificador ECOC basat en els algorismes Gentle Adaboost i Distància Euclídea. El sistema proposat ha sigut validat en bases de dades públiques realitzant una comparativa entre diferents sistemes existents mostrant un alt rendiment.

2.3 Abstract

The goal of the design systems for the object recognition are based in the capability to response of correct method in front one new entry image similar to another who the system had trained. To build this, is necessary make a first phase of training that consist in the extraction of the principal characteristics for the set of examples images, and in other phase the information is classify and saved into different classes. In this context, we propose in this project an efficient system that have capacity to object recognition, this system ,make up for a shape descriptor based on K-NN and Euclidean Distance algorithm and the other hand is compost for a ECOC classifier based on Gentle Adaboost and Euclidean Distance. The present system is validated in public databases make a comparative between different existent systems, showing high performance.

3. INTRODUCCION

Motivado en gran parte por su relación natural con los sentidos, es indudable la importancia del tratamiento de material gráfico en el mundo contemporáneo. La percepción en sí no es más que un reconocimiento en el que se establece previamente un modelo estadístico basado en una información que para las personas nos resulta familiar.

Se han llevado a cabo numerosas investigaciones con el propósito de desarrollar un modelo estadístico ideal. Con el rápido avance de las tecnologías ha aumentado la diversificación de técnicas utilizadas para cumplir este objetivo. La inteligencia artificial y la visión por computador son dos de las más importantes.

La inteligencia artificial se centra en el desarrollo de procesos que imitan la inteligencia de los seres vivos, concretamente intenta resolver problemas de interpretación, aprendizaje y razonamiento cognitivo. Como ejemplo de técnicas utilizadas en el campo de la inteligencia artificial destacaríamos el razonamiento basado en casos, los sistemas expertos, las redes bayesianas [1] y la inteligencia artificial [2] basada en comportamiento.

La visión por computador [3] podría considerarse como un subcampo dentro de la inteligencia artificial y se centra básicamente en intentar expresar el proceso de visión en términos de computación. Algunas de las técnicas más utilizadas en el campo de la visión por computador son el procesamiento de imágenes, los gráficos por computador y el reconocimiento de patrones, y a diferencia de las anteriores, éstas serán tratadas a lo largo de la memoria de este proyecto.

Centrándonos en la visión por computador, es importante destacar que el principal objetivo a alcanzar por todas las investigaciones realizadas en este campo está basado en conseguir la máxima similitud entre la visión artificial que utilizan los ordenadores y la visión biológica utilizada por los seres vivos. La visión es, sin duda, nuestro bien más poderoso y a la vez el más complejo. En un principio se pensó que crear un sistema de visión por computador era bastante fácil y basaron su teoría en que los ordenadores eran muy potentes y que si eran capaces de realizar operaciones matemáticas muy complejas, cosa muy difícil para el ser humano, incorporar visión a éstos no aportaría gran esfuerzo.

Los primeros experimentos sobre sistemas de visión artificial fueron un fracaso rotundo, la explicación de este hecho reside en lo superficial y erróneo del razonamiento anterior, ya que mientras que los seres humanos son conscientes de las etapas involucradas en la resolución de problemas matemáticos y es por ello que son conscientes de la complejidad del proceso, la mayoría del procesamiento que se realiza en la percepción visual se lleva a cabo de forma inconsciente, siendo por ello difícil de imitar o transportar a un sistema informático.

La importancia de todas las investigaciones realizadas en el campo de la visión por computador se hace evidente al observar la alta demanda por parte de la industria y la sociedad de aplicaciones capacitadas para reconocer objetos. Como ejemplo de este tipo de aplicaciones destacaríamos el caso de Fujitsu que ha diseñado robots que tienen la capacidad de reconocer imágenes en tres dimensiones. También es importante destacar el uso de estas tecnologías aplicadas en los nuevos buscadores de internet. Proyectos

como Riya ó VIMA entre otros investigan la creación de buscadores de páginas web que además de poder localizar texto como puede hacer google, permitan buscar contenidos multimedia gracias al uso de algoritmos de reconocimiento de imágenes. Éstos y otros ejemplos aparecen citados en el apéndice A del proyecto.

Basándonos en algunas de estas técnicas utilizadas en el reconocimiento de imágenes, diseñaremos y construiremos un sistema capacitado para reconocer objetos en imágenes. El sistema a grandes rasgos estará dividido en dos módulos, el primero de ellos será el descriptor cuya finalidad será extraer valores óptimos de las características de las imágenes previamente presentadas al sistema. Como base del descriptor utilizaremos el BSM (Bluerred Shape Model) desarrollado por investigadores de la Universidad Autónoma de Barcelona. El segundo módulo del sistema será el clasificador, cuya misión consistirá en distribuir las características extraídas de las imágenes en clases diferenciadas. Para ello utilizaremos como base algoritmos de clasificación como K-NN, Adaboost, y combinación de estos utilizando el marco de los códigos correctores de errores (ECOC) [4].

3.1 Objetivos

Diseñar, desarrollar y testear software capacitado para el reconocimiento automático de imágenes en blanco/negro y en color basado en técnicas de procesamiento de imágenes.

El sistema tendrá el objetivo de extraer las características visibles de la imagen, concretamente la forma de ésta, para posteriormente diferenciarla del resto. Para realizar esta diferenciación diseñaremos un descriptor basado en la teoría del BSM (Bluerred Shape Model).

Realizado el proceso de extracción de las características el sistema, se permitirá el reconocimiento de imágenes. Para incorporar esta funcionalidad se implementará un clasificador binario, basando su diseño en un modelo ECOC (técnica de códigos de salida de corrección de error).

El sistema creado será testeado para comprobar su correcto funcionamiento así como también se realizarán comparaciones entre el diseño de este sistema y otros objetos en algoritmos utilizados para la implementación de sistemas capacitados para reconocer imágenes.

3.2. Propuesta

Las imágenes serán analizadas y procesadas de forma individual, aunque la aplicación permite procesar un conjunto de imágenes de forma secuencial. Las imágenes que se introducen en el sistema son imágenes en blanco/ negro y color ubicadas en un directorio específico del ordenador. No son obtenidas a partir de ninguna cámara. El programa solicita la ruta necesaria donde permanecen las imágenes para poder trabajar posteriormente con ellas.

En el momento de introducir la imagen o imágenes en cuestión a procesar, también es necesario que sean informados los parámetros de entrada (alto, ancho) necesarios para la elección de regiones en que se desea descomponer la imagen.

Realizada ya la inserción de la imagen, el sistema inicia el proceso de extracción de los valores óptimos de las características de la imagen. La característica utilizada en este sistema se basa en la forma de la imagen.

A lo largo de la memoria de este proyecto se mostrará con detalles la estructura de un sistema diseñado para reconocer objetos, estudiando los diferentes módulos de los que se compone y analizando las posibles técnicas a utilizadas en el diseño de cada uno. Siguiendo esta estructura los capítulos 3 y 4 nos introducen en el diseño del descriptor y clasificador respectivamente, nos muestran las posibles técnicas y algoritmos empleados en los diferentes sistemas utilizados en la actualidad. En el capitulo 5 nos introducimos en la elaboración de nuestro sistema, explicando su composición, diseño e implementación, analizando en todo momento las técnicas utilizadas. A continuación encontramos en el capítulo 6 la interfaz diseñada para poder trabajar con nuestro sistema. Finalizado el proceso de diseño de la aplicación en el capitulo 7 realizamos una serie de experimentos y analizamos los resultados obtenidos, comparando nuestro sistema con otros existentes.

4. DESCRIPTOR

El algoritmo utilizado para extraer las características de las imágenes introducidas en el sistema está basado en el descriptor BSM desarrollado por el centro de visión por computador de la Universidad Autónoma de Barcelona.

4.1 Descripción BSM (Bluerred Shape Model)

4.1.1 Introducción

Para poder analizar y reconocer símbolos, el sistema divide las regiones de la imagen en subregiones (con la ayuda de una rejilla) y guarda la información de cada celda de la rejilla, de ésta forma se permite registrar cada pequeña diferencia. Dependiendo de la forma introducida, el sistema inicializa un proceso de reconocimiento de forma y al mismo tiempo comprueba cualquier posible deformación de ésta. Una vez finalizado el proceso y recopilada dicha información la imagen se clasifica automáticamente. Para comprobar la eficacia del sistema se han realizado experimentos que podemos encontrarlos en el capítulo 10.

4.1.2 Detección de Bordes

Para obtener la forma de la imagen a partir de la cual se calcula el BSM es necesario obtener los bordes o contorno de la imagen.
Los bordes de una imagen [6] se pueden definir como transiciones entre dos regiones de niveles de gris significativamente distintos. Estos nos facilitan una valiosa información sobre las fronteras de los objetos que nos interesan reconocer del resto de la imagen.

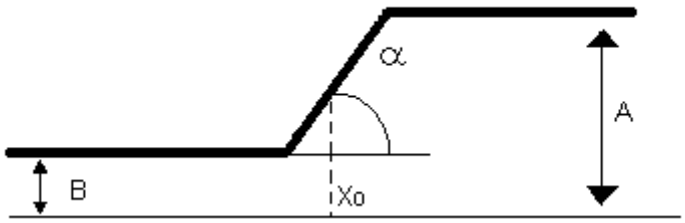


Fig. 1: Modelo unidimensional y continuo de un borde ideal

La figura 1 muestra un modelo unidimensional y continuo de un borde.
Este modelo representa una rampa desde un nivel de gris bajo *B* a uno alto *A*.
H corresponde a la variación de la intensidad. Se calcula de la siguiente forma

$$H = A\text{-}B$$

donde α corresponde al ángulo de inclinación de la rampa α y
X_0 corresponde a la coordenada horizontal X_0 donde se encuentra el punto medio de la rampa.
Un operador que proporcionara los valores de X_0 y H daría unos datos muy valiosos sobre la imagen, ya que proporcionaría la amplitud del borde, y la localizaría con exactitud dentro de la imagen.

En las imágenes reales los bordes nunca se ajustan totalmente al modelo anterior. Las causas que ocasionan este mal ajuste son diversas:

- Las imágenes son discretas
- Las imágenes están afectadas por ruido diverso
- El origen de los bordes puede ser muy diverso: bordes de oclusión, superficies de diferente orientación, distintas propiedades reflectantes, distinta textura, efectos de iluminación (sombras y/o reflejos), etc.

Debido a las circunstancias citadas anteriormente, el proceso de obtención de bordes de una imagen adquiere una gran complejidad.
Además existen en este proceso diferentes tipos de errores:

- **Error de detección:** Un operador es un buen detector si la probabilidad de detectar el borde es alta cuando éste realmente existe en la imagen, y baja cuando no existe.
- **Error en la localización:** Un operador localiza bien un borde cuando la posición que proporciona coincide con la posición real del borde de la imagen.

Ambos errores están íntimamente ligados a los problemas comentados anteriormente, y sobretodo a la presencia de ruido en la adquisición de la imagen. En la práctica, la calidad de detección y localización están en conflicto.

- **Respuesta múltiple:** Es el caso en que se detectan varios píxeles en un único borde.

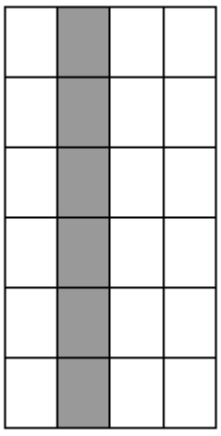


Fig. 2: Borde real

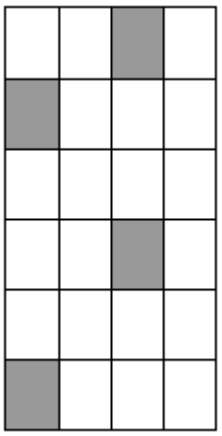


Fig. 3: Pobre detección

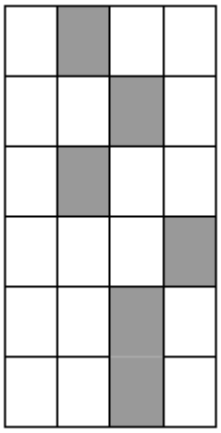


Fig. 4: Pobre localización

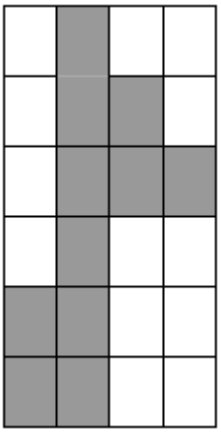


Fig. 5: Múltiple respuesta

La fig.2 muestra la posición de un borde real, como se ha comentado en el punto anterior para garantizar la extracción óptima de la forma del objeto nuestro localizador ha de proporcionarnos valores ajustados a este borde. En ocasiones, a causa de la distorsión o de la incorrecta elección del localizador los valores de los bordes obtenidos no son correctos. Las fig. 3-5 son ejemplos de una obtención defectuosa de los valores del borde.

La mayoría de las técnicas utilizadas en la detección de bordes utilizan operadores locales basados en distintas aproximaciones discretas de la primera y segunda derivada de los niveles de grises de la imagen, si bien existen otras probabilidades para ello, como el empleo de patrones de bordes ideales.

Las librerías de procesamiento de imágenes OpenCV poseen ya algunos métodos que implementan la funcionalidad de los algoritmos comentados anteriormente. Como ejemplos de algoritmos para extraer los contornos de una imagen destacaríamos Kirsch N [7], Kirsch N-E [8], Canny, Skeleton [9], Sobel[10].

El algoritmo que utiliza el sistema BSM para obtener los contornos o bordes de la imagen es el algoritmo de Canny. Podemos encontrar la descripción de este algoritmo en el apéndice 2 de este proyecto.

Nuestro sistema, como ya hemos comentado anteriormente, utiliza el operador de Canny para determinar el contorno de las imágenes a reconocer, pero los cálculos matemáticos que se han de realizar para obtenerlos no son implementados por nuestro sistema, sino que hacemos uso de las librerías de reconocimiento de imágenes OpenCV [12] que nos ofrecen la función cvCanny .

Función de Canny implementada en las OpenCV:

void cvCanny(const CvArr img, CvArr* edges, double threshold1, double threshold2, int apertureSize=3);*

- img : Imagen de entrada.
- edges : Imagen para almacenar los bordes encontrados.

- threshold1 : El primer umbral.
- threshold2 : El Segundo umbral.
- apertureSize : El mismo que para el operador Sobel.

La función cvCanny encuentra los bordes de la imagen original y marca en la imagen de salida los bordes encontrados usando el algoritmo Canny.

Los umbrales que se le introducen como parámetros a la función son los comentados anteriormente en la descripción de la búsqueda del operador de Canny concretamente en el requisito de buena detección de los bordes.

Imágenes con diferentes parámetros de entrada:

Fig 6: Imagen original

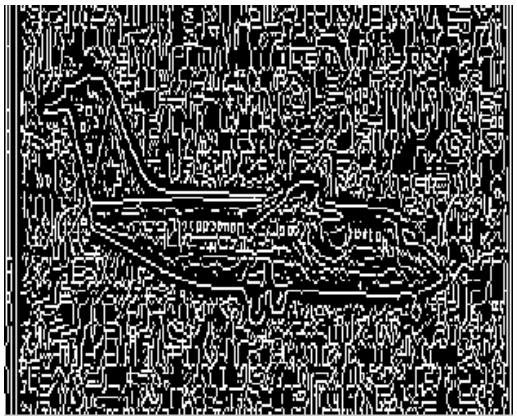


Fig 7: cvCanny(img, imgkny, 30,10, 80, 3);

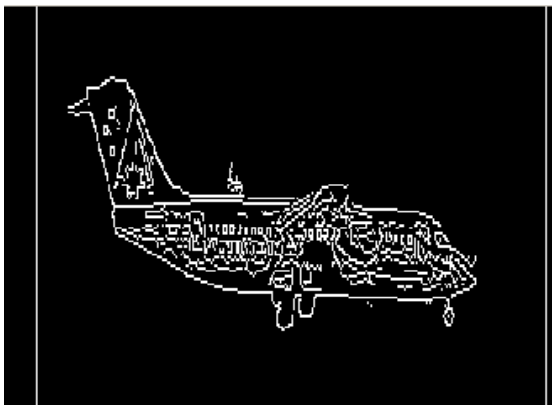


Fig. 8: cvCanny(img, imgkny, 100, 80, 3);

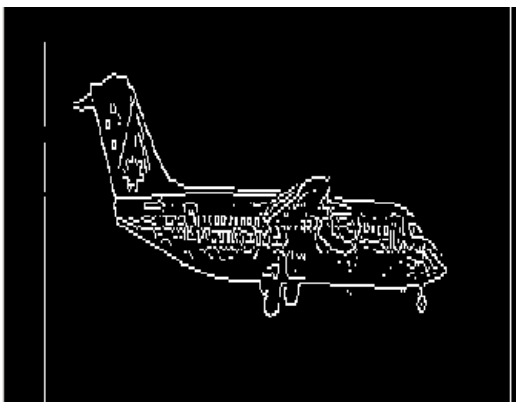


Fig. 9: cvCanny(img, imgkny, 100, 95, 3);

4.2 Descripción del algoritmo utilizado

Pasos de la función del descriptor:

- Inicializar a cero el vector de salida de longitud alto_grid x ancho_grid.

$$V = (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$$

- Dividir la imagen en regiones alto_grid x ancho_grid

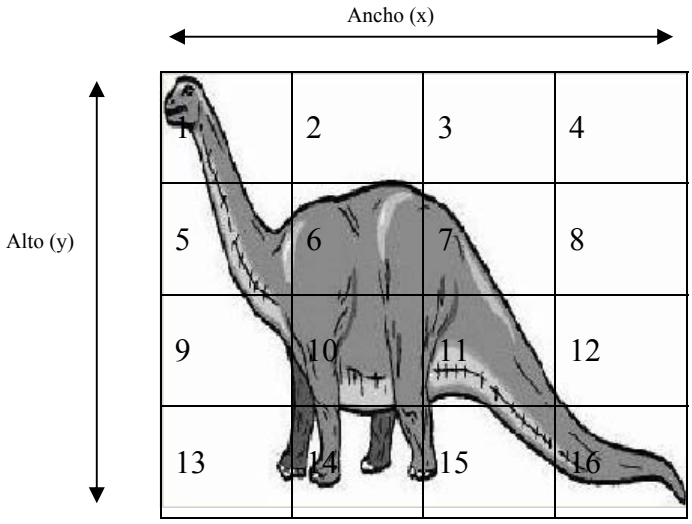


Fig. 10: Imagen de entrada dividida en regiones

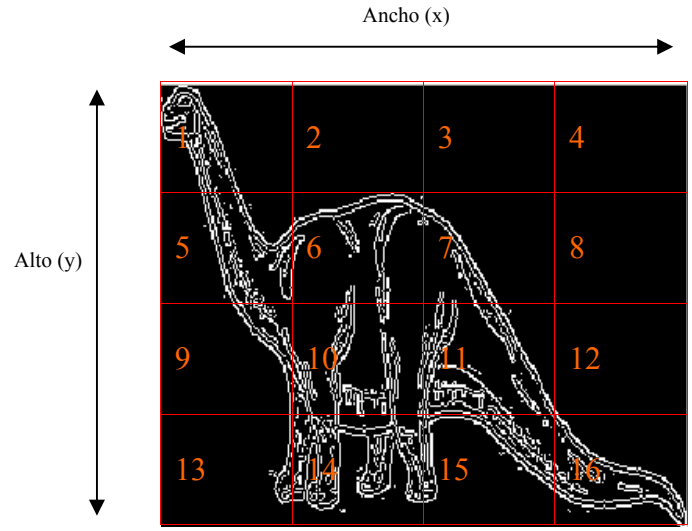


Fig. 11: Imagen salida aplicando algoritmo de Canny dividida en regiones

El sistema analiza cada región de forma independiente. Para cada región, se almacena información dependiente al resto de regiones.

Para cada región es necesario guardar una referencia de las regiones que la rodean. Las regiones que rodean a una región reciben el nombre de vecinos. Tener en cuenta que una región es vecina a sí misma.

Región	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16
1	x	x			x	x										
2	x	x	x		x	x	x									
3		x	x	x		x	x	x								
4			x	x			x	x								
5	x	x			x	x			x	x						
6	x	x	x		x	x	x		x	x	x					
7		x	x	x		x	x	x		x	x	x				
8			x	x			x	x							x	x
9					x	x			x	x			x	x		
10					x	x	x		x	x	x		x	x	x	
11						x	x	x		x	x	x		x	x	x
12							x	x			x	x			x	x
13									x	x			x	x		
14									x	x	x		x	x		
15										x	x	x		x	x	x
16											x	x			x	x

Fig. 12: Disposición de vecinos en imagen de 16 regiones

La fig. 12 muestra los vecinos que corresponden a cada región. Según el número de vecinos que tiene una región podemos clasificarlas en:

- Regiones con 8 vecinos: Son las regiones que se encuentran en el centro de la imagen.
- Regiones con 6 vecinos: Son aquellas regiones localizadas en los laterales de la imagen.
- Regiones con 4 vecinos: Son aquellas regiones localizadas en las esquinas.

Además de saber que vecinos tiene cada región también es necesario guardar el punto central de la región. Este punto se conoce con el nombre de centroide de la región. Las coordenadas del centroide se obtienen de la siguiente forma:

$$Cx = (Rxmax - Rxmin) / 2;$$

$$Cy = (Rymax - Rymin) / 2;$$

Anteriormente se ha visto que se ha utilizado el algoritmo de Canny para obtener los contornos de la imagen de entrada. La información de que puntos pertenecen al contorno se almacena en una matriz.

- Procesamos las regiones una a una.

Para cada punto de la región con valor 1 (existe contorno) se realizan los siguientes cálculos:

- Calcular distancias a todos los centroides y él mismo.

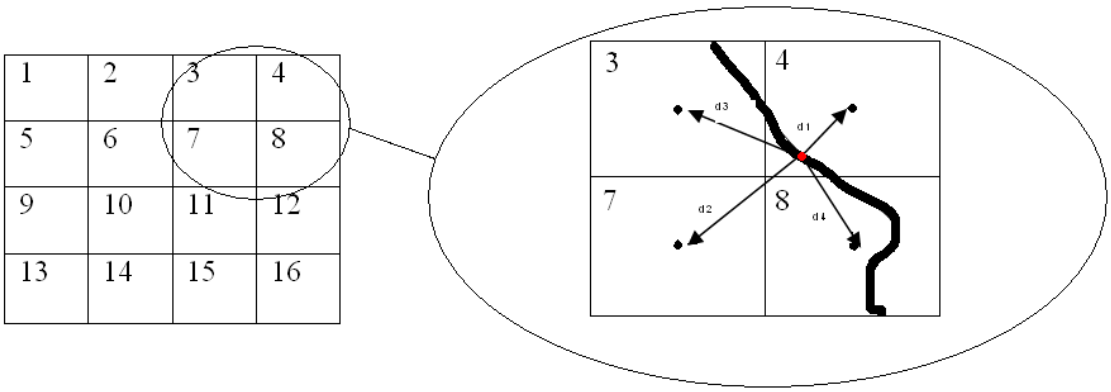


Fig 13: Calculo de distancia para un punto del contorno

$$V_{c4} = (0,0,d3,d4,0,0,d7,d8,0,0,0,0,0,0,0,0)$$

- Normalización del resultado obtenido: Consiste en dividir cada distancia por la suma total. Los elementos del vector resultante tendrán valores comprendidos entre [0..1]

$$Dt = \sum_{i=0}^{i=n} d_i \Longrightarrow V_{c4} = (0,0,d3/dt,d4/dt,0,0,d7/dt,d8/dt,0,0,0,0,0,0,0,0)$$

- Calcular la inversa de cada distancia.

- Normalización: Volver a dividir cada distancia por la suma total.

- Actualizar el vector salida sumando a las posiciones de las regiones afectadas el valor de las distancias obtenido en los pasos anteriores.

$$V_s = V_s + V_{c4}$$

$$V_s = (0,0,d3,d4,0,0,d7,d8,0,0,0,0,0,0,0,0)$$

- Normalización del vector salida: dividiendo cada distancia por la suma total.

El vector resultante, obtenido después de procesar todos los puntos de contorno de la imagen representa la función de densidad de probabilidades.

La salida del descriptor representa una distribución de probabilidades de la forma del objeto considerando distorsiones espaciales, donde el nivel de distorsión viene determinado por el tamaño de la rejilla.

Llegados a este punto ya tenemos el vector con la información perfectamente estructurada para ser almacenado.

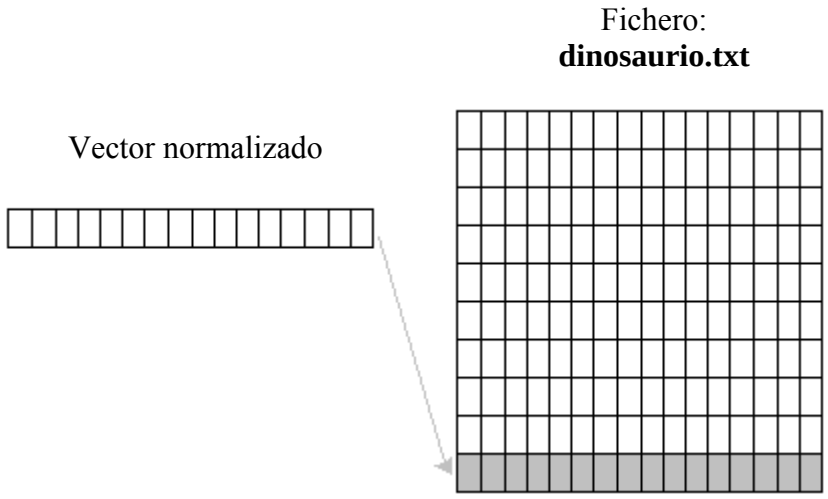


Fig. 14: Almacenamiento del vector de características en un fichero

4.3 Algoritmo en pseudocódigo Blurred Shape Model

Insertar una imagen binaria I,

Obtener la forma S contenida en I

Dividir I en n x n subregiones de idénticas dimensiones

$$R = \{ r_i,\ldots,r_{nxn}\}$$

Con c_i el centro de coordenadas de cada región

Dejar $N(r_i)$ ser la región vecina de la región r_i, definida como:

$$N(r_i) = \{r_k | r \in R, \| c_k - c_i \|^2 \leq 2 \text{ x } g^2\}$$

donde *g* es el tamaño de la celda

Para cada punto x Є S,

Para cada r_i Є $N(r_x)$

$$d_i = d(x,r_i) = \|x - c_i\|^2$$

donde *d* es la distancia del punto al centroide

Fin Para

Actualizar las probabilidades de las posiciones del vector v como:

$$v(r_i) = v(r_i) + \frac{1/d_i}{D_i} \qquad Di = \sum_{ck \in N(r_i)} \frac{1}{\| x - c_k \|^2}$$

Fin Para

Normalizar el vector v, de la siguiente forma:

$$v = \frac{v_i}{\sum_{j=1}^{n^2} v(j)} \forall_i \in [1,...,n^2]$$

5. CLASIFICADOR

Una vez definido el descriptor nos centraremos en el análisis del siguiente gran módulo de nuestro sistema, el clasificador. El objetivo del clasificador es analizar los datos obtenidos en la fase anterior para aprender a distinguir entre diferentes objetos. Este conocimiento puede ser expresado como reglas, patrones, asociaciones, relaciones o de forma general como modelo.

La imagen que aparece a continuación muestra a grandes rasgos el ciclo de trabajo de un modelo:

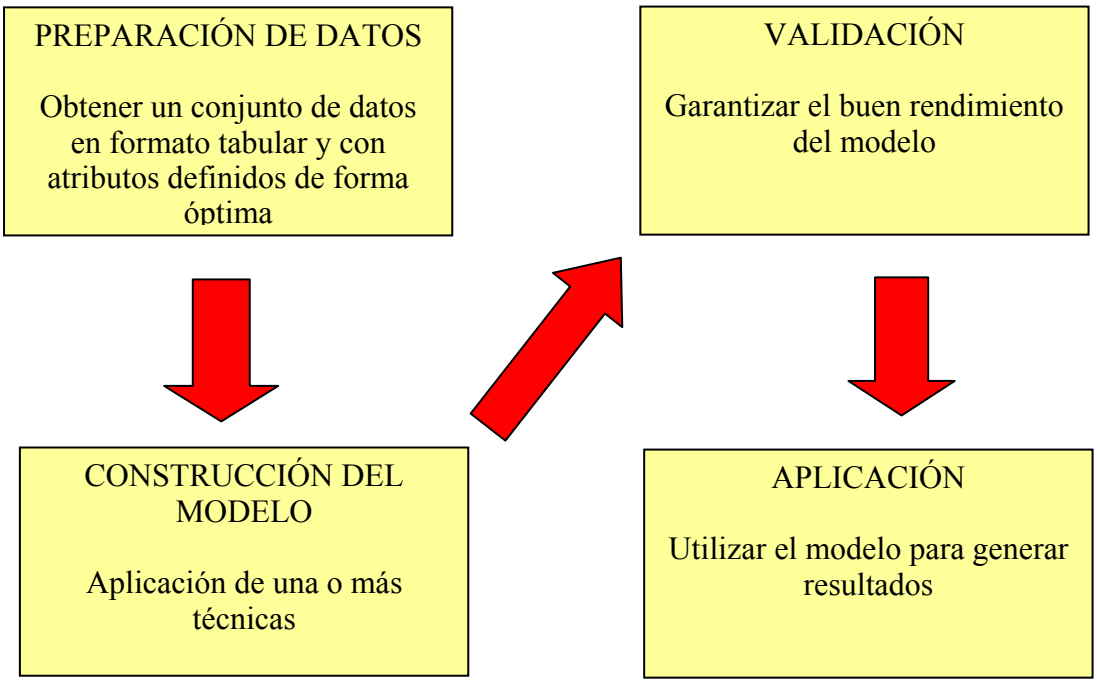


Fig. 15: Ciclo de trabajo de un modelo

Como acabamos de explicar y tal como se refleja en la fig. 15 el objetivo del clasificador es analizar datos para extraer conocimientos, partiendo de esta premisa nuestro objetivo consistirá en diseñar y construir un conjunto de modelos para aplicar a este clasificador y así proporcionarle el poder de clasificación y predicción de datos.

Existen diferentes tipos de modelos, como ejemplo, destacar aquellos modelos basados en Árboles de decisión, redes neuronales y redes Bayesianas. Estos modelos se describen en el apéndice C.

Los ejemplos que se muestran en el apéndice podrían catalogarse como tipos de modelos simples, y representan una parte del conjunto de técnicas utilizadas para la generación de modelos. En ocasiones es necesario utilizar modelos más complejos cuyo diseño se basa en las combinaciones de tipos simples.

En general el error de combinar varios clasificadores se explica por lo que se conoce como bias-variance decomposition. El sesgo (bias) de cada clasificador viene dado por su error intrínseco y mide lo bien que un clasificador explica el problema. La varianza está dada por los datos que se usan para construir el modelo.

El error esperado total de clasificación está dado por la suma del sesgo y la varianza. Al combinar múltiples clasificadores se reduce el error esperado ya que se reduce la varianza.

Ejemplos de modelos utilizando combinaciones homogéneas:

1. Bagging [15]: El algoritmo de Bagging (Bootstrap Aggregating) genera clasificadores de un sub-conjunto de muestras.

Es especialmente útil en algoritmos de aprendizaje inestables (cambian mucho sus estructuras al cambiar un poco los ejemplos), por ejemplo, los árboles de decisión.

Una muestra de ejemplos bootstrap se genera al muestrear uniformemente 'm' instancias del conjunto de entrenamiento con reemplazo. Se generan *T* muestras, $B_1, \ldots, B_T$ y se construye un clasificador Ci para cada muestra. Con estos, se construye un clasificador final C* de todos los C_1 a C_t cuya salida es la salida mayoritaria de los clasificadores.

Para una de las muestras, un ejemplo tiene la probabilidad de $1 - (1 - 1/m)m$ de ser seleccionado por lo menos una vez en las 'm' veces que se selecciona una instancia. Para valores grandes de 'm' se aproxima a $1 - 1/e = 63.2\ \%$. Por lo que cada muestra tiene aproximadamente un 63% de aparecer en los ejemplos de entrenamiento.

En general, se puede mejorar el resultado si se quita la opción de podado en los árboles de decisión, lo que los hace más inestables.

Los resultados de bagging, son difíciles de interpretar, además que en la práctica existe un solo conjunto de entrenamiento y al tratar de obtener más se vuelve poco práctico, o a veces, hasta imposible [15]. A continuación la fig.16 muestra en pseucódigo el algoritmo de bagging

Para i = 1 a T

S' = muestra de S (i.i.d con reemplazo)

Ci = Ind(S')

C* = argmar $_{y \in Y \Sigma i: ci(x)=y}$ ***1***

(i.i.d independiente e idénticamente distribuida)

Fig. 16: Algoritmo de bagging

1. Boosting [16]**:** La variante más utilizada es AdaBoost [17] (Adapting Boosting). Adabost genera un conjunto de clasificadores. A diferencia de Bagging que permite generarlos en paralelo, Adaboost únicamente permite la generación de forma secuencial.

El primer paso consiste en generar los ejemplos, a éstos se les asigna el mismo peso (1/m). A medida que se genera un nuevo modelo se cambian los pesos de los nuevos ejemplos utilizados para el siguiente clasificador. El objetivo es obligar al clasificador a minimizar el error esperado. Para esto se les asigna más peso a los ejemplos mal clasificados y menos a los bien clasificados.

Con este algoritmo se pretende crear modelos que se vuelvan expertos en los datos que no pudieron ser explicados por los modelos anteriores.

2. ECOC [18]**:** Es un marco de trabajo simple pero muy potente cuando es aplicado a problemas de categorización multiclase de objetos utilizando clasificadores. Se explicará con más detalle en las siguientes secciones.

Además de los ejemplos comentados anteriormente existen otro tipo de modelos de combinaciones heterogéneas, tales como Stacking [19] ó Gráding [20].

Analizadas las diferentes técnicas de generación, en este proyecto hemos utilizado K-NN (K-Nearest Neighbor), distancia euclídea, Adaboost y los Códigos de Corrección de Errores, como técnicas base del modelo de nuestro sistema.

5.1 Algoritmo K-NN(K-Nearest Neighbor)

KNN (vecino más cercano) [21] ha sido uno de los métodos de clasificación utilizado en este proyecto. El principal motivo de nuestra elección es debido a que el algoritmo K-NN es un método de clasificación muy simple pero a la vez muy potente.

Para explicar su funcionamiento utilizaremos un ejemplo:

Supongamos que tenemos un conjunto de entrenamiento de N vectores de características. Estos vectores están agrupados en distintas clases (C_1, C_2, C_3,..). Supongamos también que se introduce en el sistema un nuevo vector de características. Nuestro objetivo sería determinar la clase a la que pertenece. Para determinarla el algoritmo K-NN busca los K vecinos más próximos al vector de entrada, y posteriormente realiza una valoración de las clases a las que pertenecen estos vectores vecinos. La valoración consiste en comparar las distancias existentes entre las clases. Los algoritmos utilizados en esta valoración pueden ser distintos, en nuestro sistema hemos utilizado la Distancia Euclídea.

5.2 Distancia Euclídea

La distancia Euclídea [22] es una de las métricas más utilizadas, ésta se obtiene a partir de una generalización del teorema de Pitágoras.

En un espacio tridimensional, la distancia euclídea entre 2 puntos (X_1,X_2,X_3) y (Y_1,Y_2,Y_3) viene dada por la siguiente ecuación:

$$d = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2}$$

Más concretamente, la distancia euclidiana entre dos puntos p1 y p2 es:

$$d(p_1, p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

5.3 Adaboost

Adaboost es un algoritmo utilizado para construir clasificadores sólidos utilizando la combinación lineal de clasificadores simples.

El primer paso consiste en generar los ejemplos, a éstos se les asigna el mismo peso (1/m). A medida que se genera un nuevo modelo se cambian los pesos de los nuevos ejemplos utilizados para el siguiente clasificador. El objetivo consiste en minimizar en cada iteración el error esperado. Es por ello que se asignan pesos superiores a los ejemplos mal clasificados. Con este algoritmo se pretende crear modelos que se vuelvan expertos en los datos que no pudieron ser explicados por los modelos anteriores.

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

En la fig. 17 aparece el algoritmo utilizado por Adaboost.

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

Inicializar los pesos D1(i) = 1/m

Para t = 1, ..., T:

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

2. Escogemos αt Є R,

3. Actualizamos

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

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

Como salida obtenemos un clasificador más fuerte:

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

Fig. 17: Algoritmo de Adaboost

5.4 ECOC (Técnica de códigos de salida de corrección de error)

El algoritmo utilizado por el ECOC se puede dividir en tres pasos:

- Agrupación y Codificación
- Comparación y Decodificación
- Análisis de resultados

5.3.1 Codificación

La codificación es el primer paso dentro del proceso del ECOC y su objetivo es asignar códigos identificadores a las clases que forman parte del problema. Supongamos que tenemos un conjunto de N_c clases, ECOC inicia el diseño de palabras código que serán asignarán a cada una de las clases.

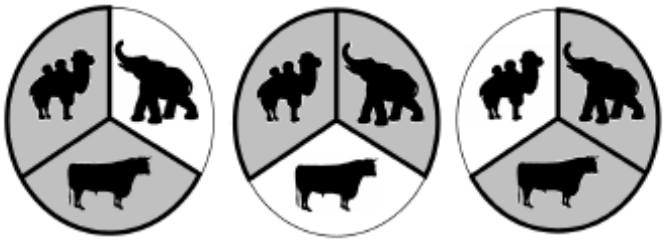


Fig. 18: Particiones binarias de clases
para un problema multiclase.

La fig. 18 muestra las diferentes agrupaciones que se pueden obtener a partir de emparejar 2 de las 3 clases que presentamos para este problema.

Posteriormente procedemos a ordenar los códigos asignados a las clases como filas de una matriz. Al finalizar el resultado obtenemos una matriz de códigos. Se define esta Matriz codificada, donde $M \in \{-1, 0, 1\}\ N_{cx\,n}$, siendo n la longitud del código. Desde el punto de vista del aprendizaje, *M* se ha construido considerando n problemas binarios, donde cada problema corresponde con una columna de la matriz.

Si unimos las clases en grupos, cada grupo define una partición de clases.

Estas clases son codificadas utilizando el siguiente patrón:

- Clase codificada con +1 o -1: Clases escogidas para el problema. La asignación de valor +1 o -1 a una clase depende del grupo al que pertenece en un problema binario.

- Clase codificada con 0: Si esta clase no es escogida para un determinado problema binario.

En la fig. 19 podemos observar una matriz codificada *M* diseñada para el ejemplo de a fig. 18.

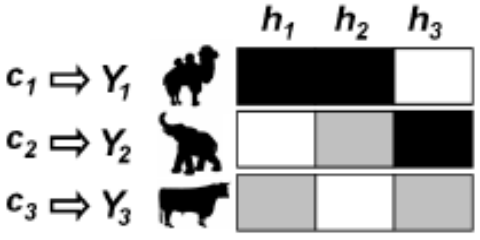


Fig. 19: Matriz codificada. Diseñada
para un problema de 3 clases

5.3.2 Comparación y Decodificación

Analizando la fig.19 observamos como cada región de la matriz tiene asignado un color diferente. Las regiones negras son codificadas con valor +1, las regiones blancas son codificadas con valor -1 y las regiones grises son las clases que no forman parte de la comparación.

Llegados a este punto, las filas de la matriz *M* definen las palabras código $\{Y_1, Y_2, Y_3\}$ para sus correspondientes clases $\{c_1, c_2, c_3\}$.

Para el paso de decodificación, aplicamos un clasificador binario [23] entrenado n veces. Utilizamos el clasificador para realizar las comparaciones, que se realizarán de forma secuencial. Para una mejor comprensión utilizaremos un ejemplo.

Objetivo del ejemplo: Reconocer una imagen de un camello.

Elementos que tenemos:

- Matriz de códigos *M* (fig. 19).
- Conjunto de 3 clases. Camellos, vacas y elefantes.
- En el paso anterior hemos asignado los códigos necesarios a cada clase.
- Clasificador binario entrenado

Realizamos las comparaciones utilizando el clasificador binario

comparación binaria	C 1	Código C1	C 2	Código C2	Comparación	Clase resultante	Código clase resultante
1		1		-1	→		1
2		1		-1	→		1
2		1		-1	→		-1

Fig. 20: Tabla de Comparaciones

Como resultado de aplicar el clasificador obtenemos un código para cada comparación.

Vector resultante X	1	1	-1

El código obtenido se compara con el código de cada clase definido en la matriz M, (que en el ejemplo), tenía codificadas todas las posibles clases para entrenar. Para realizar esta comparación utilizamos el método de distancia Euclídea explicado anteriormente.

$$d(X1, Y_i) = \sqrt{\sum_{j=1}^{n} (X(j) - Y_i(j))^2} \qquad \text{donde } i \in [1, .., 3]$$

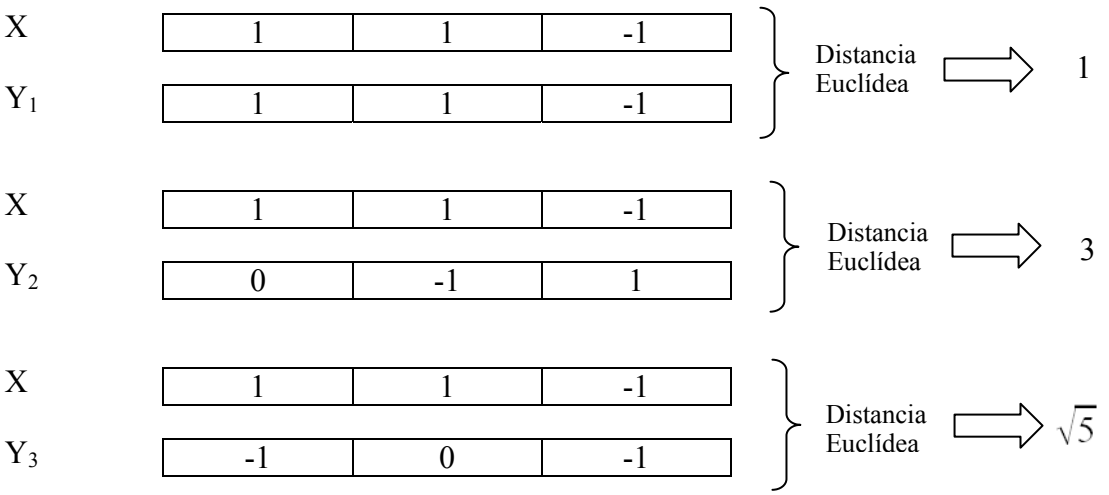


Fig. 21: Distancia Euclídea

Una vez realizadas todas las comparaciones, el ejemplo de testeo se asignará a la clase con el código más pequeño. En este caso es la clase C1 que corresponde a la clase Camello

6. SISTEMA

La mayoría de sistemas diseñados para reconocer imágenes se construyen siguiendo el mismo patrón. Como punto de partida se han de obtener imágenes del medio, seguidamente estas imágenes son procesadas con el descriptor con el fin de extraer las características necesarias, por último el clasificador hace uso de esta información en el proceso de reconocer nuevas imágenes.

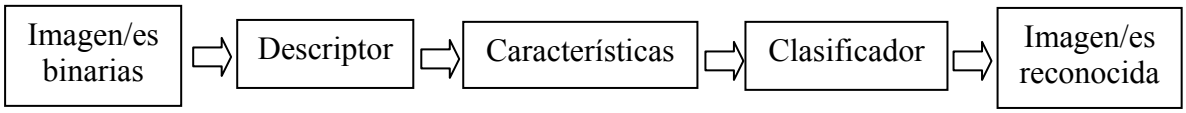


Fig. 22: Esquema general del sistema.

Utilizaremos el patrón comentado anteriormente e ilustrado en la fig. 22 como base del diseño de nuestro sistema. La problemática de obtener imágenes del medio no influye en nuestro sistema, ya que es el usuario que le indica la ruta donde se localizan y no es necesaria la incorporación de cámaras que capturen imágenes.
Una vez que el usuario indique la ruta donde se localizan las imágenes, el sistema mediante la interfaz gráfica permite al usuario escoger las que se desean procesar.
Las imágenes seleccionadas serán analizadas y procesadas de forma individual aunque la aplicación permite procesar un conjunto de imágenes de forma secuencial. Las imágenes que se introducen en el sistema son imágenes en blanco/ negro y color, que carecen de deformaciones y de rotaciones ya que el sistema no se ha diseñado para contemplar estos casos. En el momento de introducir la imagen o imágenes en cuestión a procesar, también es necesario que sean introducidos los parámetros de entrada (alto, ancho) necesarios para la elección de regiones en que se desea descomponer la imagen.

Realizada la inserción de la imagen, el sistema inicia el proceso de extracción de los valores óptimos de las características de la imagen. La característica utilizada en este sistema es la forma de la imagen. El módulo encargado de realizar esta tarea es el descriptor.

Como ya se ha comentado en capítulos anteriores, el primer paso del descriptor consiste en obtener los bordes de la imagen. Nuestro descriptor utiliza el algoritmo de Canny para extraer estos bordes.

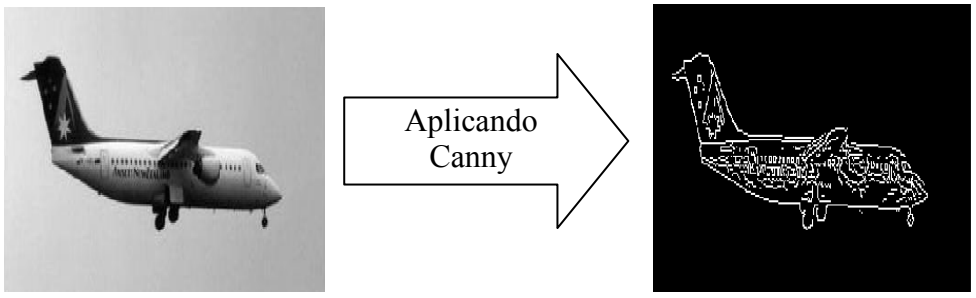


Fig. 23: Paso-1. Descriptor. Aplicación del algoritmo de Canny.

Como muestra la fig. 23, el algoritmo de Canny recibe una imagen de entrada y retorna una imagen en blanco y negro que contiene únicamente los bordes de la imagen de entrada.

El segundo paso del descriptor requiere dividir la imagen en regiones. El número de regiones dependerá de los parámetros introducidos previamente por el usuario.

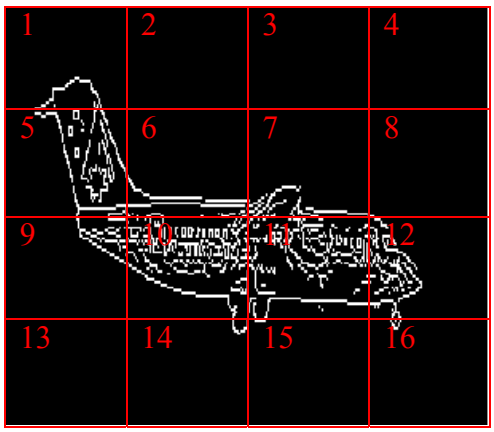


Fig. 24: Paso-2. Descriptor. División de imagen en regiones.

El sistema analiza cada región de forma independiente. Para cada región el sistema almacena información. Ésta consta de identificador, centroide y sus regiones vecinas. El sistema contiene la información de qué puntos pertenecen al contorno almacenado en una matriz. Se recorren los puntos del contorno y se aplican las operaciones del descriptor presentadas en el apartado del descriptor.

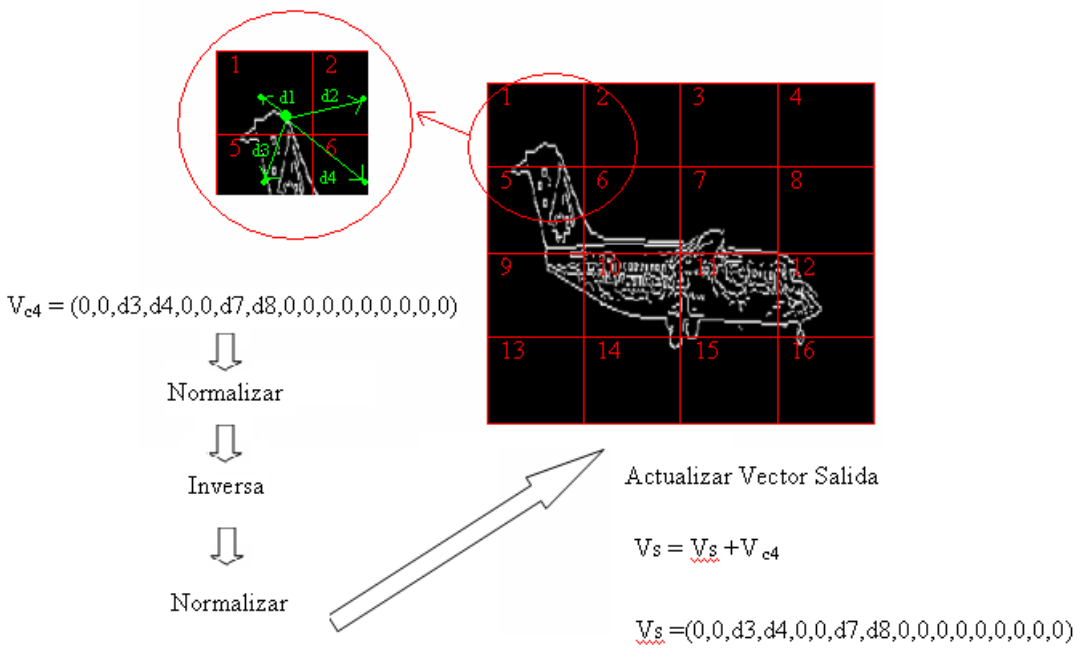


Fig. 25: Paso-4. Descriptor. Cálculo de vector de características.

Como resultado de los cálculos matemáticos obtenemos un vector de tamaño n x m (dimensiones de la matriz de regiones) que contiene las características óptimas de la forma de la imagen. Nuestro sistema permite la división de la imagen en regiones, donde el número de éstas es elegido por el usuario, y el tamaño de los vectores de características dependerá del número de regiones en que sea dividida la imagen. Para una mejor comprensión ilustramos en las figuras los valores del descriptor en forma de imagen.

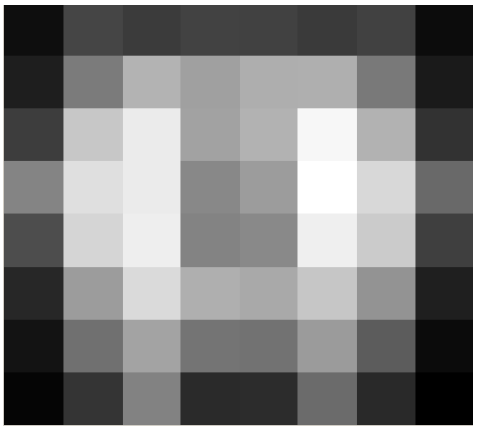


Fig. 26: Representación de la imagen (8x8)

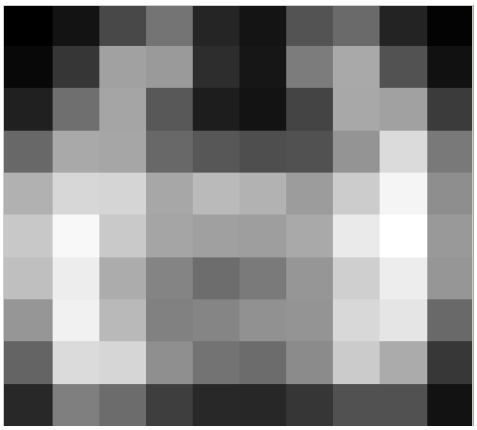


Fig. 27: Representación de la imagen (10x10)

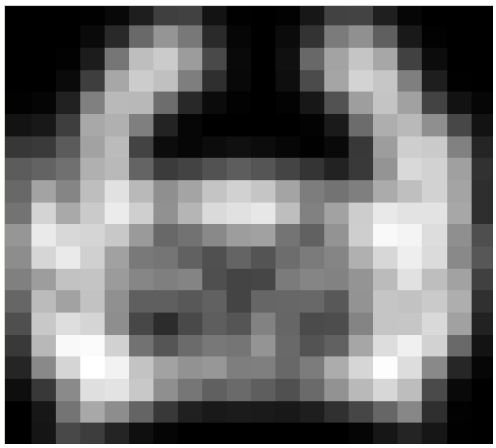


Fig. 28: Representación de la imagen (20x20)

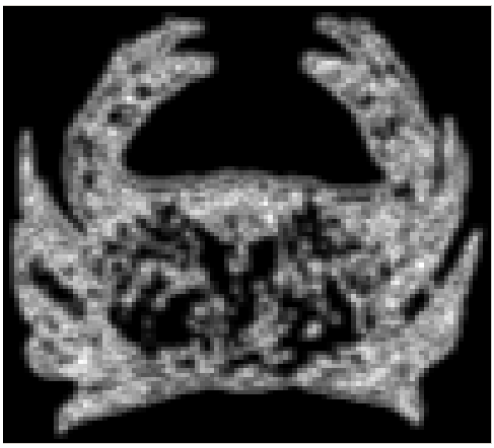


Fig. 29: Representación de la imagen (100x100)

Para obtener las imágenes mostradas en las fig. 26, 27, 28, 29 se ha asignado el mismo valor a todos los píxels que pertenecen a una región. El valor asignado es el valor de uno de los elementos del vector de características. El número del elemento corresponde con el número de la región. Comentar que a medida que aumenta el número de regiones los cálculos necesarios en cualquier proceso del sistema aumentan potencialmente, y puede además aportar distorsiones en los datos creando así un factor negativo para el sistema. La salida del descriptor representa una distribución de probabilidades de la forma del objeto considerando distorsiones espaciales, donde el nivel de distorsión viene determinado por el tamaño de la rejilla.

Supongamos como muestra la fig. 30, que se inroducen 40 imágenes de 4 objetos diferentes (10 de cada objeto), y supongamos también que el descriptor ya tiene calculados los vectores de carácterísticas de cada imagen y se encuentran en ficheros separados por clases, una clase por tipo de objeto, y cada objeto con un código asociado que lo identifica. Entonces el sistema ya estará listo para poder reconocer imágenes de 4 tipos. Ahora depende del usuario ya que es el encargado de volver a introducir una imagen para comprobar el correcto funcionamiento del clasificador y el reconocimiento de la imagen.

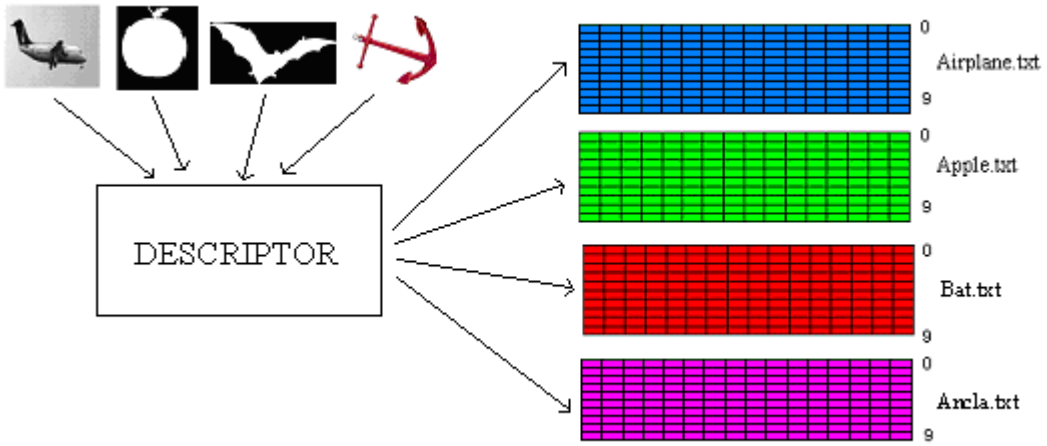


Fig. 30: Paso-5. Descriptor. Descriptor entrenado para reconocer 4 tipos de imágenes

En el proceso de reconocimiento de la nueva imagen se diferencian dos pasos, siendo el primero de ellos prácticamente igual al que acabamos de explicar. Después de que la imagen sea introducida en el sistema, ésta es procesada por el descriptor con el fin de extraer sus características, con la diferencia que ahora no guardamos el vector de características en ningún fichero matricial, sino que lo mantenemos en memoria del sistema. Finalizada esta primera etapa entra en juego el clasificador.

El clasificador antes de trabajar con el vector de características ha de realizar diversas tareas:

1- Recorrer la carpeta destino donde se encuentran las clases con la información obtenida por el descriptor. En esta primera tarea el clasificador obtiene información del número de clases (4 en este ejemplo) que hay en el sistema y que códigos (nombre) se asigna a cada una.

2- Generar la matriz de códigos, ésta dependerá del número de clases.

   a. Tamaño de la matriz

      i. $T_M = \dfrac{(n \times (n-1))}{2}$

b. Algoritmo utilizado para generar la matriz de forma dinámica

```
c = 0

Para i = 1 a n-1
        Para j = i+1 a n
                M[i][c] = 1;
                M[j][c] = -1;
                c ++;
        Fin Para
Fin Para

C → contador
```

Fig. 31: Algoritmo que genera una Matriz codificada *MC* de n clases

c. Ejemplo de matriz

	h1	h2	h3	h4	h5	h6
C1	1	1	1	0	0	0
C2	-1	0	0	1	1	0
C3	0	-1	0	-1	0	1
C4	0	0	-1	0	-1	-1

Fig. 32: Matriz codificada. Diseñada para un problema de 4 clases

La matriz que aparece en la Fig 32 se almacena en memoria ya que será utilizada más adelante. Finalizada la obtención de esta información el clasificador inicia el proceso de comparación binaria. Para ello el clasificador va seleccionando las clases del sistema de dos en dos. El clasificador localiza los ficheros ubicados en el PC asociados a las 2 clases escogidas para la comparación, lee el contenido de los ficheros y lo encapsula en una matriz de tamaño 20x17. Para no confundirnos denotaremos a esta matriz como MD (matriz de comparación). Cada fila de esta matriz corresponde con el vector de características de una imagen más un dígito identificador de la clase (+1 o -1). El clasificador utiliza como primer operando de la comparación el vector de características asociado a la imagen de entrada y como segundo operando una de las filas de la matriz.

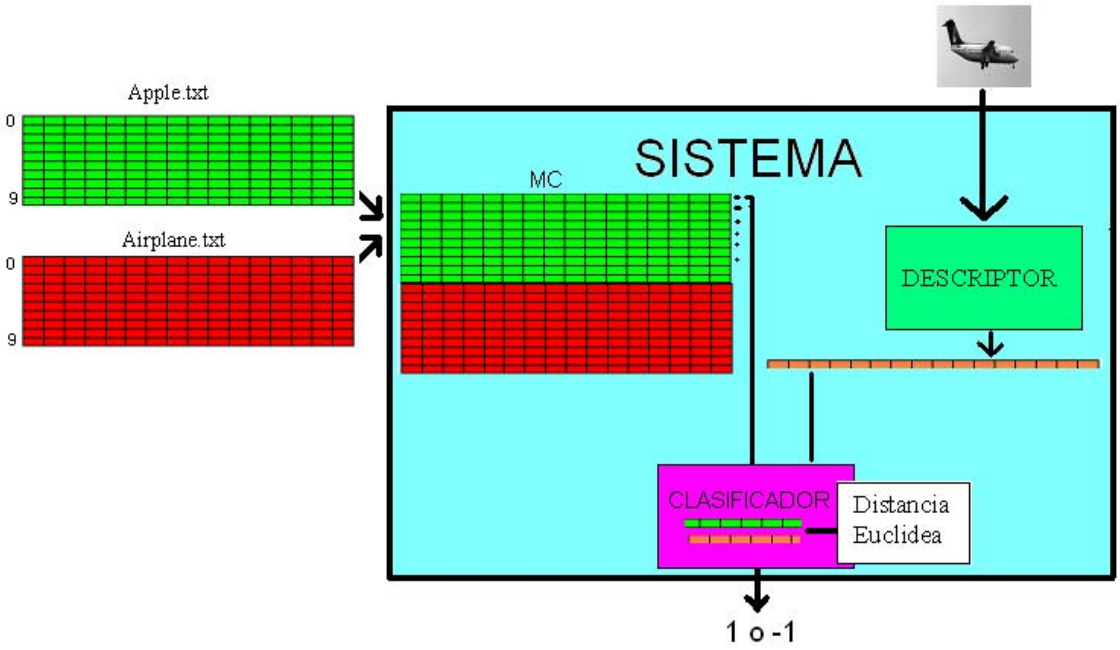


Fig. 33: Proceso de comparación desarrollado en el clasificador binario

Como se muestra en la fig. 33 de la comparación entre el vector de características asociado a la imagen de entrada y cada una de las filas de la matriz MD obtenemos un +1 o -1. Si el resultado es +1 indica que alguna de las filas de la la clase 1 contiene el vector que ha obtenido la mínima dístancia. El método utilizado para calcular ésta distancia es la distancia Euclídea comentada en puntos anteriores de la memoria. Si el resultado es -1 sería la clase 2 la que contendría dicho vector. Se realizarán tantas comparaciones de clases como combinaciones de parejas de éstas existan, y se realizarán tantas comparaciones entre vectores como filas tengan estas clases.

Nº de comparaciones = 6 parejas * 20 comparaciones = 120 comparaciones

Los resultados de estas comparaciones se guardan en un vector de longitud 6. El valor de cada elemento del vector corresponde a 1 o -1 y dependerá del resultado de la comparación. El último paso del clasificador consiste en comparar el vector obtenido anteriormente con cada una de las filas de la matriz codificada.

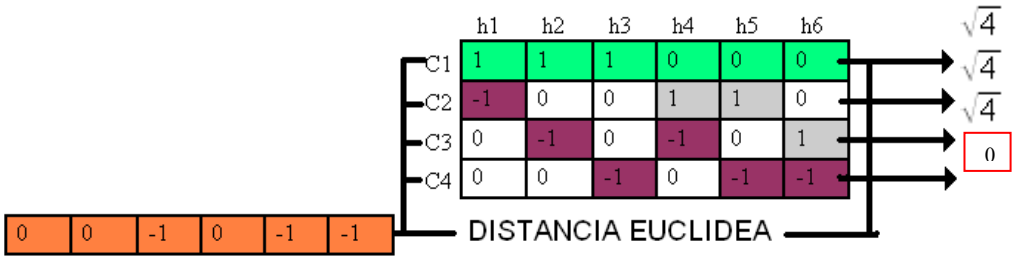


Fig. 34: Fin proceso reconocimiento (clasificador binario).

De los resultados obtenidos escogemos el que tenga la distancia mínima, tal y como muestra la fig.34. Una vez más hemos utilizado el algoritmo de distancia Euclídea para realizar el cálculo de distancias.

6.1 Características Técnicas

El sistema ha sido desarrollado bajo el entorno Microsoft Visual Studio 6.0 y utilizando C++ como lenguaje de programación, además de la incorporación de las librerías de procesamiento gráfico OpenCV.

6.2 Características de Micorsoft Visual C++

Microsoft Visual C++ [24] proporciona un entorno de desarrollo eficaz y flexible para crear aplicaciones basadas en Microsoft Windows y en Microsoft .NET. Se puede utilizar como un sistema de desarrollo integrado o como un conjunto de herramientas individuales. Visual C++ se compone de estos componentes:

El compilador tiene nuevas características que ayudan a los desarrolladores que trabajan con plataformas de equipo virtual como Common Language Runtime (CLR)[25]. Las bibliotecas de Visual C++ 2005. Incluyen las siguientes bibliotecas: Active Template Library (ATL) [26] estándar del sector, Microsoft Foundation Class (MFC) [27], la Biblioteca estándar de C++ y la Biblioteca en tiempo de ejecución de C (CRT) [28], que se ha ampliado para proporcionar alternativas de seguridad mejorada a funciones que sufrían problemas de seguridad conocidos. Una nueva biblioteca, la Biblioteca de compatibilidad de C++, está diseñada para simplificar programas destinados al CLR.

Aunque las herramientas y bibliotecas del compilador de C++ se pueden utilizar desde la línea de comandos, el entorno de desarrollo proporciona una eficaz ayuda para la administración y configuración de proyectos (incluida una mejor compatibilidad con grandes proyectos), edición y exploración del código fuente y herramientas de depuración. Este entorno también admite IntelliSense [29], que realiza sugerencias contextuales y bien fundamentadas cuando se crea el código.

Además de las aplicaciones de interfaz gráfica de usuario convencionales, Visual C++ permite a los desarrolladores generar aplicaciones Web, aplicaciones smart-client basadas en Windows y soluciones para dispositivos móviles thin-client [30] y smart-client.[31] C++ es el lenguaje de nivel de sistemas más popular del mundo, y Visual C++ ofrece a los desarrolladores una herramienta universal con la que generar software.

6.3 Características de Open CV

OpenCV es una librería que contiene muchas estructuras de datos complejos y funciones de alto nivel utilizadas para procesos de flujo óptico, reconocimiento de patrones, procesamiento 2D-3D en tiempo real, calibración de cámaras y muchas otras más. Además permiten trabajar con múltiples formatos de video como AVI y contiene una interficie gráfica de usuario llamada Highgui que nos permite un fácil y rápido forma de interactuar y visualizar las imágenes.

6.4 Estructura de las librerías OpenCV

Los archivos de cabeceras donde están las definiciones de las funciones a utilizar se dividen en los siguientes:

- **CV.H:** En este archivo están definidas las funciones de procesamiento general de imágenes, filtros, conversión de colores, análisis de movimiento, operadores morfológicos, análisis, reconocimiento de patrones (detección de objetos), calibración de cámaras y reconstrucción 3D.

- **CXCORE.H:** Estructuras básicas, operadores aritméticos/lógicos (copia, transformación), estructuras dinámicas (asignación, grafos, árboles), funciones de dibujo, errores de enlace y funciones de sistema.

- **CVAUX.H:** Correspondencia estéreo, descriptores de textura, procesamiento 2D-3D, segmentación de fondo, morfología, etc.

- **HIGHGUI.H:** Corresponde a la interficie de usuario

Como se ha comentado en el punto anterior la aplicación se ha desarrollado bajo el entorno Microsoft Visual Studio 6.0, utilizando C++ como lenguaje de programación, además se ha hecho uso de funciones propias de las librerías de procesamiento gráfico OpenCV. Desde un punto de vista funcional destacaríamos 4 clases de la aplicación:

- **Clase Entrenar:** Funcionalidad Caso de uso entrenar. Comparte junto con la clase Descripción funcionalidades propias del Descriptor.
- **Clase Descripción:** Funcionalidades propias del descriptor. Destacamos la utilización de funciones propias de las librerías OpenCV.
- **Clase RecoImagen:** Combinación de funcionalidades tanto del descriptor como del clasificador. Corresponde a la lógica del caso de uso de reconocer imagen.
- **Clase Porcentaje:** Al igual que la clase RecoImagen combina la funcionalidad del descriptor y del clasificador, la diferencia entre éstas recae en su objetivo, ya que la clase Porcentaje se ha diseñado únicamente para calcular en tiempo real la probabilidad de acierto de la aplicación en el proceso de reconocimiento de objetos.

La descripción de las principales funciones de estas clases se encuentra en el apéndice D.

6.5 Casos de uso

6.5.1 Diagrama de casos de uso del sistema

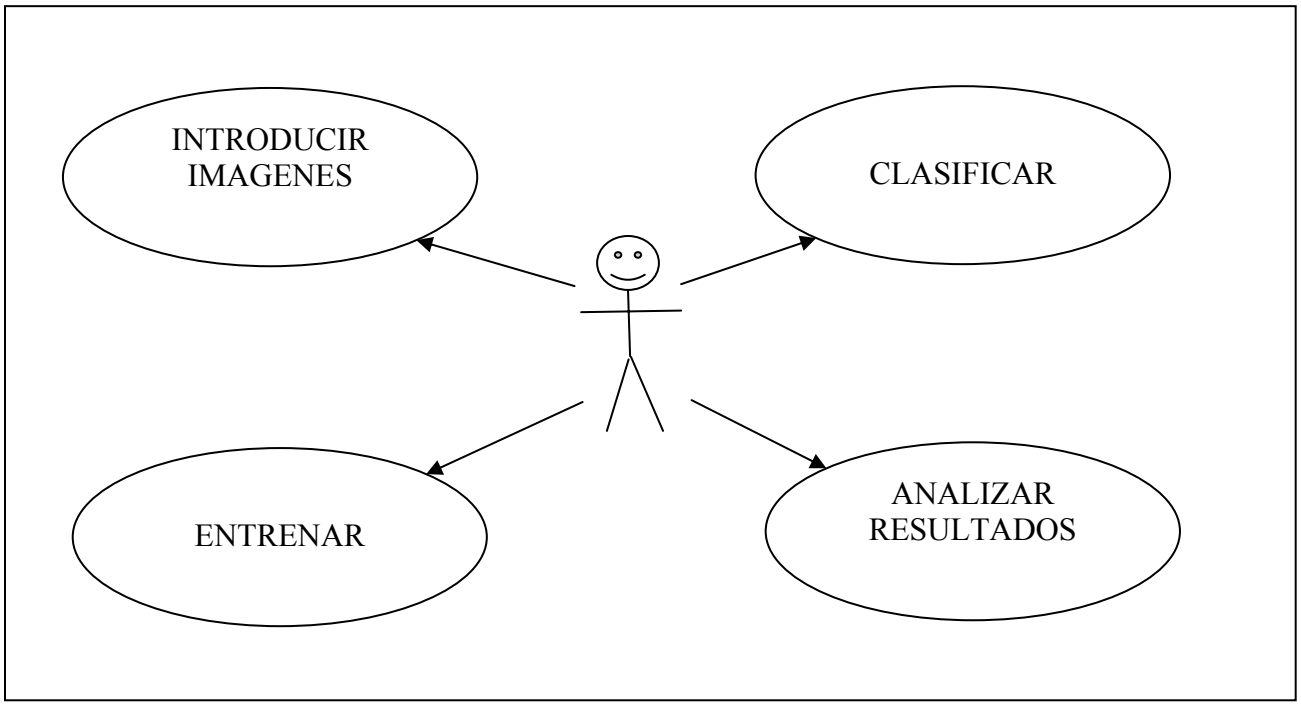


Fig. 35: Resultado en porcentaje de aciertos en cada fase

El diagrama de la fig. 35 muestra los diferentes casos de uso de la aplicación:

- **Introducir imágenes:** El sistema permite al usuario introducir imágenes utilizando la funcionalidad de la pantalla de de entrenamiento. El usuario escoge la ruta donde se encuentran las imágenes y selecciona las que desea utilizar.

- **Entrenar:** El sistema procesa las imágenes introducidas por el usuario comentado anteriormente y guarda los vectores de características para posteriores procesos.

- **Clasificar:** El sistema clasifica los vectores de características en distintas clases.

- **Analizar resultados:** El usuario obtiene los porcentajes de acierto en el proceso de reconocimiento de objetos.

6.5.2 Diagrama del caso de uso Entrenar

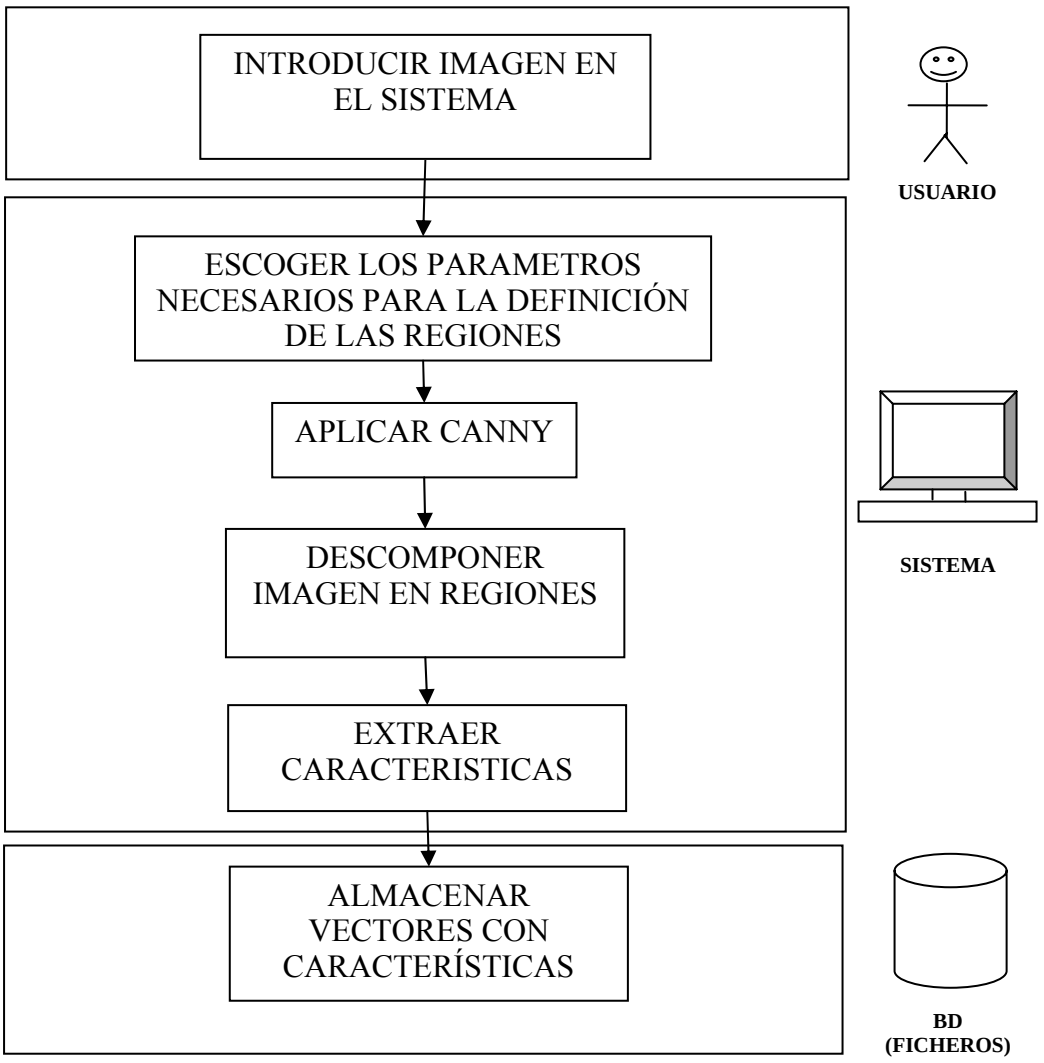


Fig. 36: Diagrama de caso de uso entrenar

El diagrama de la fig. 36 muestra el proceso completo que se produce en el sistema en la realización del caso de uso entrenar, en dicho proceso el que participan el usuario, el descriptor que realiza el proceso de extracción de características de las imágenes de entrada, y los ficheros que se utilizan para almacenar los vectores de características.

6.5.3 Diagrama de de flujo del caso de uso Reconocer

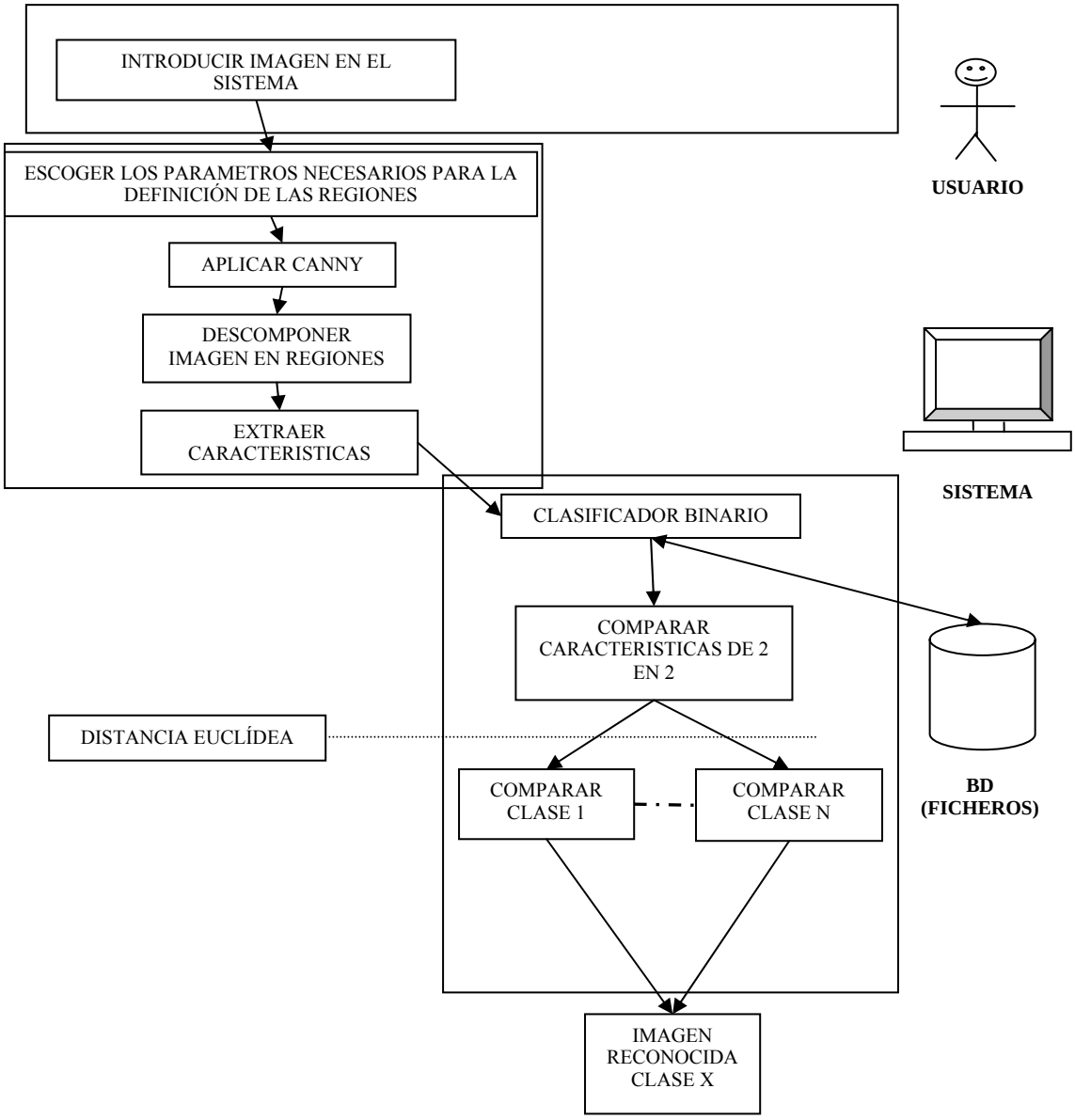
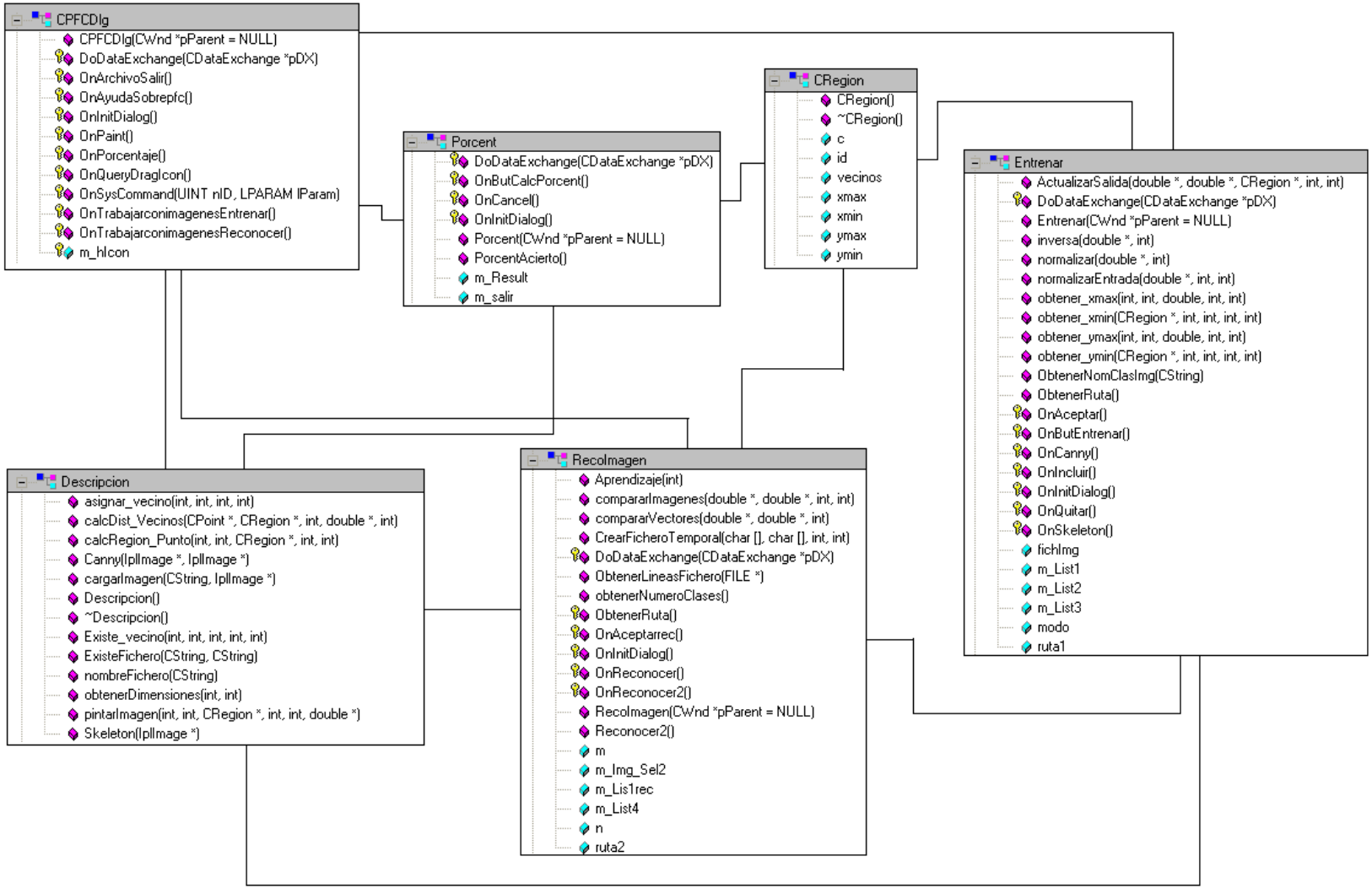


Fig. 37: Resultado en porcentaje de aciertos en cada fase

El diagrama de la fig. 37 muestra el proceso completo que se produce en el sistema en la realización del caso de uso reconocer, en dicho proceso participa el usuario, el descriptor que realiza el proceso de extracción de características de las imágenes de entrada, los ficheros que se utilizan para almacenar los vectores de características, y el clasificador que se encarga de realizar las comparaciones binarias entre vectores para obtener el que más se aproxime al objeto de entrada, y por lo tanto realizar el reconocimiento.

6.5.4. Diagrama de clases



7. INTERFAZ DE LA APLICACIÓN

La aplicación dispone de una interfaz gráfica que permite interactuar al usuario con el sistema. El hecho de que sea una interfaz gráfica facilita la comprensión de la aplicación y ameniza el proceso de inserción de los datos requeridos por el sistema en cada paso dentro del proceso global de reconocimiento de imágenes. La interfaz esta diseñada e implementada utilizando las ventajas del entorno Visual de Microsoft.

La interfaz esta formada por 4 formularios (pantallas), la pantalla principal que es la primera pantalla que se muestra al usuario y las diferentes pantallas dependiendo de las acciones que desee realizar el usuario que se comentan a continuación.

7.1 Pantalla principal

Es la pantalla principal de la aplicación. Es la primera pantalla que le aparece al usuario cuando arrancamos la aplicación.

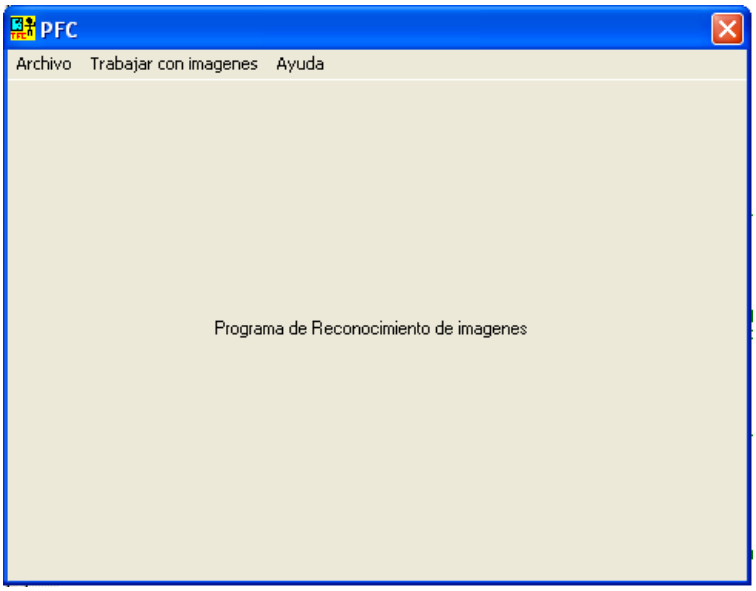


Fig. 38: Pantalla principal

Cosas a destacar de esta pantalla:

Icono personalizado: Este icono es el emblema de la aplicación y representa una persona mirándose al espejo.

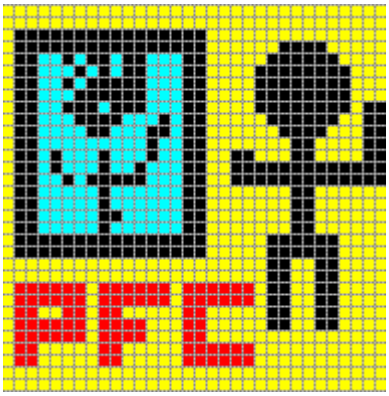


Fig. 39: Icono de la aplicación

Menús: La aplicación ofrece a disposición del usuario una barra de menús situada en la parte superior de la pantalla principal. Permite a usuario desplazarse por las diferentes pantallas de forma rápida e intuitiva.

Menú Archivo: Permite al usuario salir de la aplicación.

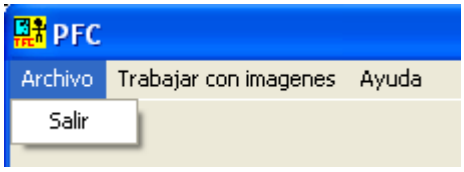


Fig. 40: Menú Archivo

Menú Trabajar con imágenes: Funcionalidades de la aplicación relacionados con el reconocimiento de imágenes.

Fig. 41: Menú Trabajar con imágenes

Entrenar clasificador: Pulsando esta opción del menú accedemos a la pantalla de entrenamiento del clasificador.

Reconocer imágenes: Seleccionando esta opción el usuario accederá a la pantalla de reconocimiento de imágenes.

Porcentaje de acierto: Permite al usuario acceder a la pantalla de Porcentajes.

Menú Ayuda: Accedemos a la pantalla de de créditos de la aplicación.

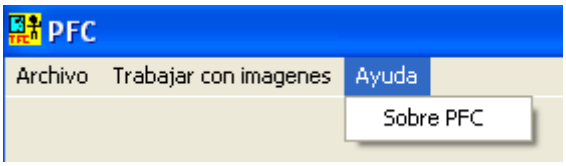


Fig. 42: Menú Ayuda

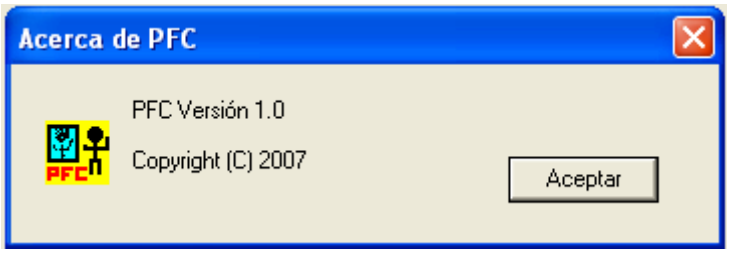


Fig. 43: Pantalla de créditos

Esta pantalla muestra los créditos de la aplicación. Pulsando al botón aceptar retornamos a la pantalla principal.

7.2 Pantalla Entrenar clasificador

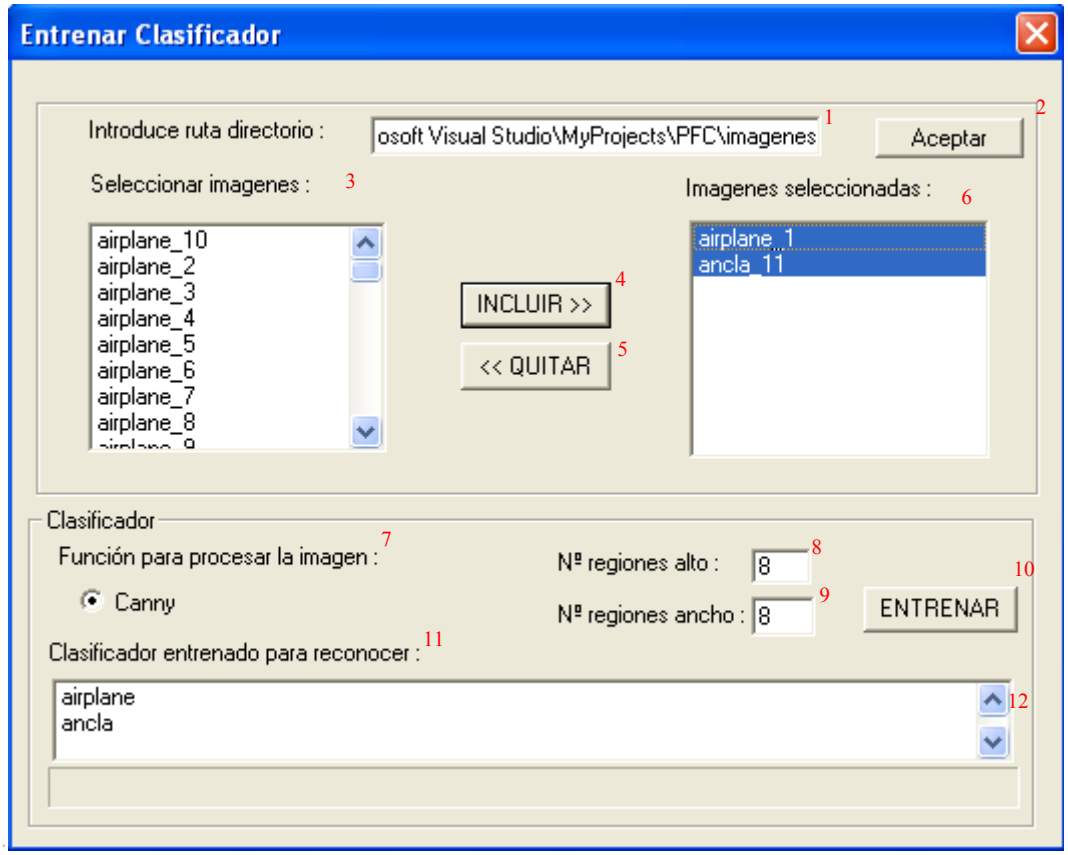


Fig. 44: Pantalla de Entrenar Clasificador

Comentaremos cada uno de los elementos de esta pantalla utilizando la numeración que contiene la imagen anterior

1. Ruta de localización de imágenes: El usuario ha de indicar al sistema la ruta donde están localizadas las imágenes

2. Botón Aceptar: Una vez introducida la ruta el usuario presiona el botón Aceptar el cual tiene asociada la función ObtenerRuta(), que valida si la ruta es correcta y busca las imágenes que se encuentran bajo el directorio especificado.

3. Lista de Selección de imágenes 1: Muestra las imágenes que puede escoger el usuario.

4. Botón incluir: Desplaza las imágenes seleccionadas de la lista de selección 1 a la lista de selección 2.

5. Botón excluir: Desplaza las imágenes seleccionadas de la lista de selección 2 a la lista de selección 1.

6. Lista de Selección 2: Muestra las imágenes seleccionadas por el usuario para ser procesadas por el sistema.

7. Función Canny: Muestra al usuario el nombre del algoritmo que se utilizará en el proceso de extracción de características, tarea realizada por el descriptor.
8. Nº Regiones Alto: Permite al usuario introducir el nº de divisiones verticales en que se dividirá la imagen.

9. Nº Regiones Ancho: Permite al usuario introducir el nº de divisiones horizontales en que se dividirá la imagen.

10. Botón Entrenar: Botón asociado a la función entrenar que inicia el proceso de procesamiento de imágenes por parte del descriptor.

11. Lista de imágenes descriptor: Una vez se ha procesado una imagen esta lista muestra el nombre de la clase a la que pertenece la imagen.

12. Barra de desplazamiento: Indica el porcentaje de proceso realizado.

7.3 Pantalla de Reconocimiento de Imágenes

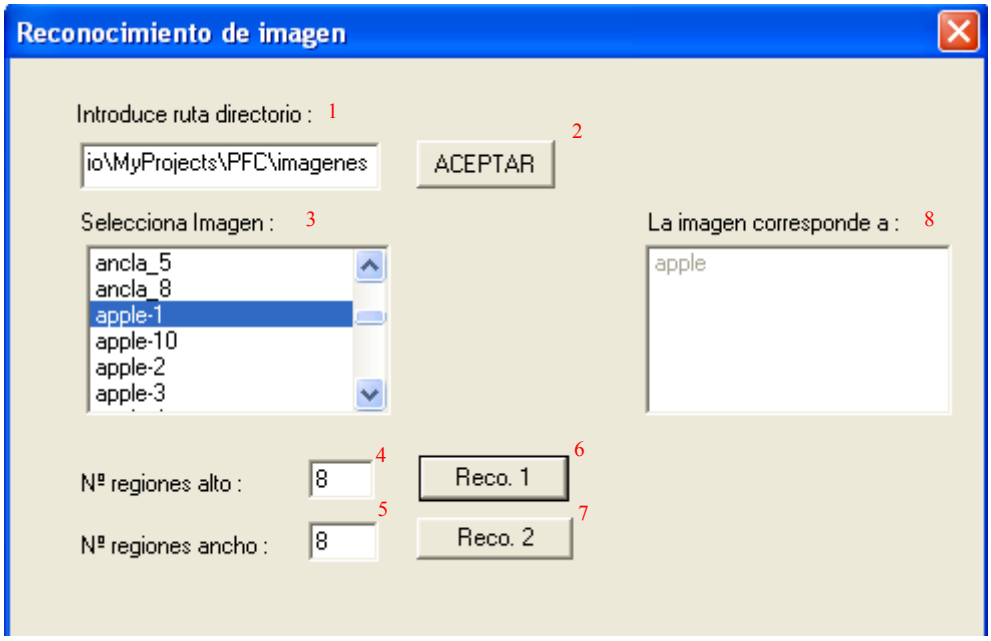


Fig. 45: Pantalla de Reconocimiento de imágenes

1. Ruta de localización de imágenes: El usuario ha de indicar al sistema la ruta donde están localizadas las imágenes

2. Botón Aceptar: Una vez introducida la ruta el usuario presiona el botón Aceptar el cual tiene asociada la función ObtenerRuta(), que valida si la ruta es correcta y busca las imágenes que se encuentran bajo el directorio especificado.

3. Lista de Selección de imágenes: Muestra las imágenes que puede escoger el usuario.

4. Nº Regiones Alto: Permite al usuario introducir el nº de divisiones verticales en que se dividirá la imagen.

5. Nº Regiones Ancho: Permite al usuario introducir el nº de divisiones horizontales en que se dividirá la imagen.

6. Botón Reconocer 1: Botón asociado a la función Reconocer1 que inicia el proceso de procesamiento de imágenes por parte del clasificador que utiliza únicamente distancia Euclídea.

7. Botón Reconocer 2: Botón asociado a la función Reconocer1 que inicia el proceso de procesamiento de imágenes por parte del clasificador binario BSM.

8. Lista de imágenes Reconocidas: Una vez se ha procesado una imagen esta lista muestra el nombre de la clase reconocida.

7.4 Pantalla de Porcentaje de Acierto

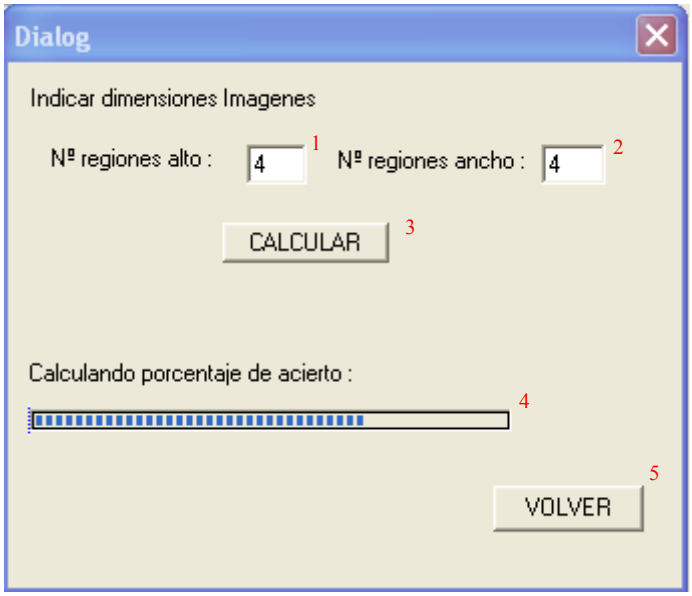


Fig. 46: Pantalla de Porcentaje de Acierto

1. Nº Regiones Alto: Permite al usuario introducir el nº de divisiones verticales en que se dividirá la imagen.

2. Nº Regiones Ancho: Permite al usuario introducir el nº de divisiones horizontales en que se dividirá la imagen.

3. Botón Calcular: Botón asociado a la función PorcentAcierto(), que realiza el cálculo de las probabilidades que tiene el sistema para reconocer una imagen de forma correcta. El resultado de la probabilidad de acierto aparece en una ventana de aviso emergente.

Fig. 47: Resultado de porcentaje de acierto

4. Barra de desplazamiento: Indica el porcentaje de proceso realizado.

5. Botón Volver: Pulsando el botón el usuario puede volver a la pantalla principal de la aplicación.

8. RESULTADOS

Finalizada la fase de diseño e implementación de nuestro sistema reconocedor, ha llegado el momento de comprobar su eficacia. Con éste objetivo realizamos un experimento utilizando imágenes en color. Estas imágenes son software libre, forman parte del paquete de imágenes conocido como Caltech. La fig. 48 muestra una parte de las imágenes pertenecientes a la Caltech. Éste paquete de imágenes es comúnmente utilizado en muchas investigaciones de aplicaciones relacionadas con el reconocimiento de imágenes.

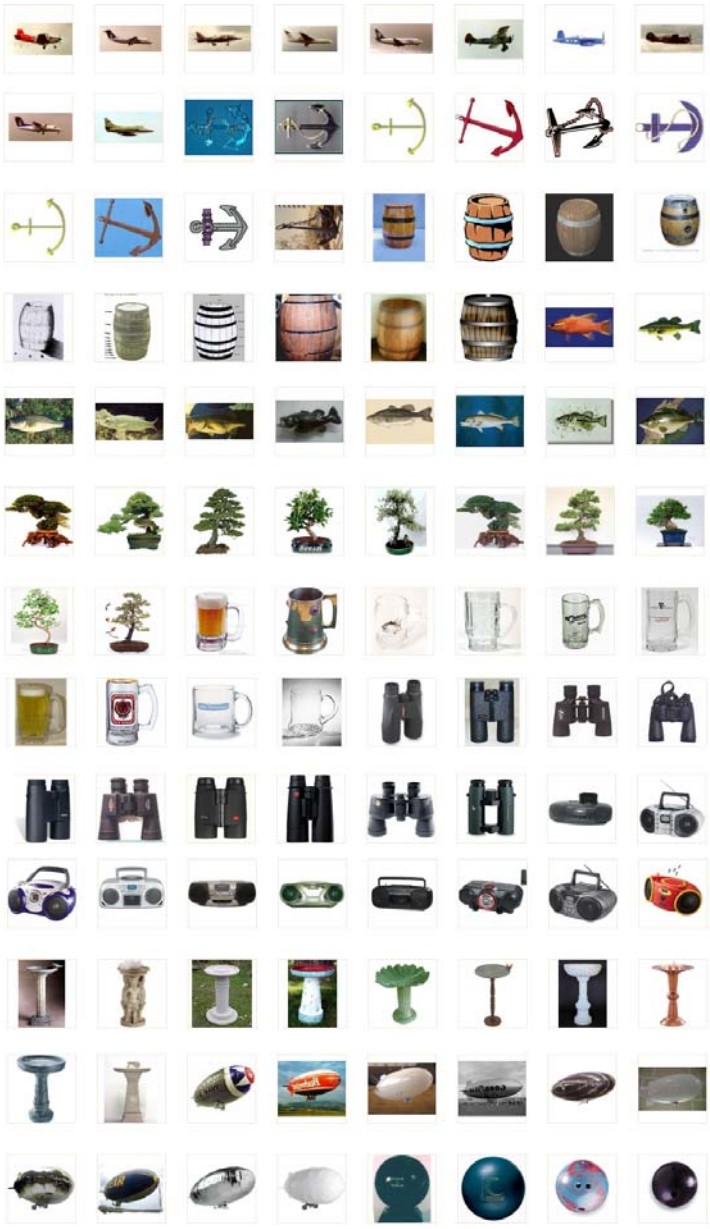


Fig. 48: Ejemplos de imágenes extraídas del paquete Caltech

El experimento consiste en entrenar a nuestro sistema con diferente número de clases e imágenes. Partiremos de 3 clases con 10 imágenes cada una e iremos aumentando el número de clases hasta 20. Cada entrenamiento lo llamaremos fase de entrenamiento, por tanto contaremos con 18 fases de entrenamiento. En cada fase de entrenamiento se comprobará el correcto funcionamiento del sistema y extraeremos el resultado en tanto por ciento del número de aciertos.

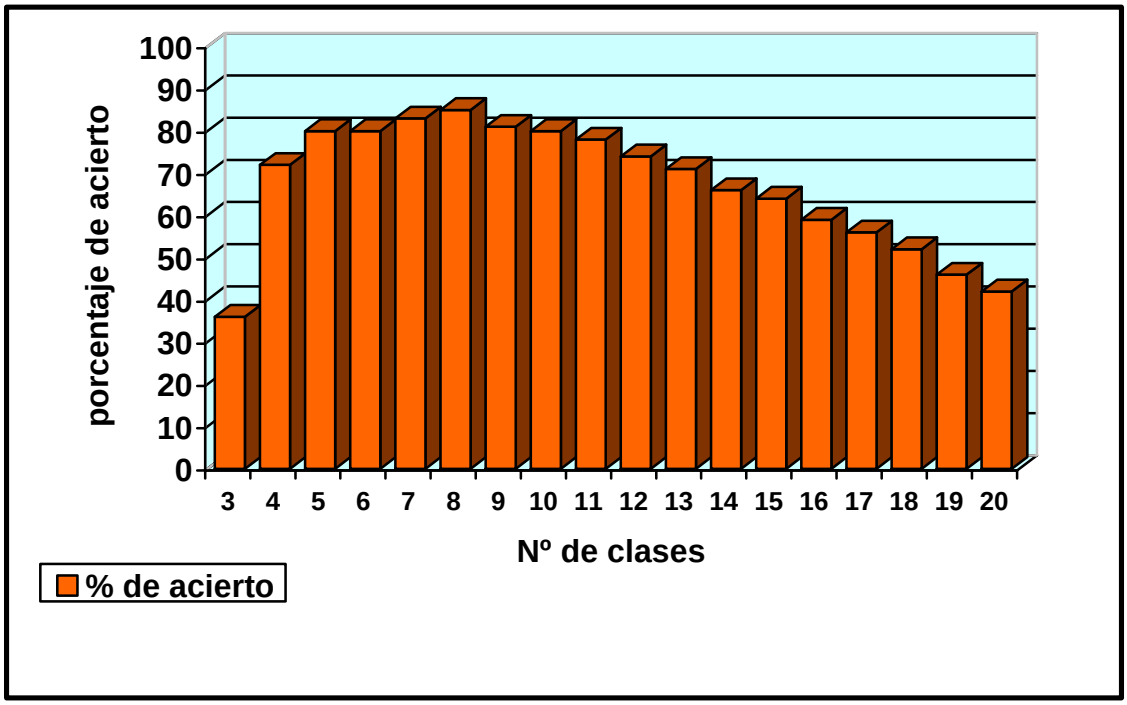


Fig. 49: Resultado en porcentaje de aciertos en cada fase

Como podemos ver en la fig. 49 nuestro sistema aprende a medida que se va entrenando. Así como en la primera fase de entrenamiento cuando únicamente cuenta con 3 clases el porcentaje de acierto al reconocer una imagen es del 36 % comprobamos que ya en la segunda fase, únicamente con una clase más ya se ha mejorado su rendimiento y la probabilidad de acertar al reconocer un objeto es del 72 %.

Siguiendo con la ilustración comprobamos que la probabilidad máxima conseguida con nuestro sistema es del 85 %, resultado más que bueno considerando que el ser humano alcanza hasta un máximo del 95%.

Es también importante comentar que nuestro sistema mejora su rendimiento hasta un nivel límite y a partir de este el rendimiento no solo no se mantiene sino que baja de forma notable respecto a la anterior fase de entrenamiento. Éste acontecimiento es provocado por el aumento considerable del número de objetos a reconocer, ya que a partir de un número determinado en lugar de aprender el sistema se confunde y mezcla los diferentes tipos de objetos, fenómeno común a cualquier tipo de sistema reconocedor.

Para reafirmar el correcto funcionamiento de los algoritmos utilizados en nuestro sistema, se han realizado un conjunto de experimentos con diferentes tipos de descriptotes tales como Zernique [32], Zoning [33], curvatura CSS [34] y SIFT [35]. El

software utilizado para realizar éste experimento es software libre y al igual como las librerías de imágenes Caltech son utilizados en multitud de investigaciones relacionadas con el reconocimiento de imágenes.

Los detalles de los descriptores utilizados son los siguientes:
El tamaño óptimo de rejilla utilizada en descriptores BSM se consigue aplicando validación cruzada sobre un conjunto de entrenamiento utilizando un 10% de los ejemplos para validar los diferentes tamaños de n {8,12,16,20,24,28,32}. Para una justa comparación entre diferentes descriptores se ha de utilizar la misma longitud, si nos centramos en Zernique utilizaremos 7 momentos para obtener el descriptor, y una longitud de 200 como sigma e iremos incrementando de uno en uno aplicado a la curvatura del descriptor CSS.

Para testear y comparar el rendimiento de los diferentes descriptores, realizamos un experimento que consiste en la clasificación de 70 clases de imágenes MPEG. Cada clase contiene 20 elementos, por lo que el total de elementos total es de 1400.

Se ha escogido este conjunto de imágenes MPEG por que nos proporcionan gran variabilidad de tamaños, rotación, rigidez y deformaciones elásticas entre elementos de la misma clase y al mismo tiempo nos ofrecen una baja variabilidad entre clases. Clasificamos el conjunto de datos siguiendo este orden, comparamos descriptores de tipo BSM, Zoning, SIFT, CSS, y Zernique.

Para elegir el tamaño de la rejilla se han realizado un conjunto de pruebas. El resultado de las cuales ha determinado 8x8 como tamaño óptimo de rejilla. Además de escoger la rejilla comentar que el clasificador base utilizado en el ECOCO ha sido Gentle Adaboost con 50 iteraciones.

El rendimiento y el intervalo de confianza obtenido por 3NN y Gentle Adaboost para todos los descriptores se muestra en la tabla 53.

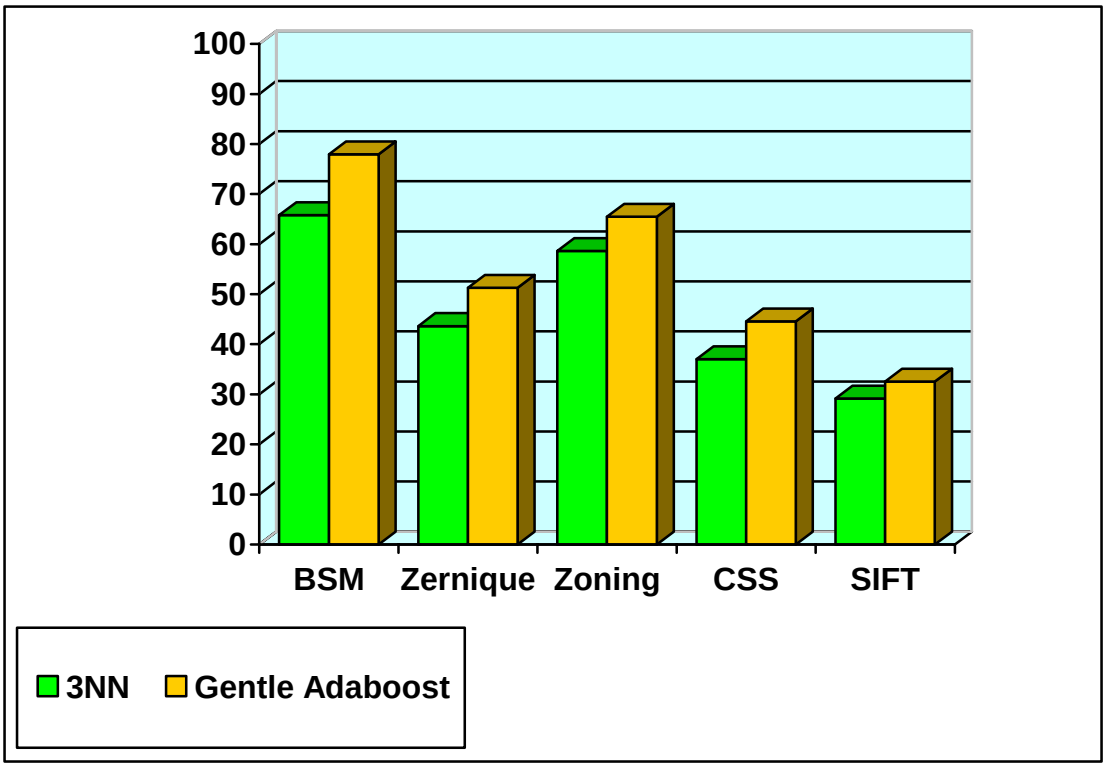


Fig. 50: Histograma resultados experimento MPEG

En la fig. 50 podemos ver los resultados obtenidos utilizando los diferentes descriptores. De ésta forma comprobamos como el clasificador que utiliza un esquema ECOC basado en Gentle Adaboost obtiene los mejores resultados llegando casi a una probabilidad de acierto del 80%, seguido de cerca por Zoning y Zérnique. Es importante destacar que existe una diferencia considerable entre casos que utilizan el mismo clasificador combinado con diferentes descriptores. Ésto es debido a que existe un gran número de clases y por lo tanto hay una alta variabilidad de forma que puede ocasionar confusiones al sistema.

9. CRONOGRAMA Y COSTES

9.1 Cronograma

TAREA \ DURACIÓN	Oct.	Nov.	Dic.	En.	Feb.	Mar.	Abr.	May.	Jun.	Jul.
Recopilación de información										
Memoria										
Diseño de la interficie										
Implementación de la interficie										
Diseño descriptor										
Implementación Descriptor										
Diseño Clasificador										
Implementación Clasificador										
Entrega del Proyecto										
Defensa del Proyecto										

Fig. 51 : Cronograma del desarrollo del proyecto

La tabla de la fig. 51 refleja el tiempo dedicado a cada tarea a lo largo de todo el proyecto. Aquellas tareas que se solapan en el tiempo, como la realización de la memoria y el diseño de la interficie, indican que durante ese periodo se ha dedicado parte del tiempo a cada una de ellas.

9.2 Gastos de personal

En el desarrollo del proyecto ha trabajado 1 persona a media jornada.

- **Trabajadores:**

Cantidad de trabajadores	Categoría Profesional	S.B.A (€)
1	Ingeniero Técnico Informático	18.000

Horas de actividad al año por trabajador:

3 h*20días*10meses /año = 600 h/año

$$Gastos\ \ personal\ (€) = RH\ (horas \cdot persona) \cdot \frac{SBA + SS\left(\frac{€}{año \cdot persona}\right)}{\left(\frac{número\ de\ horas\ trabajadas}{año}\right)}$$

SS = Cuota de empresa a la seguridad social 0,32 € · SBA

RH (horas · persona)	SS (€)	Gasto personal por categoría profesional (€)
600	5.760	23.760
Gasto total Personal: 23.760€		

EQUIPO ELÉCTRICO

Ordenador portátil .....1299,37 €
Licencia Microsoft Visual Studio 6.0 ..... 200 €

RESUMEN PRESUPUESTO

Equipos eléctrico…….1499,37 €
Gasto de personal.....23.760 €

Gasto total…….35.25937 €

10. CONCLUSIONES

El reconocimiento de imágenes es una disciplina que siempre ha causado gran interés entre la sociedad científica, siendo cada vez más precisa y compleja, aun así hay que destacar la distancia que separa hoy en día la visión artificial de la biológica. La principal causa que imposibilita una mayor aproximación entre la visión artificial de la biológica son las restricciones a las que se ha de someter el modelo base a aplicar en el sistema a reconocer imágenes.

También hay que tener en cuenta que los humanos tenemos un 4% de error al reconocer objetos o personas; muchas veces hemos confundido a personas a pesar que existen rasgos muy diferentes. Por lo cual nos es posible aún lograr un 96% de reconocimiento por medio del ordenador, y mucho menos reconocer al 100% un objeto.

Existen muchas técnicas para desarrollar sistemas capaces de reconocer imágenes, como en cualquier campo, el problema reside en la elección de la técnica adecuada según la funcionalidad final que se desea aportar a la aplicación.

En el desarrollo de la aplicación propia de este proyecto se han tenido en cuenta los factores comentados anteriormente y es por ello que el primer objetivo fue marcar que tipo de entes tendría que reconocer nuestro sistema. Una vez decidido que el ente a reconocer serían imágenes estáticas en blanco/negro y color y que no se requería ningún proceso para su obtención, se factores y se ha se

Implementación de algoritmos para el entrenamiento de nuestro programa con el fin de aumentar la probabilidad de acierto en el proceso de reconocimiento.

Dentro de este marco encapsularíamos el proceso de recepción de ficheros de imágenes, la extracción de las principales características implementando un descriptor basado en la teoría del BSM (Blurred Shape Model), y el uso de clasificadores binarios tales como K-NN, con distancia Euclídea ó Adaboost, combinados dentro del marco de los ECOC. También se ha implementado una interfaz de usuario basada en el entorno Visual Studio C++, permitiendo así la interacción del usuario con el sistema.

El sistema se ha validado en bases de datos públicas en color y en escala de grises, realizando una comparativa entre diferentes descriptores y clasificadores existentes, mostrando un alto rendimiento del sistema desarrollado en este proyecto.

Dada la duración del proyecto se desestimó la opción de almacenar dicha información en una base de datos de apoyo ya que hubiera alargado el proceso final de implementación y no hubiera aportado una ganancia considerable en su complejidad ni en sus prestaciones.

11. POSIBLES AMPLIACIONES DEL PROYECTO

Como posible mejora de la aplicación podríamos dotar al sistema de una BD para almacenar los resultados de los algoritmos. Nos permitiría aportar escalabilidad, consistencia y mejor control de datos. Así como utilizar consultas SQL para interactuar con los datos y obtener aquellos que nos interesen en cada momento.
Otra posible mejora aportar un mecanismo de obtención de imágenes del medio ya que nuestro sistema obtiene las imágenes de una carpeta del ordenador. Este mecanismo podría ser una cámara web.

12. AGRADECIMIENTOS

Quiero agradecer a todas aquellas personas que me han ayudado y dado soporte a la hora de realizar el presente proyecto, especialmente a aquellas a las que se menciona a continuación:

Mi principal agradecimiento va dirigido al director de este proyecto el Sr. Sergio Escalera por su continuo seguimiento y colaboración a lo largo de todo el proyecto.

También quiero agradecer a toda la gente que colabora en el foro dedicado a las librerias OpenCV de yahoo ya que sus aportaciones en el foro me han ayudado mucho en la implementación de la aplicación.

Finalmente, quiero agradecer a mi novia y a mi familia por la paciencia que han tenido conmigo en los buenos y malos momentos que he pasado en el transcurso de este proyecto.

13. APENDICE A:

DEFINICIONES Y CAMPOS DE APLICACIÓN

13.1 Definiciones

Inteligencia artificial: Campo que ayuda a tomar decisiones mientras se resuelven ciertos problemas concretos, los sistemas expertos que buscan soluciones a través del conocimiento previo del contexto en que se aplica y de ciertas reglas o relaciones, las redes bayesianas que proponen soluciones mediante inferencia estadística y la inteligencia artificial basada en comportamientos que tiene autonomía y pueden auto-regularse y controlarse para mejorar.

Procesamiento de imágenes: Campo que se basa en la transformación de imagenes para obtener otra de más calidad o mejor acondicionada para la posterior extracción de información.

Gráficos por computador: Campo que aborda el problema de plasmar en formato bidimiensional el mundo real.

Reconocimiento de patrones: Campo que clasifica objetos en clases representadas por prototipos o patrones.

13.2 Campos de Aplicación

La industria relacionada con la visión por computador esta actualmente en fase de rápido crecimiento, debido en parte a la incursión en nuevas áreas de aplicación como también a la mayor importancia que esta adquiriendo en las que ya está consolidada.

CAMPO DE APLICACION	EJEMPLO
INDUSTRIA	
Control de calidad e inspección	Inspección láminas de aluminio, nivel de llenado de botellas, etc.
Identificación de piezas	Clasificación automática de piezas
Ensamblaje	Montaje de chips en placas de circuito impreso
Medición de objetos	Medición del espesor granular en lingotes de hierro
Guiado de robots	Control de robot en soldadura al arco.
MEDICINA	
Pruebas de laboratorio automáticas	Recuento glóbulos en sangre, detección de células anormales, etc.
Diagnóstico por computador	Tomografía computacional, resonancias magnéticas, etc.
TELE-MEDICIÓN	
Exploración geofísica	Interpretación de fotografías aéreas
Meteorología	Pronóstico meteorológico de fotos de satélites
DEFENSA MILITAR	
Vigilancia por satélite	Detección de movimiento de tropas o misiles
Guiado de larga distancia	Guiado de misiles crucero
Armas/municiones inteligentes	Misiles aire-aire, bombas guiadas por visión
OTROS CAMPOS	
Campo científico	Análisis de dinámica de flujos turbulentos
Sistemas de seguridad	Detección de movimientos, identificación intrusos, etc.
Entornos peligrosos	Inspección de conductos en centrales nucleares, desactivación de explosivos

Fig. 52: Aplicaciones del reconocimiento de imágenes

En la tabla de la fig. 52, los principales campos de aplicación de la visión por computador, incluyendo ejemplos ilustrando cada uno de ellos.

Destacamos también algunos ejemplos de empresas que utilizan dotan a sus sistemas de aplicaciones capaces de reconocer objetos.

13.2.1 Fujitsu

Diseña robots que tienen la habilidad de reconocer imágenes en tres dimensiones.
Aunque todavía no se han facilitado demasiados detalles, Fujitsu asegura que este desarrollo trabajará en tiempo real y será cinco veces superior a los actuales desarrollos en la materia, y al ser tan pequeño el chip puede ser embebido en cualquier tipo de dispositivos, desde robots industriales, domésticos o inclusive para la domótica.
Fujitsu ha prometido que preparará un prototipo funcional para demostrar el modo de trabajo de un robot con este chip.

13.2.2 Riya

Buscador de que utiliza algoritmos de reconocimiento de imágenes.
Buscador de nueva generación en fase de pruebas que permite la búsqueda de contenido multimedia, utilizando el propio cuerpo de los objetos multimedia a diferencia de lo que se había realizado hasta ahora que consistía en buscar en la metainformación suministrada conjuntamente con el contenido multimedia.
De esta forma Riya nos permite buscar fotos o videos en los que aparezca Zapatero como protagonista a diferencia de los anteriores buscadores que limitaban su criterio de búsqueda a archivos en los que aparecía el nombre de Zapatero en las etiquetas o páginas que citen a este personaje.
El reto tecnológico que se ha propuesto Riya es doblemente complejo ya no solo utiliza algoritmos de búsqueda de rostros y de reconocimiento de imágenes sino que además a ello hay que sumarle las tareas propias de un buscador de indexar las imágenes en la red.
Otros ejemplos de buscadores con reconocimiento de imágenes:

13.2.3 VIMA Technologies

Utiliza múltiples parámetros como el color, la textura, la forma,… para clasificar las imágenes. Luego procesa dichos datos en las búsquedas. Va mostrando imágenes y tienes que especificarle si se parece o no a lo que buscas. De esa forma va afinando el resultado. Es capaz de filtrar imágenes pornográficas.

13.2.4 Polar Rose

Dispone de plugin para Firefox y próximamente van a hacer pública una API para usar gratuitamente. Reconoce rostros de una forma bastante efectiva, con un modelo en 3D. Con una serie de filtros, reconoce las partes más prominentes e importantes de la cara y luego extrapola mediante algoritmos un modelo en tres dimensiones de la misma. Esta es la información que se almacena en su base de datos.

13.2.5 Daem Interactive

Han desarrollado software que permite reconocer imágenes con el móvil.
Un ejemplo que ilustra las posibilidades que nos ofrece este software sería el caso de una persona que puede realizar una foto a un anuncio, el software instalado en el móvil analiza la foto, y envía los datos a un servidor que después devuelve contenido al móvil para que el usuario pueda consultarlo. Puede ofrecerle archivos multimedia, participar en promociones, concursos, etc.
Aunque actualmente están ofreciendo esta tecnología dentro de estrategias de marketing asociadas a anuncios, lo interesante es que es un sistema de reconocimiento de formas.
En Deam interactive te plantean cosas como el poder hacer una foto con tu móvil a una caja de galletas "María", por ejemplo, y automáticamente en la pantalla podrían salir opciones como las siguientes:
- Comprar galletas María online.
- Buscar tienda próxima.
- Enviar a un amigo.
- Ir a web de fabricante.
- Descargar politono.
- etc.

¿Y si en lugar de una caja de galletas fuera una foto un coche? ¿o a un monumento?
Esto deja patente la multitud de posibilidades que nos puede ofrecer.
El reconocimiento de imágenes a través de la cámara de móvil abre un mundo realmente fascinante.

13.2.6 BitMakers

Empresa especializada en automatización industrial y desarrollo de soluciones de negocio. Ofrecen soluciones en Automatización Industrial cómo Reconocimiento de Imágenes.

Disponen de una gran experiencia en automatización industrial, que conjuntamente con sus productos innovadores, fiables y competitivos como Reconocimiento de Imágenes les permiten aportar soluciones de calidad y aportar mayor valor a sus clientes.

Esta especializada en ofrecer soluciones a sus clientes utilizando las últimas tecnologías, algunos de sus productos que podríamos destacar son:
productos basados en Reconocimiento de Imágenes
Detectores de Fallos en Cadena de Montaje
Detección de Nivel en envase
Cabezales Laser
Sensores RGB

14. APÉNDICE B: ALGORITMO DE CANNY

La diferencia del algoritmo de Canny [11] con los otros algoritmos es que Canny esta basado en un desarrollo analítico de optimización partiendo de un modelo continuo unidimensional de un escalón.

Consideramos una función escalón con amplitud *hE* afectada por un ruido gaussiano de desviación típica σ_n, supongamos que la detección de bordes se va a llevar a cabo mediante la convolución[1] de una función continua unidimensional *f(x)* con una función respuesta antisimétrica *h(x),* cuya amplitud fuera del intervalo *[-W, W]* es igual a cero.

El borde que busca la función *f(x)* se marcará donde aparezca el máximo local gradiente, obtenido tras la convolución de *f(x)* con *h(x)*.

Para poder determinar la función *h(x)* es necesario que se cumplan los siguientes requisitos:

Buena detección: Se maximiza la amplitud de la relación señal-ruido (*snr*) del gradiente para obtener una mayor probabilidad de marcar el borde donde lo hay y menor probabilidad de marcarlo donde no la hay.

Para detectar los bordes se utilizan dos umbrales T_1 y T_2, siendo T_2 el umbral mayor. Si únicamente, se usase T_1 se detectarían bordes poco importantes. Si por el contrario usáramos solo T_2, nos dejaríamos sin detectar puntos que realmente sí que pertenecen a los bordes.

Por este motivo lo que haremos será imponer la condición de que para que un punto pertenezca a un borde ha de ser mayor que T_2 o mayor que T_1 siempre que algún punto vecino sea mayor a T_2.

La relación señal-señal ruido para el modelo e estudiar se obtiene de la siguiente expresión:

$$snr = \frac{h_E}{\sigma_n} S(h)$$

Donde S (h) se obtiene de la siguiente manera:

[1] **Convolución.** Es una función matemática sencilla que es la base de muchos operadores de procesamiento de imágenes. Proporciona una manera de "multiplicar" dos arreglos de números, generalmente de tamaños distintos, pero la B misma dimensionalidad, para productor un tercer arreglo de números de la misma dimensionalidad.

$$S(h) = \frac{\int_{-w}^{0} h(x)}{\int_{-w}^{w} [h(x)^2]dx}$$

Buena localización: Los puntos del borde marcados por el operador deben estar tan cerca del borde como sea posible.

Definimos el factor de localización con la siguiente expresión:

$$LOC = \frac{h_E}{\sigma_n} L(h)$$

Donde L (h) se consigue de la siguiente fórmula:

$$L(h) = \frac{h'(0)}{\int_{-w}^{w} [h(x)^2]dx}$$

Basándonos en los dos requisitos anteriores obtenemos que el detector óptimo de bordes de escalón ha de ser un escalón truncado (diferencia de escalones), similar al mostrado en la fig. 1.

Por otro lado, éste operador tiende a generar muchos máximos locales como respuesta a bordes ruidosos de tipo escalón. Aunque si seguimos el criterio del primer requisito, estos bordes tendrían que considerarse erróneos.

Descartamos todos aquellos puntos que no sean máximos, y sólo aquellos que sean máximos en su entorno serán los candidatos a ser contorno.

Si se examinara la respuesta del operador de diferencia de escalones, se podría observar que la salida a un escalón con ruido produce un pico triangular con varios máximos cercanos al borde. Por ese motivo es necesario introducir un nuevo requisito para poder corregir este problema.

Respuesta única: Obligación de obtener una única respuesta para cada borde.

La distancia que hay entre picos del gradiente cuando únicamente existe ruido se representa como X_m y se establece como una fracción de K del ancho del operador y se puede obtener con la siguiente fórmula:

$$X_m = KW$$

El operador de Canny utiliza estos tres criterios para minimizar el producto $S(h)xL(h)$.

Debido a la complejidad de esta formulación no existe una solución analítica a este problema.

En la siguiente ilustración se refleja la respuesta impulsional del operador Canny para distintos valores de X_m.

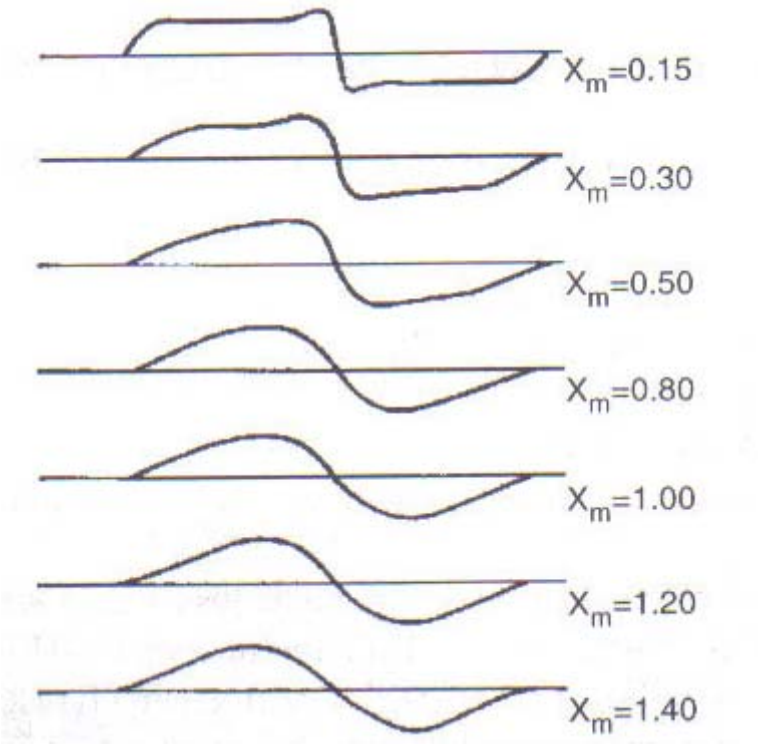


Fig. 53: Respuestas impulsionales del operador Canny

Fig. 54: Respuestas impulsionales de la derivada de la Gaussiana

Como podemos observar en las fig. 53 y 54 el operador X_m para pequeños valores se asemeja a una función cuadrada mientras que para valores altos se aproxima a una derivada Gaussiana.

La conclusión que llegamos con estas observaciones es que el operador óptimo es equivalente a la derivada de una gaussiana.

15. APÉNDICE C: MODELOS

Árboles de decisión [13]: Son utilizados para clasificar vectores de atributos (características de la imagen). Permiten establecer en que orden testear los atributos para conseguir la clasificación del vector de entrada. La base de este orden es la correcta elección de los atributos. Se escogerán en primer lugar aquellos que aporten mayor ganancia de información2 y por tanto nos faciliten descubrir a que clase pertenece el vector de entrada. En la fig. 55 se muestra un ejemplo de un árbol de decisión.

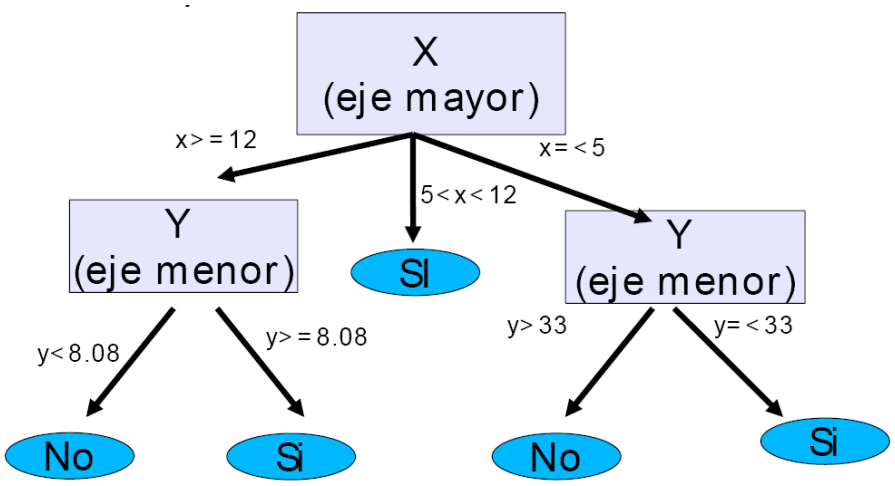


Fig. 55: Ejemplo de árbol de decisión

[2] Ganancia de información: Medida de cuanto ayuda el conocer el valor de una variable aleatoria X (atributo) para conocer el verdadero valor de otra Y (clase). Una alta ganancia permite reducir la incertidumbre de clasificación del ejemplo de entrada.

Redes neuronales [14]: Las redes neuronales están formadas por una serie de unidades de procesamiento muy simples que están conectadas entre sí y, cuyas salidas dependen, además de la entrada, de las conexiones existentes entre las unidades de procesamiento.

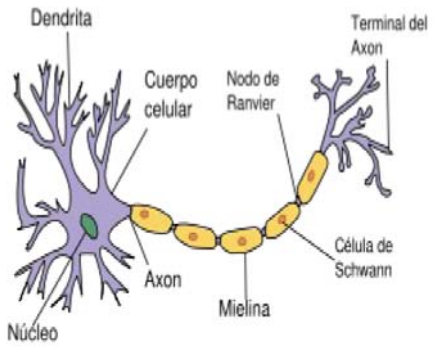
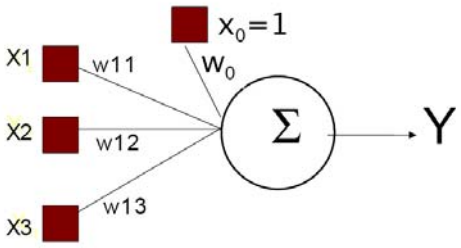
NEURONA NATURAL	CARACTERÍSTICAS
	Cargada electronegativamente : -70mV Canales iónicos. Si supera este umbral se produce una diferencia de potencial.
NEURONA ARTIFICIAL	**CARACTERÍSTICAS**
	Intención de conseguir similitud biológica. Representa la unidad de entrada. Dividida en 2 partes: Capa de entrada → X_1, X_2 ,.. Capa de salida → Y

Fig. 56: Características de una neurona biológica y una neurona artificial

Como se muestra en la fig. 56, las redes neuronales se construyen a partir del modelado formal (normalmente matemático) del funcionamiento de las redes neuronales de los seres vivos.

Los cuerpos celulares se representan por los nodos, y los caminos entre neuronas, formados por axones conectados con dendritas a través de la sinapsis, que se representan mediante enlaces.

Son muy útiles a la hora de resolver problemas en los que sea muy difícil poder separar las clases de un problema ya que permiten particionar el espacio de características mediante fronteras complejas lineales a trozos, añadiendo capas a la red.

Las características fundamentales que hacen de las redes neuronales una potente herramienta para el diseño de sistemas de reconocimiento de formas son la capacidad de aprendizaje, la facilidad de diseño y la capacidad de manejo de problemas no-lineales.

Redes Bayesianas: Son grafos dirigidos acíclicos cuyos nodos representan variables y los arcos que los unen codifican dependencias condicionales entre las variables. Los nodos pueden representar cualquier tipo de variable, ya sea un parámetro medible (o medido), una variable latente o una hipótesis.

Si existe un arco que une un nodo A con otro nodo B, A es denominado un padre de B, y B es llamado un hijo de A. El conjunto de nodos padre de un nodo Xi se denota como padres(Xi). Un gráfico acíclico dirigido es una red Bayesiana relativa a un conjunto de variables si la distribución conjunta de los valores del nodo puede ser escrita como el producto de las distribuciones locales de cada nodo y sus padres:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid padres(X_i))$$

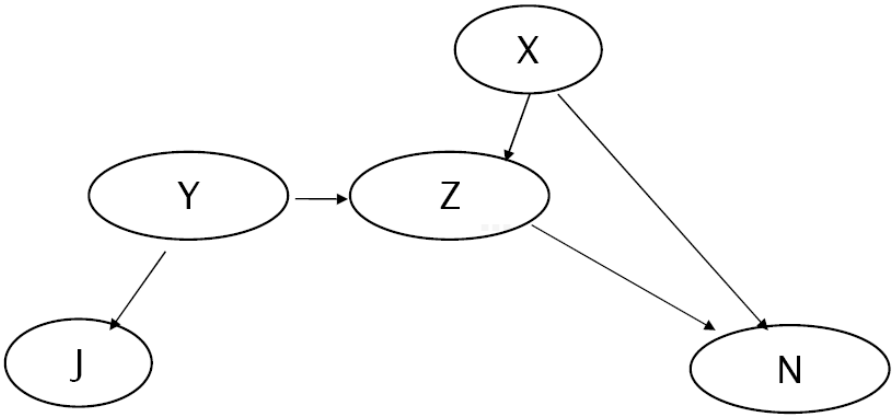


Fig. 57: Ejemplo de red Bayesiana

El nodo X de la fig. 57 no tiene padres, por tanto su distribución local de probabilidad se toma como incondicional, en otro caso es condicional. Si el valor de un nodo es observable y por tanto etiquetado como observado, dicho nodo es un nodo de evidencia.

16. APÉNDICE D: MÉTODOS

16.1 Clase Entrenar

void Entrenar::ObtenerRuta();

Permite obtener la ruta donde se encuentran las imágenes.

void Entrenar::OnButEntrenar();

Función invocada al pulsar el botón entrenar de la pantalla de entrenamiento. Es la función más compleja de la clase, realiza el proceso de extracción de las características de las imágenes que se hayan introducido en la aplicación. Deja los vectores resultantes en un archivo '*.txt' bajo la ruta predeterminada. En el flujo de ésta función se realizan invocaciones a muchas de las funciones comentadas.

int Entrenar::obtener_xmax(int region,int ancho,double r_anch, int n, int m);

Parámetros de entrada:
Int region: Nº de región
Int ancho: Ancho de la imagen
Double r_anch: Ancho de la región
Int n: Parámetro introducido por el usuario que indica el nº de regiones en que se dividirá la imagen en término de coordenadas y.
Int m: Parámetro introducido por el usuario que indica el nº de regiones en que se dividirá la imagen en término de coordenadas x.
Función que calcula la x máxima de una región. Retorna el valor de xmax.

int Entrenar::obtener_ymax(int region,int alto,double r_alt, int n, int m)

Esta función calcula la 'y' máxima de una región.

int Entrenar::obtener_xmin(CRegion *pRegion,int region,int ancho,int n, int m);

Esta función calcula la 'x' mínima de una región.

int Entrenar::obtener_ymin(CRegion *pRegion,int region,int alto, int n, int m);

Función que calcula la 'y' mínima de una región

void Entrenar::normalizar(double *distancias,int n)

Parámetros de entrada:
Double distancias: Vector de entrada
Int n: tamaño del vector
Función que realiza el procceso de normalización del vector de entrada. Retorna un nuevo vector resultado de la normalización.

void Entrenar::inversa(double *distancias, int n)

Función que realiza el cálculo de la inversa del vector de entrada. Retorna un nuevo vector resultado de la inversa.

void Entrenar::ActualizarSalida(double * distancias, double * salida, int regiones);

Parámetros de entrada:
double distancias: Vector de entrada que contiene las distancias de un punto a los centroides de las regiones vecinas y al suyo propio.

Double *salida: Vector que contiene la suma de los vectores distancias.

Int regiones: Nº total de regiones.
Función que actualiza el contenido de los elementos del vector salida sumando los elementos del vector distancias con valor diferente a 0 y con posición correspondiente.

CString Entrenar::ObtenerNomClasImg(CString nombre);

Parámetros de entrada:
Nombre: Nombre de la imagen
Función que a partir del nombre de la imagen crea el nombre de la clase. Retorna un CString que contiene el nombre de la clase.

16.2 Clase Descripción

IplImage * Descripcion::cargarImagen(CString rutanombreImagen, IplImage *img);

Parámetros de entrada:
CString rutanombreImagen: Ruta donde está almacenada la imagen
IplImage *img: Puntero de tipo IplImage que corresponde a una estructura propia de las Open CV que representa una imagen.
Ésta función asigna una imagen almacenada en nuestro ordenador a una estructura IplImage.
Retorna un puntero a la estructura IplImage donde se ha cargado la imagen.

bool Descripcion::ExisteFichero(CString nombre,CString ruta1);

Parámetros de entrada:
CString nombre: Nombre del fichero
CString ruta1: Ruta donde ha de encontrarse el fichero.
Ésta función comprueba la existencia de un fichero en la ruta especificada. Retorna TRUE si realmente existe tal fichero en esa ruta y FALSE en caso contrario.

CString Descripcion::nombreFichero(CString nombre);

Función que asigna nombre al fichero donde se almacenarán los vectores de características utilizando como base el nombre de la imagen de entrada.

void Descripcion::calcDist_Vecinos(CPoint *punto,CRegion *pRegion,int r,double *distancias, int num_vecinos);

Parámetros de entrada:
CPoint *punto: Puntero de tipo CPoint que representa un punto del contorno
CRegion *pRegion: Puntero a CRegion que representa una región
Int r: Nº identificador de la región
Double *distancias: Puntero del vector de distancias
Int num_vecinos: Numero de vecinos propios de la región
Ésta función realiza el cálculo de las distancias de un punto del contorno a los centroides de las regiones vecinas y al de la suya propia.

int Descripcion::asignar_vecino(int vec,int r,int n,int m);

Parámetros de entrada:
Int vec: Nº identificador de vecino.
Int r: Nº identificador de la región.
Int n,m: comentadas anteriormente.
Ésta función comprueba que la región de entrada y el número de vecino de entrada sea correcto y seguidamente le asigna el vecino a la región de entrada. Retorna el número de región que corresponde al vecino, en caso de no corresponderle la función retorna -1.

bool Descripcion::Existe_vecino(int vecino,int vecin,int r,int n,int m);

Ésta función es invocada por la función asignar_vecino y realiza la validación de la existencia del vecino para una determinada región. Retorna TRUE si realmente existe el vecino y FALSE en caso contrario.

IplImage * Descripcion::Canny(IplImage * img,IplImage * imgkny);

Parámetros de entrada:
IplImage *img: Puntero de tipo IplImage que corresponde a una estructura propia de las Open CV que representa una imagen.
IplImage *imgKny: Puntero de tipo IplImage que corresponde a una estructura propia de las Open CV que representa una imagen.
Función que invoca a la función CVCanny propia de las OpenCV que extrae los contornos de la imagen de entrada img y guarda el resultado en la segunda imagen de entrada imgKny. Retorna un puntero a la imagen generada por el CVCanny.

int Descripcion::calcRegion_Punto(int px, int py,CRegion *pRegion,int n, int m);

Parámetros de entrada:
Int px: Coordenada 'x' de un punto determinado
Int py: Coordenada 'y' de un punto determinado
CRegion *pRegion, int n, int m: Comentados en funciones anteriores
Dependiendo de los valores de px y py se calcula a que región debe pertenecer el punto.
Retorna el identificador al que pertenece el punto.

IplImage * Descripcion::pintarImagen(int ancho, int alto, CRegion *pRegion,int n,int m, double * entrada);

Parámetros de entrada: Todos ya han sido comentados en funciones anteriores.

Utilizando los valores del vector entrada crea una imagen en escala de grises. Asigna a todos los puntos de una región el valor que le corresponde del vector de entrada. Muestra por pantalla la nueva imagen. Retorna la imagen resultante.

16.3 Clase RecoImagen

void RecoImagen::OnReconocer();

Función invocada al pulsar el botón Reconocer de la pantalla Reconocer. Realiza el proceso de reconocimiento de una imagen utilizando como base un clasificador binario simple y distancia euclídea como base en las comparaciones uno contra uno.
int RecoImagen::ObtenerLineasFichero(FILE *imgs);

Parámetros de entrada:
FILE *imgs: Puntero de tipo FILE.
Ésta función recorre el fichero y calcula el número de líneas que tiene. Retorna el número de líneas del fichero.

double RecoImagen::compararVectores (double *sal, double *ent, int m);

Parámetros de entrada: comentados en funciones anteriores
Realiza la comparación de los dos vectores de entrada utilizando el algoritmo de la Distancia Euclídea. Retorna el valor de la comparación.

double RecoImagen::compararImagenes(double *sal, double *ent, int n, int m);

Parámetros de entrada: comentados en funciones anteriores.
Realiza la comparación de los dos vectores de entrada utilizando el algoritmo de la Distancia Euclídea. Retorna el valor de la comparación.

void RecoImagen::CrearFicheroTemporal(char temp[150], char origen[150],int n, int m);

Parámetros de entrada:
Int n, int m: ya comentados en anteriores funciones
Char Temp. [150]: Cadena de characters que contiene el nombre del fichero temporal
Char origen[150]: Cadena de characters que contiene el nombre del fichero origen.
Realiza una copia del fichero de origen en otro temporal.

int ** RecoImagen::Aprendizaje(int n);

Parámetros de entrada: comentados en funciones anteriores
Función que genera la matriz de aprendizaje (0's y 1's). Retorna un doble puntero que representa la matriz de aprendizaje.

void RecoImagen::OnReconocer2();

Función invocada al pulsar el botón Reconocer2 de la pantalla de Reconocer. Realiza el proceso de reconocimiento de una imagen utilizando como base un clasificador binario basado en ECOC y distancia Euclídea como base en las comparaciones uno cotra uno.

int RecoImagen::obtenerNumeroClases();

Función que obtiene el número de clases que utilizará el clasificador.

int * RecoImagen::Reconocer2()

Función invocada por la funcion OnReconocer2()
Realiza el proceso de comparaciones binarias entre los elementos de las distintas clases utilizadas por el descriptor en el proceso de reconocimiento. Retorna un puntero a un vector de enteros (1,-1,1…) que contiene los resultados de las comparaciones.

16.4 Clase Porcentaje

void Porcent::PorcentAcierto();

Función que es invocada al pulsar el botón Calcular en la pantalla de porcentaje. Calcular la probabilidad de acierto del sistema en el proceso de reconocimiento de objetos.

17. APÉNDICE E: CONTENIDO DEL CD

17.1 Requisitos previos

Para plataformas Win32:

-Win9x/WinNT/Win2000/WinXP/

-Compilador C++ (Incluido con la distribución de Microsoft Visual C++ 6.0)

Este software ha sido probado únicamente sobre Windows XP, por lo que no podemos garantizar el correcto funcionamiento bajo otro S.O. aunque podríamos asegurar que no surgirían problemas bajo otros S.O. diseñados bajo la plataforma Win32.

17.2 Instalación

19.2.1 Instalación de las librerias OpenCV para MVC++

Bajarnos el fichero OpenCV_1.0.exe y dejarlo en la unidad C:.
Realizar doble clic sobre el fichero OpenCV_1.0.exe y dejar las opciones que nos vayan a pareciendo en las diferentes pantallas del asistente.

Una vez finalizada la instalación comprobaremos que se nos ha creado la carpeta OpenCV dentro de la siguiente ruta:

C:\Archivos de programa\Intel

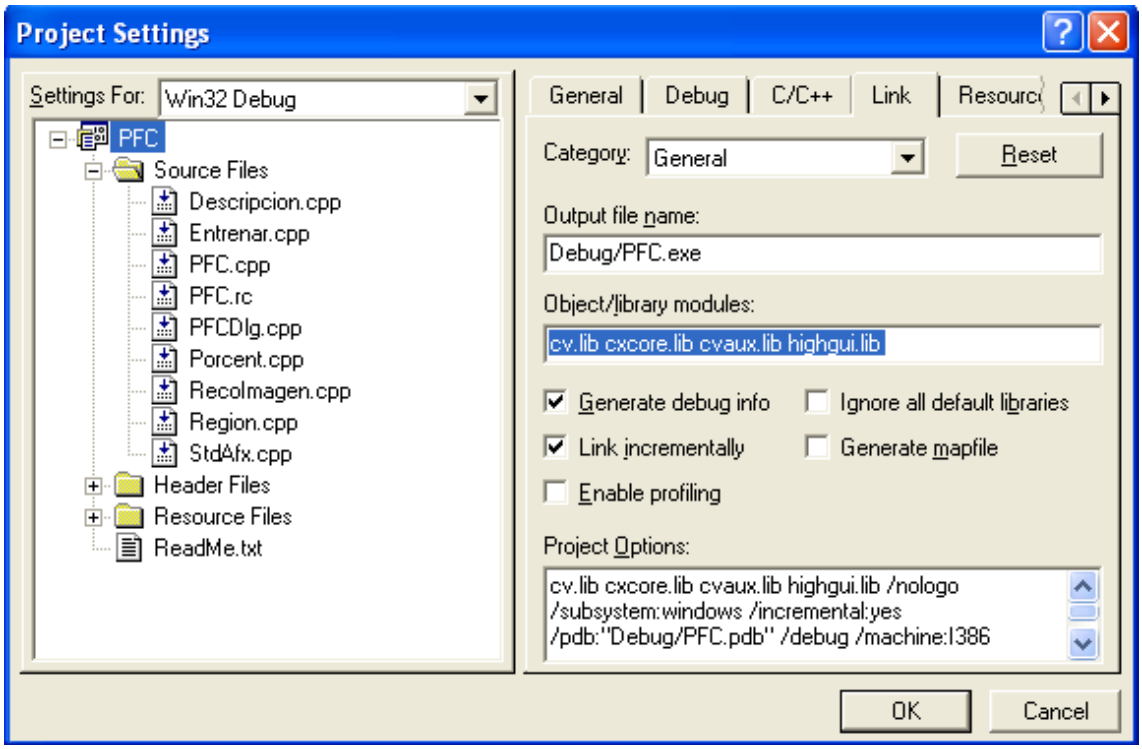
Finalizado el proceso de instalación se inicia el proceso de configuración de nuestro entorno para poder utilizar estas librerías.

Lo primero que debemos hacer es importar los siguientes ficheros de cabeceras como dependencias externas.

- 'ipl.h', 'cv.h', 'cxcore.h', 'highgui.h', 'cvaux.h', 'cvhaartraining.h'

Para realizarlo accedemos al menú 'Project' escogemos 'Settings' y seleccionamos sobre la pestaña 'Link'.
Una vez estamos en la pestaña 'Link ' añadir en el campo con etiqueta 'Object/library modules' lo siguiente:

ipl.lib cv.lib cxcore.lib highgui.lib cvaux.lib cvhaartraining.lib (Si queremos utilizar funciones de detección de caras cosa que no es nuestro caso)



Fig, 58: Pantalla de Project Settings

El siguiente paso que hemos de realizar es indicar al sistema donde se encuentras los directorios de estas librerías, para realizarlo volvemos a acceder al menú 'Tools' escogemos 'Options' y seleccionamos en la pestaña con el nombre de 'Directories'.
Situados en la pestaña 'Directories'

En el campo de texto con etiqueta 'Platform' escoger 'Win32'
En el campo de texto con etiqueta 'Show directories for' escoger 'Include Files'

Por último los siguientes directorios en la lista inferior con etiqueta 'Directories:'

C:\ARCHIVOS DE PROGRAMA\INTEL\PLSUITE\INCLUDE
C:\ARCHIVOS DE PROGRAMA\INTEL\OPENCV\CV\INCLUDE
C:\ARCHIVOS DE PROGRAMA\INTEL\OPENCV\OTHERLIBS\HIGHGUI
C:\ARCHIVOS DE PROGRAMA\INTEL\OPENCV\CXCORE\INCLUDE
C:\ARCHIVOS DE PROGRAMA\INTEL\OPENCV\apps\HaarTraining\include

(Si utilizamos detector de caras que como ya hemos comentado antes no será nuestro caso)

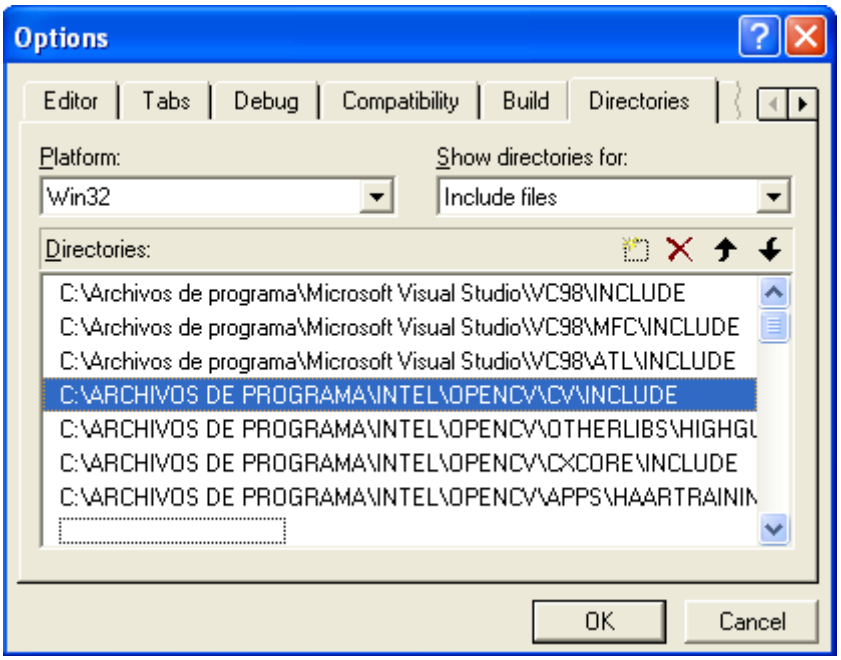


Fig. 59: Pantalla de Opciones

Sin movernos de la pestaña de 'Directories' cambiamos el valor del campo de texto con etiqueta 'Show directories for:' y introducimos Library files.
Añadimos las siguientes rutas en la lista inferior de 'Directories'

- C:\ARCHIVOS DE PROGRAMA\INTEL\PLSUITE\LIB\MSVC
- C:\ARCHIVOS DE PROGRAMA\INTEL\OPENCV\LIB

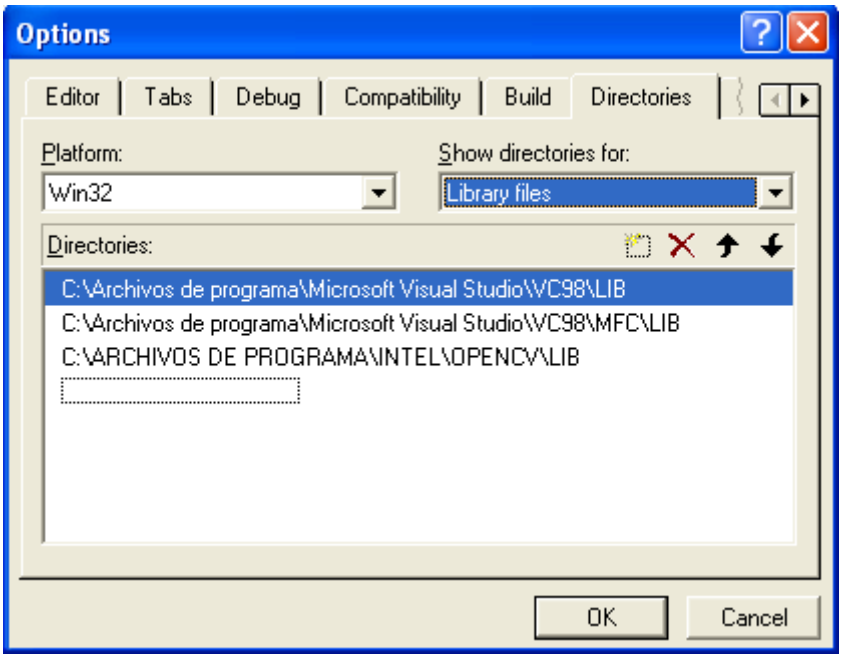


Fig. 60: Pantalla de Opciones

Por último solo nos queda definir las variables de entorno necesarias para que el sistema sitúe las DLL.
Para acceder a las variables de sistema pulsamos inicio, botón derecho sobre el icono Mi PC, escogemos Propiedades.
Dentro de la ventana de Propiedades, elegimos la pestaña de Opciones avanzadas.
En la pestaña de opciones avanzadas seleccionamos el botón con nombre Variables del sistema, dentro de la pantalla que se nos muestra podemos ver las variables de entorno del sistema.
Realizaremos las siguientes modificaciones.

A la Variable Path le añadimos al final de su descripción la siguiente ruta: %SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;C:\Archivos de programa\ATI Technologies\ATI Control Panel;**C:\Archivos de programa\Intel\OpenCV\bin**

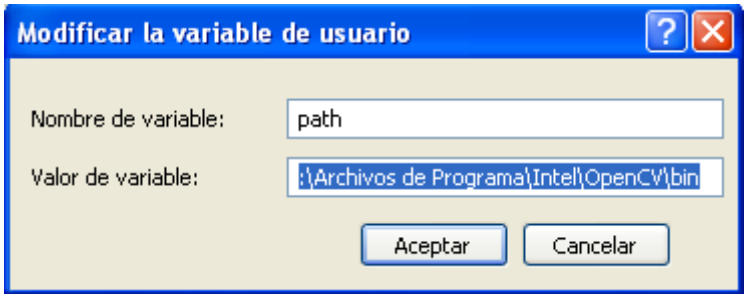


Fig. 61: Pantalla de Variables de Entorno

17.2 Contenido del CD

17.2.1 Código del ejecutable

Definimos una estructura de carpetas para incluir de forma ordenada los ficheros necesarios para poder ejecutar el programa.
Creamos una carpeta con el nombre de PFC. Esta carpeta contiene los ficheros relacionados con la implementación del software. En el interior de esta carpeta encontramos 2 subcarpetas, la primera con el nombre de Código Ejecutable donde se encuentra el código ejecutable del programa. Destacamos de los ficheros ejecutables el fichero con nombre PFC.exe ya que es el que utilizaremos para arrancar la aplicación.
La segunda subcarpeta recibe el nombre de Código fuente, su contenido se explicará en el siguiente apartado.

17.2.2 Código fuente del programa

Incluimos en el CD todos los ficheros fuente utilizados. Entre los ficheros fuente se encuentran:

- Ficheros fuente : Implementación de las funciones de la aplicación
- Ficheros de cabecera : Definición de las funciones de la aplicación
- Icono de la aplicación

17.2.3 Ficheros de imágenes

Además del código propio del programa incluimos un conjunto de imágenes en blanco y negro y en color.

Las imágenes están incluidas dentro de la carpeta Imágenes. Esta a su vez contiene dos subcarpetas, la primera con el nombre de blanco/negro y la segunda con el nombre de color.

18. BIBLIOGRAFÍA

[1] Winston, Patrick H., "Inteligencia Artificial", Addison-Wesley Iberoamericana, 3ª ed.,1994.

[2]Winograd, T.A., Flores, "F. Understanding Computers and Cognition", Norwood, N.J. 1986.

[3]Javier Gonzalez Jiménez , "Visión por Computador", Editorial Paraninfo, 200

[4] C.Burges, "A tutorial on support vector machines for pattern recognition", Knowledge Discovery and Data Mining 2(2), 1998.

[5]http://www.informatik.uni-trier.de/~ley/db/conf/icvs/icvs2008.html#EscaleraPR08

[6]Kuncheva L., "Combining pattern classifiers", Wiley, 2004

[7] http://members.fortunecity.es/davidweb2/visart/bordes.htm

[8]http://campusvirtual.uma.es/tdi/www_netscape/TEMAS/Tdi_11/index.php#2.-%20El%20método%20de%20Kirsch

[9] Wai-Pak Choi, Kin-Man Lam and Wan-Chi Siu, "Pattern Recognition", 2003.

[10] Lewis G. Minor, "Application of Pattern Recognition and Image Processing Techniques to Lock-on-After-Launch Missile Technology," US Army Missile Laboratory, Alabama, Apr. 5, 1981.

[11] Lijun Ding and Ardeshir Goshtasby, "Pattern Recognition", 2001

[12] http://groups.yahoo.com/group/OpenCV/

[13]Breiman, L., Friedman, J., Olshen, R., and Stone C. ,"Classification and Regression Trees" , Wadsworth, 1984

[14] Bishop C. , "Neural Networks for Pattern Recognition", Clarendon Press, Oxford, 1995

[15] Leo Breiman, "Bagging predictors", Machine Learning, 1996

[16] S. Kotsiantis, P. Pintelas, "Combining Bagging and Boosting", International Journal of Computational Intelligence, 2004

[17] http://www.robots.ox.ac.uk/~az/lectures/cv/adaboost_matas.pdf

[18] E. Kong and T. Dietterich, “Error-correcting output coding corrects bias and variance”, In Proceedings of the 12th International Confer- ence on Machine Learning, 1995.

[19] Carlos J. Alonso, Oscar J. Prieto, Juan J. Rodríguez, Aníbal Bregón and Belarmino Pulido1 “Current Topics in Artificial Intelligence”, Springer Berlin / Heidelberg, 2007

[20] Jäppinen A., Beauregard R., “Scandinavian Journal of Forest Research”, 2000

[21] Belur V. Dasarathy, “Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques”, 1991

[22] http://es.wikipedia.org/wiki/Distancia

[23] Basilio Sierra Araujo, “Aprendizaje Automático: conceptos básicos y avanzados”, (2006)

[24] Chuck Sphar, “Aprenda Microsoft Visual C ++ 6.0 Ya” , 2001

[25] http://msdn.microsoft.com/en-us/library/ddk909ch(vs.71).aspx

[26] msdn.microsoft.com/en-us/library/t9adwcde(VS.80).aspx?

[27] Microsoft Corporation, “Microsoft Visual C++ Programming with MFC”, Microsoft Press, (1995).

[28] http://msdn.microsoft.com/de-de/library/abx4dbyh(VS.80).aspx

[29] http://msdn.microsoft.com/es-es/library/ms174184.aspx

[30] http://www.computerworld.com/action/article.do?command=viewArticleBasic&

articleId=9015280

[31] http://www.microsoft.com/spain/interop/smartcless.mspx

[32] Madhavanath, S. & Govindaraju, V., “Preceptual features for off-line handwritten word recognition: a framework for prediction, representation and matching.” Advances in Pattern Recognition, august, 1998,

[33] W. Zhang, L. Wenyin, K. Zhang, “Symbol recognition with kernel density matching”, IEEE Trans. Pattern Anal. Mach, 2006

[34] CSS F. Mokhtarian, A. Mackworth, “Scale-based description and recognition of planar curves and two-dimensional shapes”, in IEEE TPAMI, 1986

[35] D.Lowe,”Distinctive Image Features from Scale-Invariant Keypoints”, Journal of Computer Vision, 2007