Universidad Carlos III de Madrid

Escuela Politécnica Superior

Departamento de Teoría de la Señal y Comunicaciones

Ingeniería Técnica de Telecomunicación

especialidad Sonido e Imagen

Proyecto de Fin de Carrera

APLICACIÓN DEL MODELO *BAG-OF-WORDS* AL RECONOCIMIENTO DE IMÁGENES

Autor: Sara Pardo Feijoo

Tutor: Iván González Díaz

Cotutor: Darío García García

Julio de 2009

Abstract

Object recognition on images has been more investigated in the recent years. Its principal application is the image retrieval and, therefore, image searchers would find the solution to the query based on whether the image has certain objects in its visual content or not instead of based on the adjacent textual annotations. Content based image retrieval would improve notoriously the quality of searchers. It is neccesary to have models that classify an image based on its low level features. In this project, it is used the 'Bag of words' model. Multimedia information retrieval entails many fields involved, and has many applications. The objective of this project is the indexing of images of a database based on content. It tries to eliminate the semantic gap finding the descriptors of each imagen, and therefore decide to which class or which semantic concept belongs.

Index terms: bag of words, multimedia information retrieval, content based image retrieval, low level features, descriptors.

Índice de contenidos

Índice de figuras

Cap. 1: Introducción y objetivos del proyecto

1.1. Introducción

El reconocimiento de objetos en imágenes es un campo cada vez más investigado y que se aplica principalmente a la recuperación de imágenes basada en contenido, es decir, a buscadores de imágenes que encontrarán la solución a una consulta basándose en si la imagen contiene ciertos objetos o no en función de su contenido visual, y no de las anotaciones textuales colindantes. Su aplicación surge de la necesidad de sistemas de gestión automatizada de documentos multimedia que sustituyan a la gestión manual, ya que ciertas bases de datos de información multimedia tienen tamaños impracticables para realizar una anotación manual.

La recuperación de imágenes basada en contenido mejoraría significativamente la calidad de las búsquedas. Para ello es necesario disponer de modelos que se enfrenten a la clasificación de una imagen a partir de sus características de bajo nivel. En este proyecto se va a utilizar el modelo *Bag-of-words* (BoW).

La recuperación de información multimedia conlleva muchos campos involucrados: clasificadores de información, estadísticas de señales, visión artificial… Por otro lado, también tiene multitud de aplicaciones: buscadores Web, detección de rostros en fotografías, recuperación de imágenes médicas, robótica, etc.

Este proyecto tiene como objetivo la indexación de las imágenes de una base de datos basándose en el contenido. Trata de eliminar la laguna semántica hallando los descriptores de cada imagen de la base de datos para luego discernir a qué clase o concepto semántico pertenecen.

La presente memoria está estructurada de la siguiente manera:

- Capítulo 2: se introducirán conceptos de importancia en el proyecto tales como la visión artificial y los sistemas de recuperación y anotación de imágenes. Además se tratarán técnicas involucradas en la realización del proyecto como los sistemas de clasificación de patrones, algunas características de bajo nivel, el modelo *bag-of-words*, técnicas de agrupamiento o *clustering*, algoritmos de aprendizaje máquina para clasificación y medidas de calidad que permitan evaluar los sistemas.

- Capítulo 3: este capítulo describe el modelo propuesto en el proyecto, explicando su desarrollo e implementación, y comentando elementos tales como la base de datos empleada, el desarrollo del proyecto, la implementación, los experimentos realizados y su evaluación.

- Capítulo 4: en este capítulo se presentan las conclusiones más relevantes del estudio y se introducen las líneas futuras que permitirían extender el mismo.

- Capítulo 5: en esta sección se proporciona un presupuesto económico detallado de la realización del proyecto.

- Apéndices: por último, esta memoria incluye algunos apéndices donde se explica el entorno de desarrollo, se proporcionan nociones básicas acerca de la utilización del programa, se comentan las librerías de funciones externas empleadas en el proyecto, y se proporciona el código fuente software desarrollado.

1.2. Objetivo del proyecto

El objetivo de este proyecto consiste en la aplicación del modelo *bag-of-words* (BoW, bolsa de palabras) al reconocimiento de imágenes. Este modelo, procedente del mundo del análisis y clasificación de contenidos textuales requiere de una cierta adaptación al dominio visual y la clasificación de imágenes.

El modelo BoW define una metodología de trabajo para clasificar imágenes, si bien numerosos aspectos concretos de su aplicación quedan pendientes del diseño del desarrollador. Dichos parámetros sobre el modelo, tales como el tamaño y estructura de los vocabularios, o el empleo de clasificadores de distinta índole (perceptrones y máquinas de vectores de soporte), serán estudiados a lo largo del proyecto.

Se implementarán diversas alternativas para la clasificación, lo que requerirá el cálculo de los parámetros óptimos para cada una de las herramientas y la comparativa de todos los resultados mediante el cálculo de medidas de calidad.

Cap. 2: Estado del arte

2.1. Visión artificial

La visión artificial es la iniciativa de automatizar e integrar un amplio rango de procesos y representaciones usadas para la percepción de la visión. Incluye muchas técnicas que son usadas por si mismas, como el procesamiento de imágenes (transformar, codificar y transmitir imágenes) o la clasificación estadística de patrones (teoría estadística de la decisión aplicada a patrones generales, visuales u otros). Por otro lado, también considera técnicas de modelado geométrico y procesamiento cognitivo.

2.1.1. Habilidades de alto y bajo nivel

La visión artificial se enfrenta a un problema muy difícil; debe reproducir, a través de hardware digital, las capacidades del sistema visual humano, de diversos caracteres: análogos y paralelos, o especializados.

La visión artificial se preocupa tanto de las cuestiones de procesamiento temprano o de bajo nivel como del uso del conocimiento o alto nivel.

2.1.1.1. Habilidades de alto nivel

Las habilidades de alto nivel, como los objetivos y el conocimiento son muy importantes, ya que pueden mejorar considerablemente un sistema visual. Éstas requieren cálculo y una gran cantidad de conocimiento de objetos en el mundo, cómo son y cómo se comportan. Las potencias de alto nivel están tan bien integradas en la "visión" que pueden ser efectivamente separables.

2.1.1.2. Habilidades de bajo nivel

La visión requiere muchas habilidades de bajo nivel que habitualmente se dan por hecho; por ejemplo, nuestra habilidad para extraer imágenes intrínsecas de “brillo”, “color” y “rango”. La fusión estéreo (estereopsis) es una habilidad de bajo nivel para la percepción tridimensional de rango corto.

Una importante habilidad de bajo nivel es la percepción de objetos. Los sistemas de visión biológicos maduros están especializados y puestos a punto para tratar con los objetos relevantes en sus entornos.

Un tipo básico de capacidad de reconocimiento de objetos es la discriminación “*figure/ground*”, que separa el objeto del fondo. Otras predisposiciones de organizaciones básicas son reveladas por las leyes Gestalt de agrupamiento, que demuestran reglas que nuestro sistema usa para formar muestrarios (*arrays*) simples de estímulos en grupos espaciales más coherentes (por ejemplo, la formación de un objeto completo a partir de sus partes constituyentes).

2.1.2. Sistemas de visión artificial

La percepción visual es la relación de la entrada visual con los modelos existentes previamente en el mundo. Hay una gran laguna representativa entre la imagen y los modelos (“ideas”, “conceptos”) que explican, describen o abstraen la información contenida en la misma. Para puentear dicha laguna, los sistemas de visión artificial normalmente tienen un rango de representaciones (vagamente ordenado) conectando la entrada y la salida (una descripción final, decisión o interpretación). La visión artificial entonces envuelve el diseño de esas representaciones intermedias y la implementación de algoritmos para construir y relacionar unos con otros.

Las representaciones se pueden categorizar en cuatro tipos. No todos los niveles necesitan ser usados en cada aplicación de visión artificial; algunos pueden ser saltados, o el procesamiento puede empezar en parte arriba de la jerarquía o terminar en parte abajo.

Las imágenes generalizadas son icónicas, y constituyen representaciones analógicas de los datos de entrada. El procesamiento independiente del dominio puede producir representaciones icónicas más útiles para el procesamiento de alto nivel, como muestrarios (*arrays*) de elementos del margen (por ejemplo discontinuidades de nivel de grises). En la Ilustración 1: “Representación icónica” se puede ver un ejemplo de representación icónica.

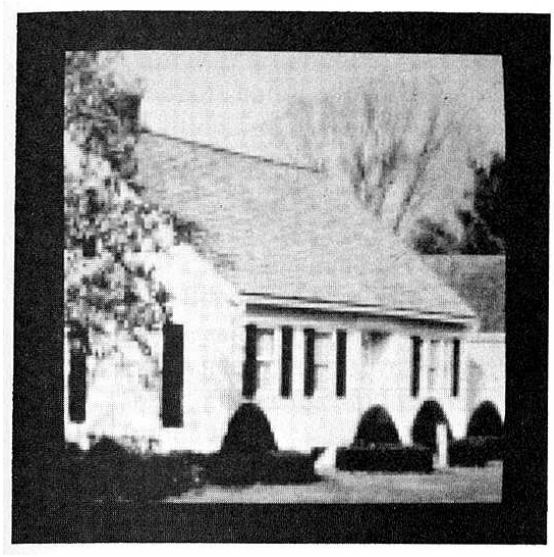

Ilustración 1: “Representación icónica”

Las imágenes segmentadas (segunda parte) surgen a partir de la imagen generalizada agrupando sus elementos en conjuntos propensos a ser asociados con objetos significativos en la escena. A partir de la imagen segmentada, el conocimiento sobre el dominio particular de un tema empieza a ser importante tanto para ahorrar cálculos como para superar los problemas del ruido y datos inadecuados. La textura y el movimiento son muy importantes en la segmentación de imágenes. En la Ilustración 2: "Representación segmentada" se puede ver un ejemplo de representación segmentada.

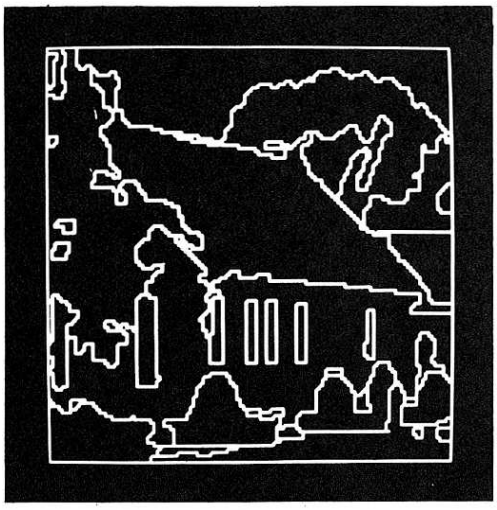

Ilustración 2: "Representación segmentada"

Las representaciones geométricas son usadas para capturar la idea importante de una forma bidimensional y tridimensional. Cuantificar una forma es tan importante como difícil. Estas representaciones geométricas deben ser lo suficientemente expresivas para soportar un procesamiento complejo y general, como la simulación de los efectos de la iluminación y el movimiento. Las estructuras geométricas son tan útiles para codificar el conocimiento adquirido previamente como para volver a representar la entrada visual actual. En la Ilustración 3: "Representación geométrica" se puede ver un ejemplo de representación geométrica.

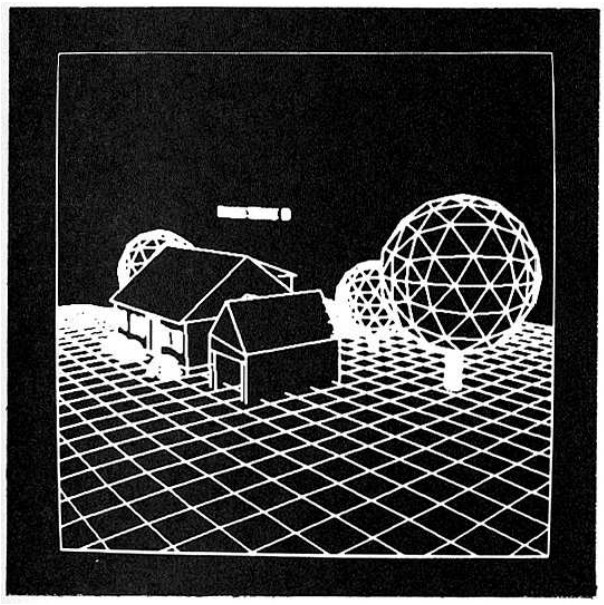

Ilustración 3: "Representación geométrica"

Los modelos relacionales son ensamblajes complejos de representaciones usadas para soportar procesamiento de alto nivel. Una herramienta importante en la representación del conocimiento son las redes semánticas, las cuales identifican y expresan las relaciones existentes entre diversos conceptos presentes en las representaciones precedentes.

El modelo básico del procesamiento cambia de construir las representaciones (a través del procesado de imagen) a relacionarlas [12]. En la Ilustración 4: "Representación relacional" se puede ver un ejemplo de representación relacional.

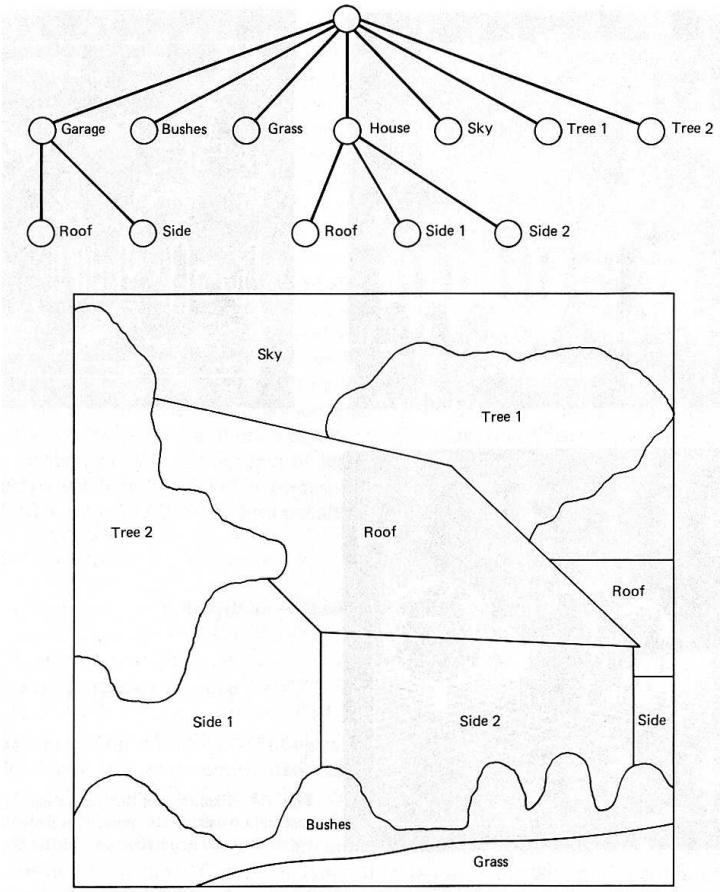

Ilustración 4: "Representación relacional"

2.2. Sistemas de recuperación y anotación de imágenes

2.2.1. Recuperación de la información

La recuperación de información multimedia (MIR, *multimedia information retrieval*) es la búsqueda de conocimiento en todas sus formas, en cualquier medio y de manera pertinente y relevante [10].

En estos últimos años ha habido una revolución de la información: el cambio desde el contenido pasivo de los usuarios que sintonizan servicios de difusión de formato rígido a los usuarios activos que demandan la propiedad de los medios y se vuelven editores [9].

La búsqueda de conocimiento sobre medios digitales comenzó hace muchas décadas cuando la idea de digitalizar los recursos multimedia era común, pero los libros seguían siendo el medio principal para almacenar conocimiento. Antes de que el campo de la información multimedia se fundiese con la comunidad científica, había muchos avances contributivos de varios campos científicos establecidos (inteligencia artificial, teoría de la optimización, visión artificial y reconocimiento de patrones).

Los primeros años de MIR se basaron frecuentemente en la visión artificial (1982) [10]. Los años 1994-2000 fueron la fase inicial de investigación y desarrollo en recuperación de imágenes por contenido. Las lagunas que definen y motivan la mayoría de los problemas relacionados son:

- Laguna sensorial: está entre el objeto en el mundo real y la información en una descripción computacional derivada de una grabación de esa escena.

- Laguna semántica: es la falta de coincidencia entre la información que uno puede extraer de los datos visuales y la interpretación que los mismos datos tienen para un usuario en una situación dada [11].

Ejemplos populares e influyentes de los primeros sistemas basados en visión artificial serían 'QBIC' (1995) y 'Virage' (1996) [11].

El sistema CBIR (*Content based image retrieval*) es el problema de búsqueda de imágenes digitales en grandes bases de datos que hace uso de los contenidos de las imágenes en sí mismas más que confiar en la información textual que las rodea. Estas técnicas usan las características extraídas automáticamente a partir del contenido (color, textura y forma) como criterios de búsqueda [15].

Ejemplos de motores de búsqueda por similitud de Internet son 'Webseek' (1997) y 'Webseer' (1996).

Un esfuerzo significante fue también la integración directa de las características basadas en búsqueda por similitud en las bases de datos a nivel empresarial para hacer el MIR más accesible a la industria privada [10]. En 2000, 'Smeulders et al.' establecieron que los dominios para la búsqueda de imágenes fueran clasificados como estrechos y anchos. Los dominios de imagen estrechos normalmente presentan una variabilidad limitada y unas características visuales mejor definidas, lo que hace la búsqueda de imágenes basadas en contenido más fácil de formular. Por otro lado, los dominios anchos (búsqueda en bases de datos de carácter general) tienden a presentar gran variabilidad e imprevisibilidad para los mismos conceptos semánticos subyacentes, lo que hace la generalización más compleja y estimulante [11].

A principios del siglo XXI, los investigadores se dieron cuenta de que las características en las que se basaban los algoritmos de búsqueda por similitud no eran tan intuitivas ni amigables como esperaban. La nueva dirección era centrarse en diseñar sistemas que fuesen más amigables para el usuario y pudieran dar el vasto conocimiento multimedia de las librerías, bases de datos y colecciones al mundo. Para hacer esto, se dieron cuenta de que la siguiente evolución de los sistemas necesitaría entender las semánticas de una consulta, no simplemente las características computacionales subyacentes de bajo nivel (el problema se llamó "puenteado de la laguna semántica"). Desde una perspectiva del reconocimiento de patrones, esto aproximadamente

significaba traducir las características fácilmente extraíbles a partir de los medios basados en contenido a conceptos de alto nivel o términos que serían intuitivos para el usuario. Seguramente el sistema de recuperación basado en contenido gráfico más temprano que abordase el problema de la laguna semántica en la interfaz de consulta, en la indexación y en los resultados fuese el motor de búsqueda de 'ImageScape' (2000) [10].

Sin embargo, todavía siguen faltando ciertas capacidades. En el contexto de la recuperación de imágenes, adquirir, almacenar y transmitir fotografías es ahora trivial, pero es significativamente duro manipularlas, indexarlas, ordenarlas, filtrarlas, resumirlas o buscar entre ellas. Los motores de búsqueda modernos y sus buscadores de imagen o vídeo descendientes han habilitado un progreso significante en dominios donde el contenido visual es etiquetado con descripciones textuales, pero sólo analizan metadatos, no las imágenes en sí, y eso los hace de uso limitado en muchos escenarios prácticos [9].

Actualmente, el problema fundamental radica en cómo habilitar o mejorar la recuperación multimedia usando métodos basados en contenido (CBIR, *content based image retrieval*). Además, estos métodos pueden mejorar potencialmente la precisión de la recuperación incluso cuando las anotaciones textuales están disponibles, dando una idea adicional dentro de las colecciones de medios [10].

El Laboratorio de cálculo visual estadístico (SVCL, *Statistical Visual Computing Laboratory*) de la Universidad de California, San Diego, ha estado considerando el problema del CBIR durante muchos años. Uno de los objetivos del SVCL es desarrollar sistemas capaces de recuperar imágenes porque las entiendan y sean capaces de representar su contenido en una forma intuitiva para los humanos. Recurriendo fuertemente a la investigación en visión artificial y aprendizaje máquina, este esfuerzo explora muchos temas en representación de imágenes y diseño de sistemas inteligentes incluyendo la evaluación de la similitud de imágenes, la anotación automática de imágenes con pies de foto descriptivos, la habilidad de entender la realimentación del usuario durante la búsqueda de imágenes (comúnmente conocida como *relevance-*

feedback o realimentación por relevancia) y el diseño de estructuras indexadas que pueden ser buscadas eficientemente.

Algunos de los sistemas de recuperación de imágenes a destacar son:

- Consulta por ejemplo semántico (QBSE, *Query by semantic example*): la idea es representar cada imagen por su vector de probabilidades a posteriori de conceptos y realizar la consulta por ejemplo en el simplex de esas probabilidades. En la Ilustración 5: "Etiquetado semántico" se puede ver un ejemplo del funcionamiento de QBSE. En la imagen A se ve el agrupamiento de imágenes por concepto semántico y aprendizaje de un modelo probabilístico de cada concepto y en la B se ve una representación de cada imagen por su vector de probabilidades a posteriori de conceptos. Como los vectores de probabilidad son distribuciones multinomiales sobre el espacio de conceptos semánticos, estos pueden referirse como multinomiales semánticos. Un sistema QBSE define una función de similitud entre esos objetos y, en respuesta a la imagen de consulta dada por el usuario, clasifica la imagen en la base de datos por la distancia de sus multinomiales semánticos a esos de la consulta [9].

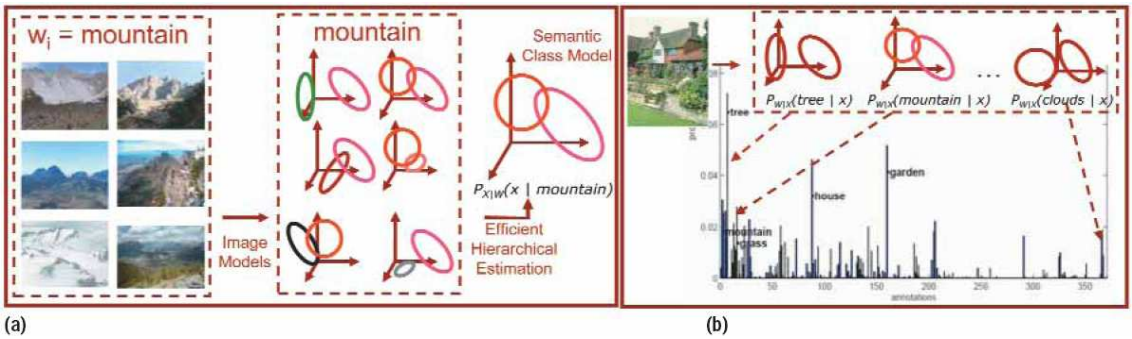

Ilustración 5: "Etiquetado semántico"

- Consulta por ejemplo visual (QBVE, *Query by visual example*): recupera imágenes usando la igualación visual estricta, imágenes de una base de datos clasificadas por similitud con una imagen de consulta dada por el usuario. El sistema extrae una firma de la consulta, compara esa firma con aquellas previamente calculadas y devuelve las igualaciones más cercanas. En la

Ilustración 6: "Cercando la laguna semántica" se ven dos imágenes. La imagen A muestra cómo la gente frecuentemente descarta indicadores visuales fuertes en sus juicios de similitud, lo que puede llevar a errores de QBVE severos, como recuperar puentes como respuesta a la consulta de tren. La imagen B muestra que la QBSE produce menos errores que QBVE, porque las buenas coincidencias requieren consenso en varias dimensiones del espacio semántico.

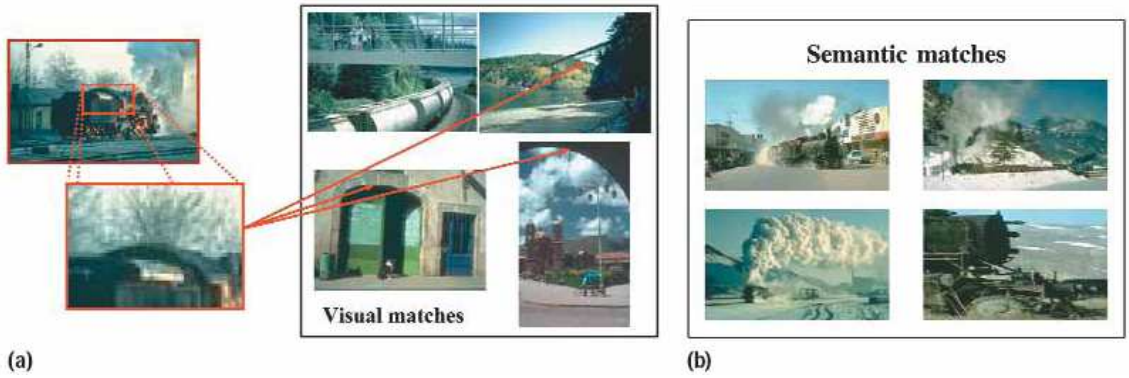

Ilustración 6: "Cercando la laguna semántica"

Las necesidades fundamentales para un sistema de recuperación multimedia son:

- Búsqueda para un medio en particular: los sistemas actuales tienen limitaciones significativas, como la incapacidad de entender un vocabulario de usuario amplio, entender el nivel de satisfacción del usuario, que no existan conjuntos de test del mundo real representativos y creíbles para evaluación ni tampoco medidas de *benchmarking* que estén claramente correladas con la satisfacción del usuario.

- Navegar y resumir una colección de medios: para facilitar la gestión de grandes bases de datos por parte de los usuarios.

Los sistemas de aprendizaje son interesantes porque permiten potencialmente al ordenador entender la colección de medios a un nivel semántico. Además, los

algoritmos de aprendizaje pueden ser capaces de adaptar y compensar el ruido y el desbarajuste en los contextos del mundo real [10].

2.2.1.1. Modalidades de consulta y procesado

En el mundo de la recuperación de imágenes, un parámetro importante para medir el nivel de interacción usuario-sistema es la complejidad de las consultas soportadas por el sistema. Desde la perspectiva del usuario, esto se traduce en las diferentes modalidades que se pueden usar para consultar un sistema. A continuación, se describen las diferentes modalidades de consulta, sus características y el soporte de sistema requerido.

- Palabras clave: en esta búsqueda, el usuario plantea una consulta simple en forma de palabra o frase. Esto es actualmente lo más popular para buscar imágenes (Google o Yahoo).

- Texto libre: cuando el usuario elabora una frase, sentencia, pregunta o historia sobre lo que desea del sistema.

- Imagen: el usuario desea buscar una imagen similar a la de la consulta (CBIR). Algunos ejemplos son Google Similar Images o Pixolu.

- Gráficos: consiste en dibujos hechos a mano o generados por ordenador para representar una consulta.

- Compuesto: métodos que envuelven más de una de las modalidades de un sistema de consulta. Esto también cubre la consulta interactiva así como los sistemas de realimentación de la relevancia.

Las modalidades nombradas requieren diferentes métodos de procesado y/o soporte para la interacción con el usuario.

- Basados en texto: normalmente se reducen a realizar búsquedas basadas en una o más palabras clave simples y después recuperar las imágenes coincidentes. El procesamiento de texto libre puede envolver análisis sintáctico, procesado y entender la consulta como un conjunto. Alguna forma del procesamiento del lenguaje natural puede también estar incluida.

- Basados en contenido: son la base de los sistemas CBIR. El procesado de una consulta (imagen o gráficos) envuelve la extracción de características visuales y/o segmentación y búsqueda en el espacio de características visuales para imágenes similares.

- Compuestos: el procesado compuesto debe envolver procesado basado en texto y en contenido en proporciones variables.

- Interactivo simple: la interacción con el usuario usando una modalidad simple es necesaria. Un ejemplo son los sistemas basados en la realimentación de la relevancia.

- Interactivo compuesto: el usuario interactuará usando más de una modalidad (por ejemplo, texto e imágenes). Ésta es probablemente la forma más avanzada de procesamiento de consulta que es requerida para ser ejecutada por un sistema de recuperación de imágenes. [11].

2.2.2. Anotación de la información

La anotación automática de contenidos, si bien resulta de sumo interés, es a veces difícil e incierta. Si se anotan las imágenes con palabras clave (*keywords*), entonces el modo típico de publicar un repositorio de imágenes es crear una interfaz de consulta basada en palabras clave para una base de datos de imágenes. Las imágenes serán recuperadas si contienen algunas de las palabras clave especificadas por el usuario. El objetivo es consultar las imágenes no sólo basándose en el conjunto, sino en los objetos individuales que aparecen en las imágenes.

Dado un conjunto de imágenes manualmente anotadas donde para cada imagen se asocian un conjunto de palabras clave que describen su contenido, los investigadores ya han propuesto varios algoritmos para determinar la correlación entre ellas y las señales de las imágenes (Mori99 [28], Jeon03 [29], Duygulu02 [30], Barnard03 [31], Blei03 [32], Li03 [33]). Una vez determinada esa correlación, puede ser usada para anotar imágenes no anotadas. Cada imagen será representada por un conjunto de palabras clave y señales visuales. Es posible que la misma señal visual pueda ser compartida por más de una imagen. Como el concepto de similitud de las señales visuales está tan mal definido en comparación con las palabras clave, las señales visuales serán agrupadas juntas y se generará un conjunto finito. La premisa es que si alguna de las señales visuales son las mismas, pertenecerán al mismo clúster. Por lo tanto, para determinar la correlación entre palabras clave asociadas y señales visuales, se necesitan abordar los siguientes problemas:

- Agrupar señales visuales similares para construir señales patrón.
- Analizar la correlación entre palabras clave y señales patrón

En la Ilustración 7: "Demostración de correspondencia entre los objetos de la imagen y sus palabras clave" se muestra un ejemplo visual de esta correspondencia.

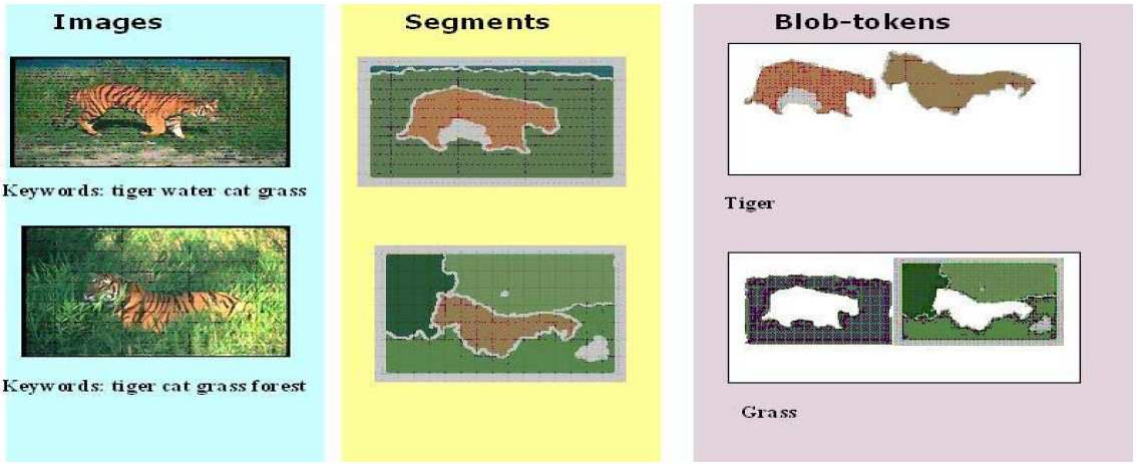

Ilustración 7: "Demostración de correspondencia entre los objetos de la imagen y sus palabras clave"

Para construir señales patrón, el actual estado del arte utiliza el algoritmo de *clustering* tradicional (por ejemplo, K-*means*). Cada señal visual está compuesta normalmente de datos de alta dimensión y los algoritmos de *clustering* normales asignan pesos equitativos a todas las dimensiones. Debido al problema de la dimensionalidad, los datos se vuelven dispersos y las medidas de distancia se vuelven gradualmente insignificantes según aumenta el número de dimensiones. Esto degradará la calidad del resultado para algoritmos de agrupamiento tradicionales. Para resolver ese problema, algunas características podrían ser más relevantes que otras para un conjunto de señales visuales [13].

El etiquetado de imágenes semántico puede ser un problema de aprendizaje supervisado o no supervisado. Los últimos esfuerzos en el área fueron dirigidos a la extracción fiable de semánticas específicas, por ejemplo: diferenciar interiores de exteriores [23], ciudades de paisajes [24], y detectar árboles [25], caballos [26] o edificios [27] entre otros. Estos trabajos transforman el problema de la extracción de las semánticas como uno de aprendizaje supervisado: un conjunto de imágenes de entrenamiento en los que el concepto de interés puede o no estar presente se emplea para entrenar un clasificador binario que permita detectar la presencia del concepto. El clasificador después se aplica a todas las imágenes de la base de datos que han sido anotadas con respecto a la presencia o ausencia del concepto.

Más recientemente ha habido un esfuerzo para resolver el problema de una forma general recurriendo al aprendizaje no supervisado ([34], [35], [36], [37], [38], [39], [40], [41]). Durante el entrenamiento, un conjunto de etiquetas es asignado a cada imagen, la imagen es segmentada en una colección de regiones, y un algoritmo de aprendizaje sin supervisión se ejecuta sobre la base de datos entera para estimar la densidad conjunta de las etiquetas semánticas y las características visuales. Dada una nueva imagen para anotar, los vectores de características visuales son extraídos, el modelo de probabilidad conjunta es instanciado con esos vectores de características, las variables de estado son marginadas, y se lleva a cabo una búsqueda para el conjunto de etiquetas que maximice la densidad conjunta del texto y la apariencia.

Ambas formulaciones tienen fuertes ventajas e inconvenientes. En términos genéricos, el etiquetado sin supervisión lleva a procedimientos de entrenamiento más escalables y produce un ranking natural de etiquetas semánticas para cada nueva imagen para anotar. Por otro lado, no trata explícitamente las semánticas como clases de imagen y, por tanto, provee pocas garantías de que las anotaciones semánticas sean óptimas en un reconocimiento o sentido de recuperación. Esto es, en vez de anotaciones que consigan la probabilidad más pequeña de error de recuperación, simplemente produce las que tengan mayor verosimilitud conjunta asumiendo el modelo mixto. Además, debido a las dificultades de la inferencia conjunta en conjuntos de variables aleatorias continuas y discretas, el aprendizaje sin supervisión normalmente requiere independencia restrictiva supuesta en la relación entre el texto y los componentes visuales de los datos de imagen anotados [14].

En definitiva, a partir de las referencias encontradas en la literatura, se podrían considerar tres modelos principales en la anotación de imágenes: anotación totalmente manual, automática y semi-automática. La anotación manual es un trabajo tedioso, necesita de esfuerzo humano al anotar cada imagen. La anotación automática se lleva a cabo sin la ayuda del hombre (métodos no supervisados) pero, dadas las limitaciones presentes en las técnicas de visión artificial y el procesado de imagen, producen conjuntos de anotaciones de escasa precisión. El modo semi-automático divide la base de datos de la imagen en dos partes, una para entrenamiento y otra para validación. El conjunto de entrenamiento es etiquetado manualmente de modo que la relación entre

imágenes y palabras es precisa. A continuación, se emplean las relaciones aprendidas entre las imágenes y las palabras para generar etiquetas en el conjunto de validación.

Atendiendo a la granularidad en el procesado de las imágenes, se podrían distinguir además tres métodos principales para la anotación de imágenes: anotación basada en la segmentación de imágenes, bloqueo de tamaño fijo y anotación basada en clasificación de imágenes.

- La anotación basada en la segmentación de imágenes depende de las características visuales de las imágenes y de los resultados precisos de la segmentación. En condiciones ideales, cada región segmentada corresponde a un objeto. Sin embargo, el resultado de la segmentación de imágenes no es satisfactorio en el presente. Por eso existe una gran diferencia entre la expresión de la parte del nivel de objetos de imágenes y el sistema de visión humana.
- En la división de imágenes en tamaño fijo (características extraídas en un *grid* o rejilla) también existe este problema, ya que podrá dividir un objeto en muchos bloques o poner muchos objetos en un bloque.
- La anotación de imágenes basada en la clasificación de imágenes puede evitar la baja precisión causada por la errónea división de imágenes, comparado con los otros dos métodos [16].

Por último, atendiendo a la naturaleza de los clasificadores empleados los métodos de anotación de imágenes pueden ser catalogados en dos categorías:

- Métodos discriminativos: tratan las palabras clave como clases y emplean clasificadores entrenados para obtener fronteras que permitan discriminar entre aquellas imágenes en las que aparece un concepto y las que no. Métodos discriminativos clásicos pueden ser las redes neuronales o las máquinas de vector soporte.

- Métodos generativos: a diferencia de los anteriores, que únicamente se preocupaban de discriminar entre casos positivos y negativos, éstos tratan de inferir las probabilidades conjuntas entre imágenes y anotaciones. Dicha

información, si bien más compleja de obtener, proporciona un conocimiento extra sobre la generación de los datos. Como trabajo pionero, Mori *et al.* [28] propuso un método para anotar imágenes cuadriculadas usando co-ocurrencias en 1999. Duygulu *et al.* [37] propuso una aproximación novel que trata la anotación de imágenes como un problema de traducción máquina. Un modelo de traducción máquina estadística fue usado para traducir palabras clave textuales a palabras clave visuales, por ejemplo señales patrón de imágenes obtenidas por agrupamiento. Otro modo de capturar información de co-ocurrencia es introducir variables latentes asociadas a los conceptos semánticos de las etiquetas. Algunos ejemplos de este tipo de trabajos son los modelos de Análisis semántico latente probabilístico PLSA (*Probabilistic Latent Semantic Analysis*) [21] y el de Asignación de Dirichlet latente LDA (*Latent Dirichlet Allocation*) [22]. Inspirados por los modelos de lenguaje de relevancia, varios modelos de relevancia han sido propuestos recientemente, incluyendo el modelo de relevancia cross-media (*Cross-Media Relevance Model,* CMRM) [29], el modelo continuo de relevancia (*Continuous Relevance Model* CRM) [29] y el modelo de relevancia de Bernoulli múltiple (*Multiple Bernoulli Relevant Model* MBRM) [38].

A pesar de los continuos esfuerzos empleados en la anotación de imágenes, los resultados obtenidos son todavía insatisfactorios, máxime cuando el dominio de estudio crece de forma considerable (número de imágenes y de categorías). Alternativamente, sería ventajoso si una aproximación dedicada pudiera refinar los resultados de anotación actuales [18].

2.3. Sistemas de clasificación de patrones

En el reconocimiento de patrones, un objeto es clasificado en una de las categorías (llamadas clases) usando características que discriminen bien las clases [3]. A través de una observación y un sistema de medición, se obtienen un conjunto de números que conforman el vector de medición.

El diseño del clasificador consta de dos partes. El entrenamiento o aprendizaje recoge las muestras de datos etiquetados y encuentra las fronteras que separan las clases. Dado que, en la gran mayoría de los casos, los clasificadores presentan ciertos parámetros libres para los que, aunque se deban ajustar, no existe ninguna forma automática de optimización, se hace necesaria una segunda fase de validación. En esta fase, cada clasificador entrenado con una configuración diferente se valida sobre un nuevo conjunto de muestras etiquetadas con el objetivo de evaluar su funcionamiento y elegir el valor de los parámetros libres [7].

De una forma más detallada, un sistema de clasificación pasa por las siguientes fases [3]:

- Determinación de características: primero se determinará un conjunto de características apropiado. Como es relativamente fácil detectar y eliminar posteriormente aquellas características que se tornen redundantes o poco discriminativas para el problema y, por otro lado, la no inclusión de características necesarias sí sería crítica para el sistema, se establece un número suficiente de características.

- Normalización de los datos: para asegurar un correcto funcionamiento del clasificador, las muestras han de normalizarse a un rango limitado (por ejemplo, el intervalo [0,1]).

- Optimización de características: se puede hacer mediante selección de características o mediante extracción de características.

- Extracción de características: las características originales se reducen en un número pequeño de características por transformaciones lineales o no lineales.
- Selección de características: se analiza si las muestras agrupadas o las características extraídas se pueden clasificar correctamente.

- División de datos: se dividen las muestras en los conjuntos necesarios, a saber: conjunto de entrenamiento, conjunto de validación, y conjunto de test. Las características de los tres conjuntos deben ser similares. Normalmente, el conjunto de muestras se separa de manera que un 60 u 80% pertenezca al subconjunto de entrenamiento y un 40 o 20% pertenezca al subconjunto de test. En caso de ser necesario un conjunto de validación, se podrían generar subconjuntos a partir del conjunto de entrenamiento.

- Ajuste de parámetros libres: se selecciona una posible configuración del clasificador, se entrena sobre el conjunto de entrenamiento y se evalúa con el conjunto de validación con el fin de decidir la configuración óptima.

- Evaluación del clasificador: una vez ajustados los parámetros libres, se selecciona el mejor clasificador en validación, se entrena con el conjunto de entrenamiento y se evalúa con el conjunto de test.

2.4. Características de bajo nivel

Las características de bajo nivel son los rasgos que pueden ser directamente extraídos a partir de las representaciones digitales de los objetos. Pueden estar muy lejos de la metodología de interpretación que los seres vivos asocian con la percepción del objeto. Algunos ejemplos de estas características se listan a continuación:

- Estadísticas de tonos y colores (medias, histogramas…)
- Descriptores de Fourier (DFT)
- Descriptores relacionados con la geometría fractal.
- Parámetros de forma (áreas, diámetros, perímetros)
- Parámetros geométricos (líneas, curvas y contornos)
- Parámetros extraídos de la imagen binaria.
- Descriptores de los modelos de textura.
- Descriptores de movimiento (para secuencias audiovisuales)

Las características de bajo nivel se pueden extraer con diversa granularidad en una imagen. En particular, se pueden distinguir varios niveles:

- Análisis orientado a la escena global. Aplicación de anotaciones de escenas básicas obtenidas mediante filtros globales o utilización de descriptores locales correspondientes a las regiones relevantes de la imagen.

- Segmentación de la imagen en varias regiones: se aplican algoritmos de segmentación de imágenes, para extraer características locales sobre dichas regiones (forma de la región, color, textura).

- Aplicación de rejillas para obtener características locales: se divide la imagen en rejillas (*grid*) de tamaño fijo para luego concatenar las características extraídas en cada una de las rejillas. Esta aproximación introduce cierta discriminación espacial (encontrar rejillas de color azul únicamente en la parte superior de la imagen, por ejemplo, puede ser indicativo del concepto cielo,

mientras que si únicamente están en la parte inferior pueden corresponderse con el concepto agua).

- Características sobre parches locales: se localizan puntos clave (*keypoints*) en la imagen que luego serán descritos [45]. Esta aproximación se describirá en detalle en la siguiente sección.

2.5. Características sobre parches locales: la transformada SIFT

SIFT (*Scale Invariant Feature Transform*) es un algoritmo de visión artificial que permite detectar y, posteriormente, describir características en regiones locales de una imagen de una forma invariante a escala. Para generar un conjunto de características sobre una imagen el algoritmo utiliza:

- Detección de extremos en el espacio de escala: en el primer paso el algoritmo busca regiones de interés sobre todas las localizaciones de las imágenes a diferentes escalas. Es implementado eficientemente usando una función de diferencia de Gaussianas para identificar puntos de interés potenciales que son invariantes a la escala y a la orientación. En la Ilustración 8: "Representación del proceso que sigue cada octava del espacio escala", se muestra cómo para cada octava del espacio escala, la imagen inicial es repetidamente convolucionada con Gaussianas para producir el conjunto de imágenes espacio escala mostrado en la izquierda. Las imágenes Gaussianas adyacentes son substraídas para producir las imágenes diferencia de Gaussiana (DoG) de la derecha. Después de cada octava, la imagen Gaussiana es submuestreada por un factor de 2, y el proceso es repetido. El objetivo de la detección es encontrar extremos en el espacio DoG, que se corresponderán con puntos de interés (*keypoints*).

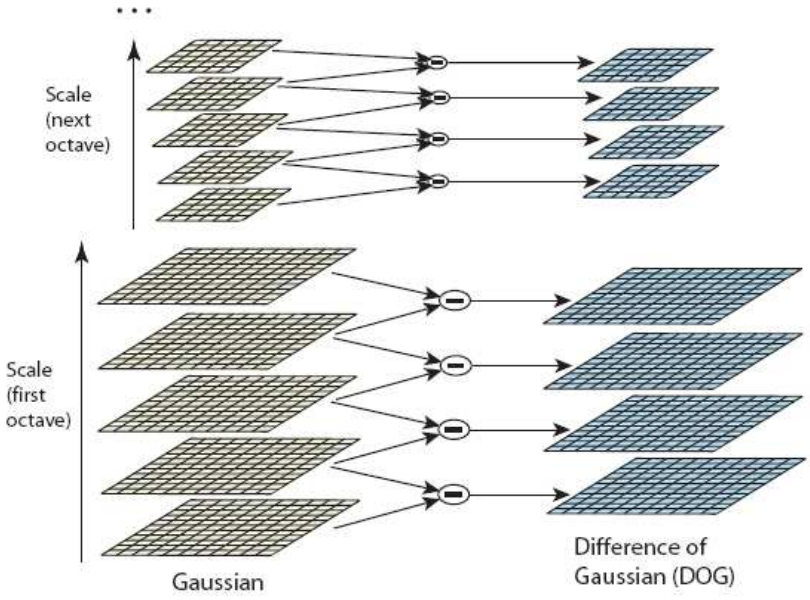

Ilustración 8: "Representación del proceso que sigue cada octava del espacio escala"

- Localización de puntos clave: a cada localización candidata, un modelo detallado se ajusta para determinar la localización y la escala. Los puntos clave son seleccionados basados en medidas de su estabilidad. En la Ilustración 9: "Fases de selección de puntos clave" se muestran, a) la imagen original, b) los 832 puntos clave originales al máximo y mínimo de la función Diferencia de Gaussianas, c) 729 puntos clave restantes tras aplicar un umbral con un mínimo contraste, y d) los 536 puntos clave finales que quedan siguiendo un umbral adicional en proporción a las principales curvaturas.

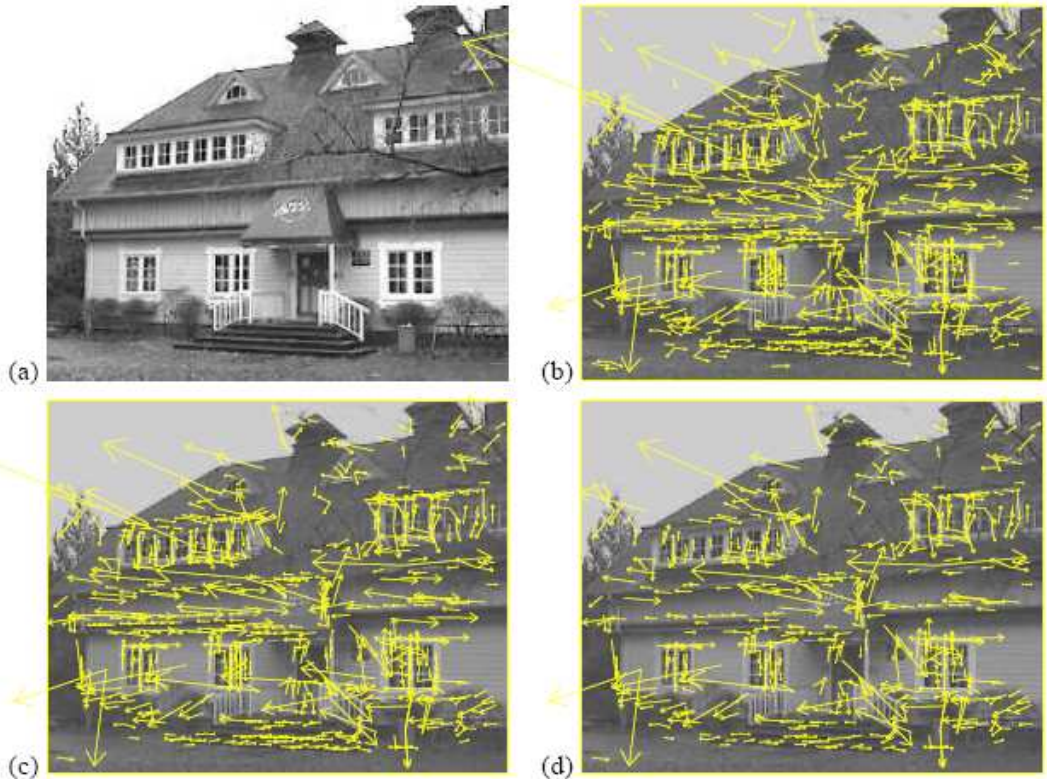

Ilustración 9: "Fases de selección de puntos clave"

- Asignación de la orientación: a cada localización de puntos clave se le asigna una o más orientaciones basadas en direcciones de gradientes de imagen local. Todas las operaciones futuras son realizadas en los datos de imagen que han sido transformados en relación a la orientación asignada, escala y localización para cada característica, proporcionando así invarianza a dichas transformaciones. En la Ilustración 10: "Gráfico Repetitividad-Ruido de imagen" se muestran tres líneas: la primera representa el porcentaje de localizaciones de puntos clave y escalas que son detectados repetidamente como una función de ruido de píxel, la segunda línea muestra la repetitividad después de también requerir un acuerdo en orientación, y la última línea indica el porcentaje final de descriptores correspondidos correctamente en una gran base de datos.

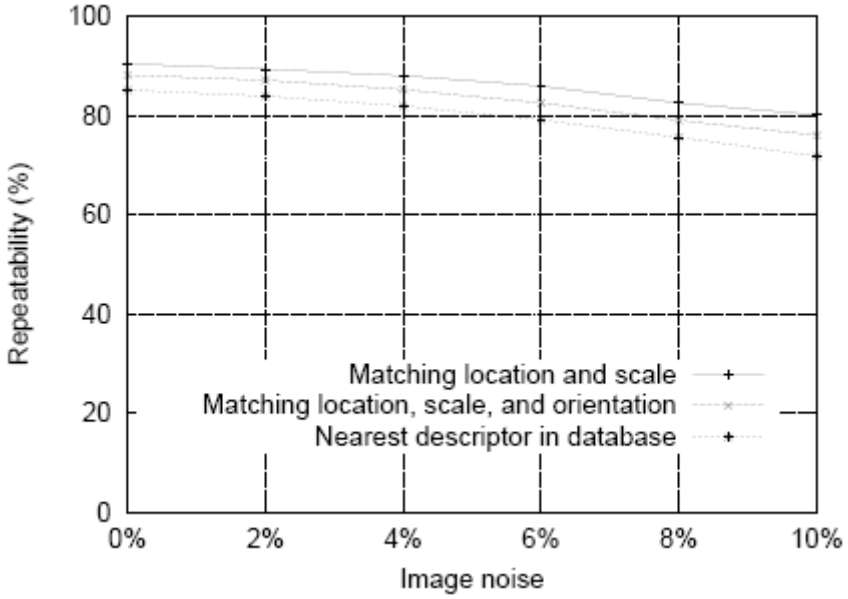

Ilustración 10: "Gráfico Repetitividad-Ruido de imagen"

- Descriptor de puntos clave (*keypoints*): los gradientes de imagen locales son medidos a la escala seleccionada en la región alrededor de cada punto clave. Estos son transformados en una representación que tiene en cuenta significativos niveles de distorsión de forma local así como cambios en la iluminación. En la Ilustración 11: "Gradientes de la imagen y descriptor de puntos clave" se puede ver cómo un descriptor de puntos clave es creado calculando la transformada SIFT. Inicialmente, como se ve en la imagen de la izquierda, se calcula la magnitud del gradiente y la orientación de cada punto en una región alrededor de la localización del punto clave. A estos se les asigna un peso con una ventana Gaussiana, indicado por un círculo superpuesto (según el peso asignado, el círculo tendrá un radio mayor o menor). Estas muestras son después acumuladas en histogramas de orientaciones, resumiendo los contenidos sobre subregiones resultantes de una división de la región en un grid 4x4, como se muestra en la imagen derecha. La longitud de cada flecha corresponde a la suma de las magnitudes de gradiente en dicha dirección dentro de la región. Esta figura muestra un array de descriptores 2x2 calculado en la subregión correspondiente.

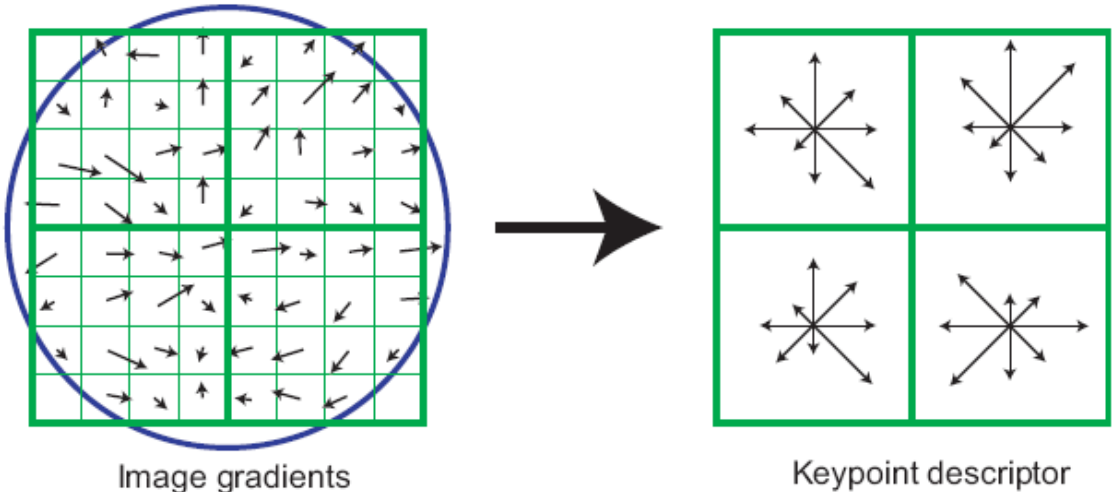

Ilustración 11: "Gradientes de la imagen y descriptor de puntos clave"

Esta aproximación ha sido llamada la transformación de características invariante a la escala (SIFT), y transforma los datos de la imagen en coordenadas invariantes a la escala relativas a las características locales.

Un aspecto importante de esta aproximación es que genera un gran número de características que cubre densamente la imagen sobre el rango completo de escalas y localizaciones. Una imagen típica de 500x500 píxeles de tamaño ocasionará unas 2000 características estables (aunque ese número depende tanto del contenido de la imagen como de las elecciones de varios parámetros). La cantidad de parámetros es particularmente importante para el reconocimiento de objetos, donde la habilidad para detectar objetos pequeños en fondos abarrotados requiere que al menos 3 características sean correctamente correspondidas para cada objeto para una identificación fiable.

Para correspondencia y reconocimiento de imágenes, las características SIFT son primero extraídas de un conjunto de imágenes de referencia y almacenadas en una base de datos. Una nueva imagen es correspondida individualmente comparando cada característica suya con esa base de datos previa y encontrando un candidato igualando las características basadas en la distancia Euclídea de sus vectores de características. [1].

2.6. El modelo *bag-of-words*

El modelo *bag-of-words* (bolsa de palabras, BoW) en el procesado de lenguaje natural es un método popular para representar documentos, que ignora el orden de las palabras. El modelo BoW permite un modelado basado en diccionario, y cada documento parece una bolsa (de ese modo no se considera el orden), que contiene algunas palabras del diccionario. En visión artificial se utiliza una idea similar para representación de imágenes (una imagen se refiere a un objeto particular, como una imagen de un coche). Por ejemplo, una imagen puede ser tratada como un documento, y las características extraídas en ciertas regiones o puntos de la imagen son consideradas palabras visuales (normalmente se necesitan algunas manipulaciones).

La implantación del modelo BoW en la visión artificial requiere los siguientes pasos: detección de características, descripción de características y generación de *codebook*.

2.6.1. Detección de características locales

Dada una imagen, la detección de características es la extracción de parches o regiones que son considerados los elementos básicos del análisis.

2.6.1.1. Cuadrícula regular

La cuadrícula regular es probablemente el método más simple y efectivo para detección de características. En este método, como se puede ver en la Ilustración 12: "Imagen a la que se le aplica una cuadrícula regular", la imagen es uniformemente segmentada por líneas horizontales y verticales, obteniendo así parches locales. Este método muestra resultados muy prometedores para la categorización natural de escenas. Su limitación es que emplea poca información de la imagen en sí.

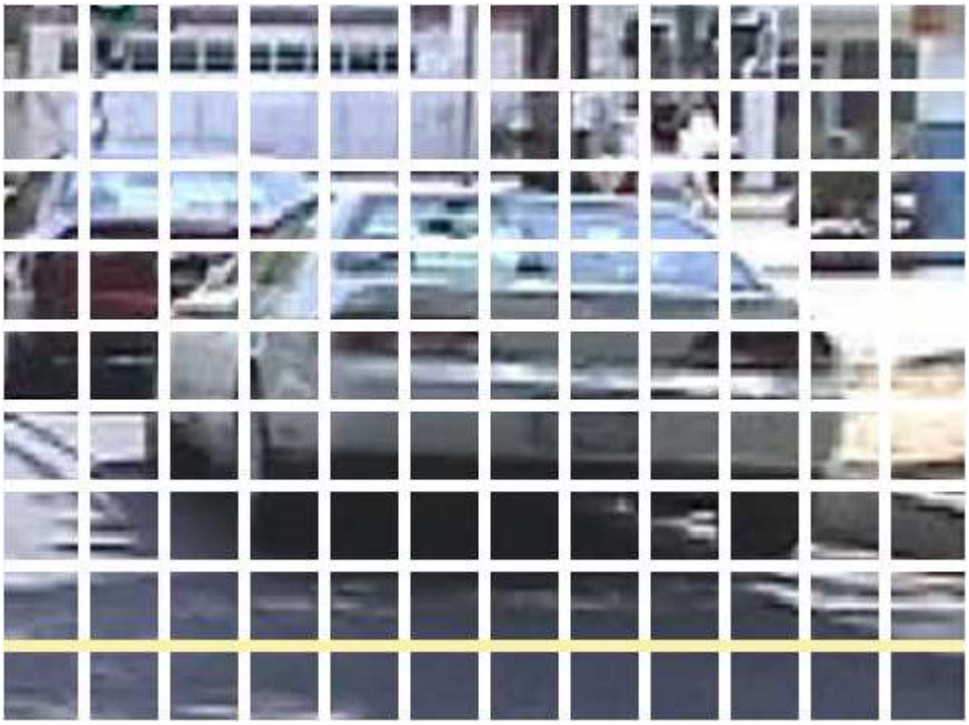

Ilustración 12: "Imagen a la que se le aplica una cuadrícula regular"

2.6.1.2. Detector de puntos de interés

Los detectores de puntos de interés tratan de detectar parches principales como bordes, esquinas y manchas en la imagen. Éstos son considerados más importantes que otros parches, como regiones que atraigan más la atención humana, que serán más útiles para la categorización de objetos. Se puede ver un ejemplo en la Ilustración 13: "Imagen a la que se le aplica la detección de puntos de interés". Algunos detectores famosos son el detector de esquinas de Harris [42], el detector de DoG (diferencia de gaussiana) de Lowe [43] y el detector de notabilidad de Kadir Brady [44].

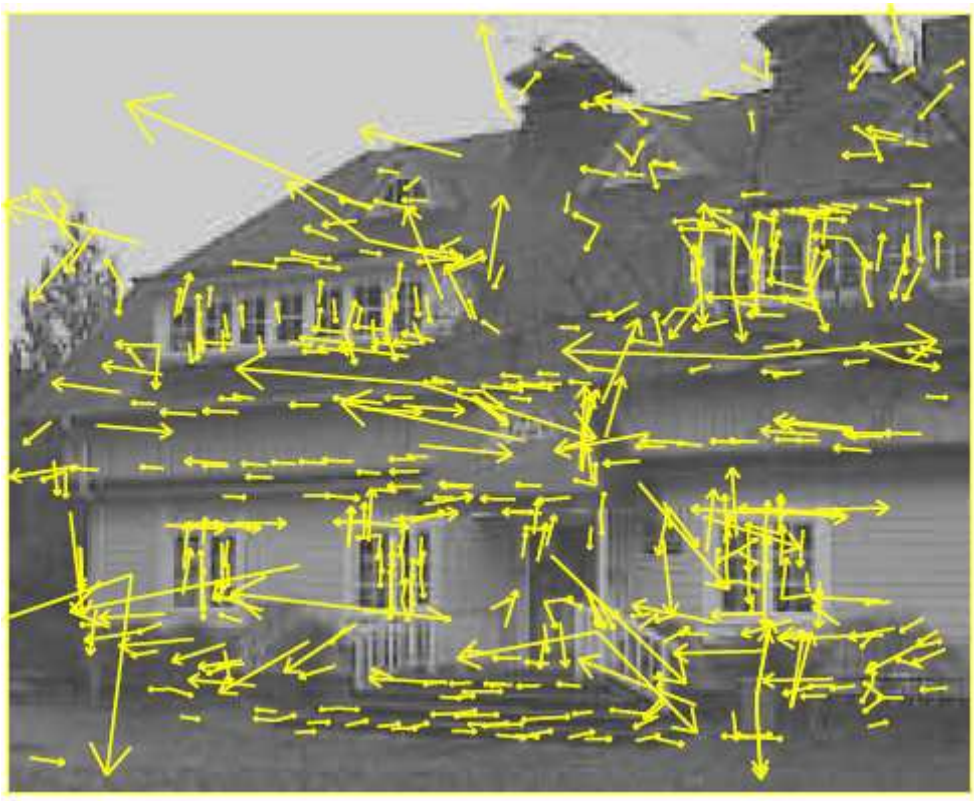

Ilustración 13: "Imagen a la que se le aplica la detección de puntos de interés"

2.6.1.3. Otros métodos

Además, también se emplea muestreo aleatorio y modelos de segmentación para la detección de características.

2.6.2. Representación de características

Después de la detección de características, cada imagen es representada a través de sus parches locales. Los métodos de representación tratan de describir los parches como vectores numéricos, llamados descriptores de características. Un buen descriptor debe tener la habilidad de manejar la intensidad, rotación, escala y variaciones afines de la misma dimensión (128 para SIFT por ejemplo), cuando el orden de los diferentes vectores no importa.

2.6.3. Generación de *codebooks* y asignación de *codewords*

Una vez todas las regiones de interés de las imágenes han sido descritas, se genera un vocabulario (*codebook*) representativo de las características que aparecen en los datos. Para ello, a través de métodos de *clustering* (no supervisados) se organizan los datos en ciertos clusters que se corresponden con palabras visuales (*codewords*) de dicho vocabulario. Cada palabra visual, por lo tanto, constituirá la caracterización (ya sea a través del color, textura u otra descripción) de un patrón visual.

Posteriormente, cada descripción asociada a un punto de interés (keypoint) en una imagen se proyectará sobre el vocabulario asignándosele la palabra más parecida.

En la práctica, para generar *codebooks* se emplean algoritmos de *clustering* bien conocidos y simples, como el algoritmo k-*means*, dada la gran cantidad de datos y la elevada dimensión de los mismos.

La generación de *codebooks* tiene un doble objetivo: por un lado permite reducir la dimensionalidad de los datos de entrada (128 en el caso de emplear descriptores SIFT, por ejemplo) al asignar cada descriptor a un único *codeword* y, en conjunción con el empleo de otras técnicas como la creación de histogramas normalizados o los modelos generativos de *bag-of-words*, permite clasificar imágenes indexadas a través de un número variable de descriptores locales (al ser variable este número, la mera concatenación de descriptores no es posible). Por otro lado, generar clasificadores a nivel de *keypoint* no es factible, pues el etiquetado de imágenes se hace a nivel global, no formando parte todos los *keypoints* del objeto de interés (el hecho de que una imagen se catalogue como coche no implica que todos los *keypoints* pertenezcan al coche).

2.6.4. Representación de las imágenes en el modelo BoW: el histograma normalizado de *codeword*

Una vez que los descriptores asociados a cada región de interés de la imagen han sido asignados al *codeword* más cercano resulta necesario generar un vector de entrada a nivel de imagen que la permita categorizar a través de algoritmos de aprendizaje máquina.

Si bien existen multitud de técnicas para generar la entrada a nivel de imagen, la más común es el histograma normalizado de *codewords*. Para generarlo, se contabiliza la aparición de las distintas palabras del vocabulario a lo largo de la imagen y se normaliza finalmente entre el número total de palabras encontradas. Así, con independencia del número de palabras presentes en cada imagen, todos los histogramas tendrán la misma longitud (el tamaño del *codebook*) y cumplirán que la suma de los valores de todas sus barras será igual a 1.

Como se había comentado con anterioridad, el empleo de esta técnica en el modelo BoW permite generar entradas a nivel de imagen con una longitud fija, independientemente del número de puntos de interés encontrados.

2.7. Agrupamiento o *clustering*. Algoritmo *K-means*

El agrupamiento consiste en dividir, de una forma no supervisada, un conjunto de datos sin etiquetar en muchos subconjuntos de acuerdo con la distribución de los datos. El agrupamiento por lotes agrupa los datos reunidos mediante cálculos iterativos [3].

El análisis de clusters es una herramienta de análisis de datos exploratorios que intenta evaluar la interacción entre patrones en grupos o clusters, de tal manera que patrones del mismo cluster son más similares que otros pertenecientes a otros clusters. Los resultados pueden ser usados para establecer hipótesis sobre los datos, para clasificar nuevos datos, para validar la homogeneidad y para comprimir los datos [7].

El algoritmo de agrupamiento k-*means* está basado en un criterio de agrupamiento conciso. La idea principal es definir *k* centroides, uno por cada cluster. Esos centroides deben ser situados inicialmente de un modo ingenioso porque las localizaciones diferentes causan resultados diferentes. Por eso, la mejor opción es situarlos lo más lejano posible cada uno del otro. El siguiente paso es tomar cada punto perteneciente a un conjunto de datos dado y asociarlos al centroide más cercano. Cuando no queda ningún punto por asociar, el primer paso es completado y un agrupamiento temprano ya está hecho. En este punto se necesita recalcular *k* nuevos centroides como baricentros de los clusters resultantes del paso previo. Después de tener esos *k* nuevos centroides, se genera un bucle. Como resultado de ese bucle se nota que los *k* centroides cambian su localización paso a paso hasta que no se hacen más cambios. Finalmente, este algoritmo requiere minimizar una función objetivo, la función de error cuadrático medio.

En otras palabras, el algoritmo divide los datos **x** incluidos en un conjunto *X* en *k* conjuntos de clústeres S_1, S_2,…, S_k minimizando la distancia euclídea entre **x** y la media de los datos pertenecientes a S_i. En este método los datos que están cerca de la media son agrupados al clúster *i* [3].

2.8. Aprendizaje máquina para clasificación

El aprendizaje máquina trata el diseño y desarrollo de algoritmos que permite a los ordenadores mejorar su rendimiento a la hora de analizar datos procedentes de diversas fuentes, como los de un sensor o los de una base de datos. Un mayor enfoque en la investigación del aprendizaje máquina produce modelos, reglas y patrones de los datos. Como los conjuntos de entrenamiento son finitos, la teoría de aprendizaje normalmente no da garantías absolutas en el rendimiento de los algoritmos

En este proyecto se van a utilizar dos tipos de algoritmos de aprendizaje máquina: las Redes Neuronales Artificiales (ANN) y las Máquinas de Vectores Soporte (SVM) [5].

2.8.1. Clasificadores de redes neuronales

Las redes neuronales artificiales son modelos simplificados de las redes neuronales biológicas. Tratan de extraer las excelentes capacidades del cerebro para resolver ciertos problemas complejos. Una red neuronal artificial es un procesador paralelo distribuido y masivamente interconectado que almacena conocimiento experimental. Las redes neuronales artificiales presentan las siguientes características:

- El conocimiento es adquirido experimentalmente.
- Los pesos (ganancias) de interconexión (sinapsis) varían constantemente.

Las redes neuronales artificiales ofrecen estas ventajas:

- No linealidad: el procesador neuronal es básicamente no lineal y, por consecuencia, la red neuronal también.

- Transformación entrada-salida: el proceso de aprendizaje consiste básicamente en presentar a la red un ejemplo y modificar sus pesos sinápticos

de acuerdo con su respuesta. Aprende, por tanto, una transformación entrada/salida.

- Adaptatividad: la red tiene capacidad de adaptar sus parámetros, aun en tiempo real.

- Tolerancia a fallos: debido a la interconexión masiva, el fallo de un procesador no altera seriamente la operación.

- Uniformidad en el análisis y diseño: esto permite garantizar características precisas.

- Analogía con las redes biológicas: esto permite la utilización mutua del conocimiento de las dos áreas.

La neurona es la unidad de proceso de información fundamental en una red neuronal.

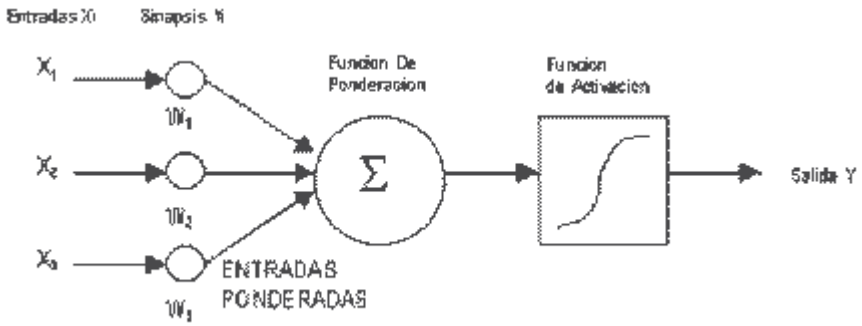

Ilustración 14: "Esquema de una red neuronal con 3 entradas"

Como se puede observar en el ejemplo contenido en la Ilustración 14: "Esquema de una red neuronal con 3 entradas", en el modelo se identifican cuatro elementos:

- Enlaces de conexión (sinapsis): parametrizados por pesos sinápticos w_{nj} (n corresponde a la neurona receptora y j corresponde a la neurona emisora) [6]. Un peso positivo trabaja para activar la neurona conectada (conexión

excitadora), y un peso negativo para desactivar la neurona conectada (conexión inhibidora) [3].

- Función de ponderación o sumador: suma los componentes de las señales de entrada multiplicadas por $w_{nj.}$

- Función de activación o de salida de las neuronas: en general, es una transformación no lineal de los datos [6]. Es modelada por una función no lineal *f (u)* con saturación, donde *u* es la entrada a la neurona [3]. Cumple con el objetivo de limitar el rango de salida de la neurona y puede ser lineal o no lineal. Se selecciona de acuerdo con el problema y a criterio del investigador, en ocasiones por ensayo y error, también depende de la precisión y velocidad requerida y del algoritmo de aprendizaje escogido. Las funciones de activación más utilizadas son:

    - Función de activación lineal: se utiliza en distintos tipos de redes, frecuentemente en la capa de salida (las funciones no lineales en la neurona de salida son comúnmente utilizadas en tareas de clasificación de patrones para restringir los valores de salida a ciertos rangos). Se observa un ejemplo en la Ilustración 15: "Función de activación lineal".

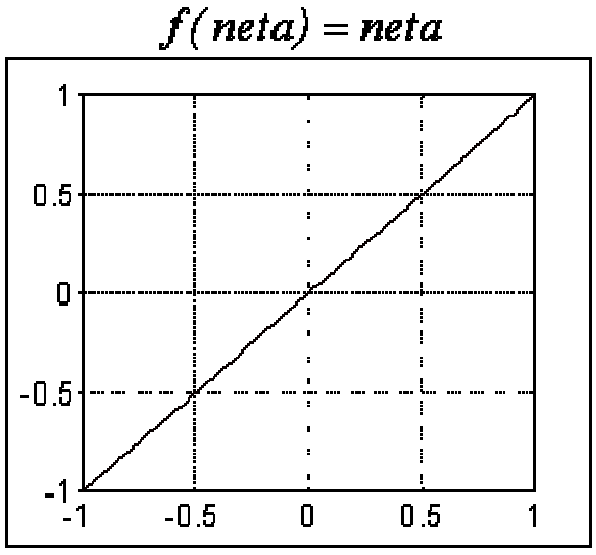

Ilustración 15: "Función de activación lineal"

- Función de activación sigmoidal: es la más comúnmente utilizada por sus características de derivación y se recomienda para problemas de predicción (aunque en el algoritmo de retropropagación la función más comúnmente utilizada es ésta, puede trabajar con cualquier otra función de activación que sea diferenciable). En la Ilustración 16: "Función de activación sigmoidal" se puede observar la forma de la función.

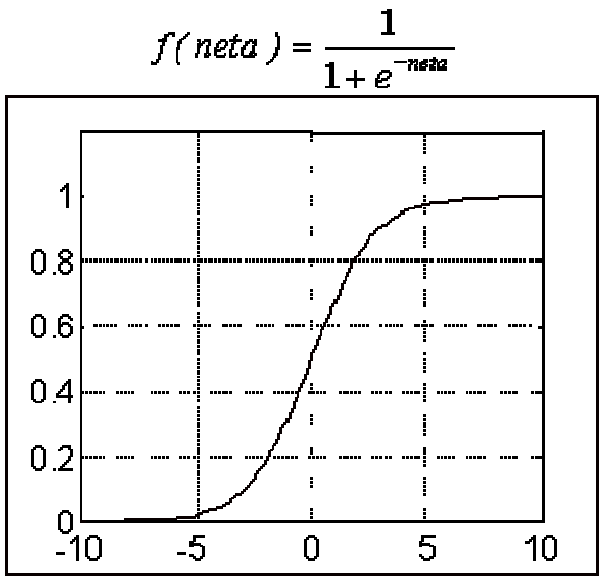

Ilustración 16: "Función de activación sigmoidal"

- Función de activación sigmoidal bipolar o tangente sigmoidal: es similar a la sigmoidal y se utiliza con frecuencia en redes multicapa. La Ilustración 17: "Función de activación sigmoidal bipolar o tangente sigmoidal" es un ejemplo de esta función.

$$f(neta) = \frac{2}{1+e^{-neta}} - 1$$

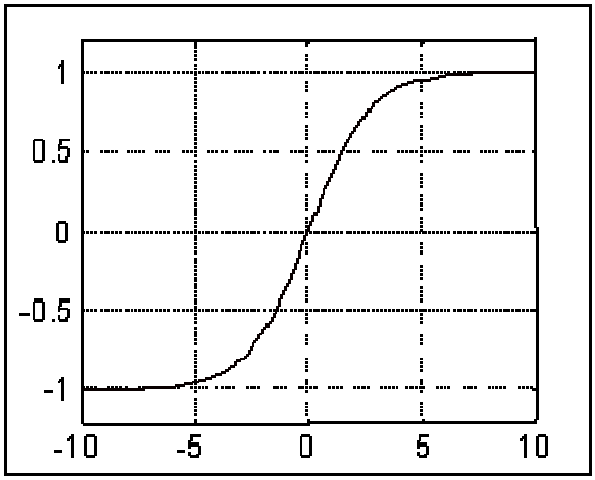

Ilustración 17: "Función de activación sigmoidal bipolar o tangente sigmoidal"

- Función escalón: para problemas de clasificación. Se puede ver un ejemplo en la Ilustración 18: "Función escalón".

$$f(neta) = \begin{cases} 1 & neta \geq 0 \\ 0 & neta < 0 \end{cases}$$

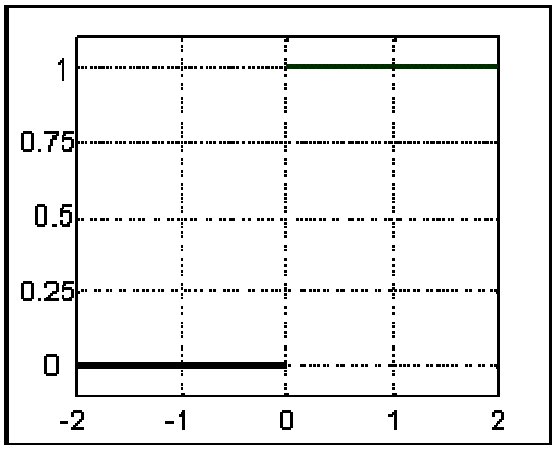

Ilustración 18: "Función escalón"

Una buena función de activación debería cumplir: primero, que ella misma y su derivada sean fáciles de computar y segundo, que la función debe tener una amplia parte lineal para lograr velocidad de entrenamiento y convergencia en pocos ciclos [17].

- Umbral: desplaza la entrada [6].

Los modelos de redes neuronales son ajustables a través de la elección de la función de activación y las conexiones de red. La determinación de los pesos se llama entrenamiento o aprendizaje [3].

El desempeño de una red neuronal depende del algoritmo de aprendizaje utilizado, del número de capas ocultas, del número de neuronas en cada capa oculta, de la conectividad o arquitectura de red y también del tipo de función de activación que se elija para cada neurona. En la actividad de entrenamiento, el algoritmo de aprendizaje aplica un procedimiento definido que modifica los pesos o ponderaciones de interconexión de la red, ajustándolos de tal forma que cada vez se obtenga un mejor resultado de la red con respecto a los datos de prueba.

Estos algoritmos pueden ser supervisados o no supervisados. Dentro de los supervisados se encuentra el algoritmo de retropropagación (*backpropagation*) o del gradiente descendente, creado como generalización del algoritmo para perceptrón continuo, cuando se tienen múltiples capas (redes neuronales perceptrón multicapa) y popularmente utilizado para la solución de problemas de clasificación y pronóstico, así como para aplicaciones de reconocimiento de patrones, procesamiento de señales, comprensión de datos y control automático. Este método es a menudo demasiado lento para problemas prácticos por lo que se han propuesto variantes de éste de mayor desempeño que pueden converger desde diez a cientos de veces más rápido utilizando técnicas heurísticas, como el método elástico de retropropagación (*resilient backpropagation* o Rprop), de muy rápida convergencia para problemas de reconocimiento de patrones.

La arquitectura más comúnmente utilizada con el algoritmo de aprendizaje de retropropagación es la red neuronal con alimentación adelantada (*feedforward*), la cual tiene en contrapartida la red neuronal de función de base radial (*radial basis network*) que quizás requiere más neuronas que la de alimentación adelantada, pero que a menudo puede ser diseñada en una fracción del tiempo que toma entrenar la otra.

Respecto al número de capas, éstas deben ser mayores si el problema es no lineal y complejo pero, en general, un problema podrá representarse bastante bien con una o dos capas ocultas. El número de neuronas por capa normalmente se determina por ensayo y error, aunque existe el criterio de considerar al promedio entre el número de entradas y salidas como un valor referencial del número de neuronas en las capas ocultas [17].

En el caso general, se quiere distinguir entre regiones del espacio. Una red neuronal debe aprender a identificar esas regiones y asociarlas a la respuesta correcta [4].

Una vez hecho el entrenamiento, se hace el test, donde aplicando los pesos calculados a las entradas se clasifican las muestras.

Todo esto se puede ver en la Ilustración 19: "Esquema de una red neuronal con una capa oculta".

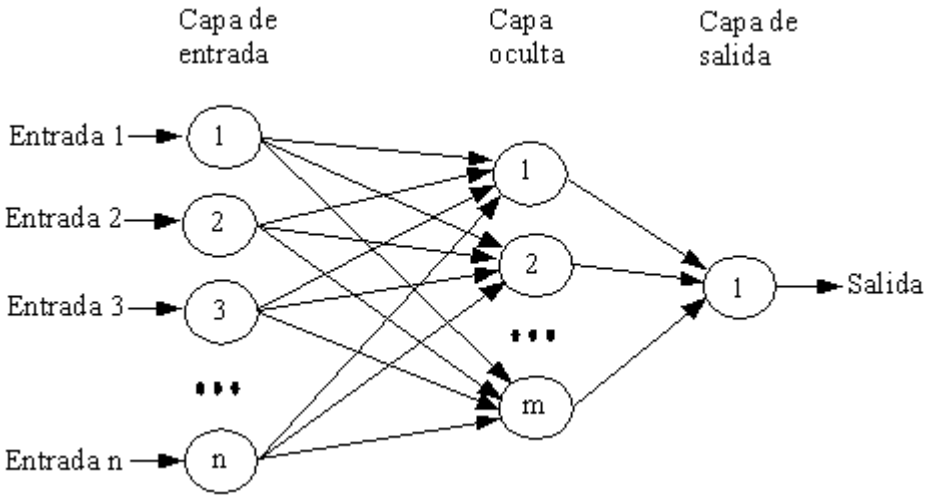

Ilustración 19: "Esquema de una red neuronal con una capa oculta"

El objetivo final es que el clasificador consiga un error de generalización pequeño. El comportamiento típico del error de entrenamiento de un clasificador decrece monótonamente durante la fase de entrenamiento, mientras que el error sobre el conjunto de validación decrece hasta un punto a partir del cual crece, lo que indica que a partir de ahí realiza un sobreajuste sobre los datos de entrenamiento. Por eso, el proceso de entrenamiento debe finalizar cuando se alcance el primer mínimo de la función del

error de validación. Se puede visualizar el aumento del error según aumenta el conjunto de validación en la Ilustración 20: "Punto de detención de la validación".

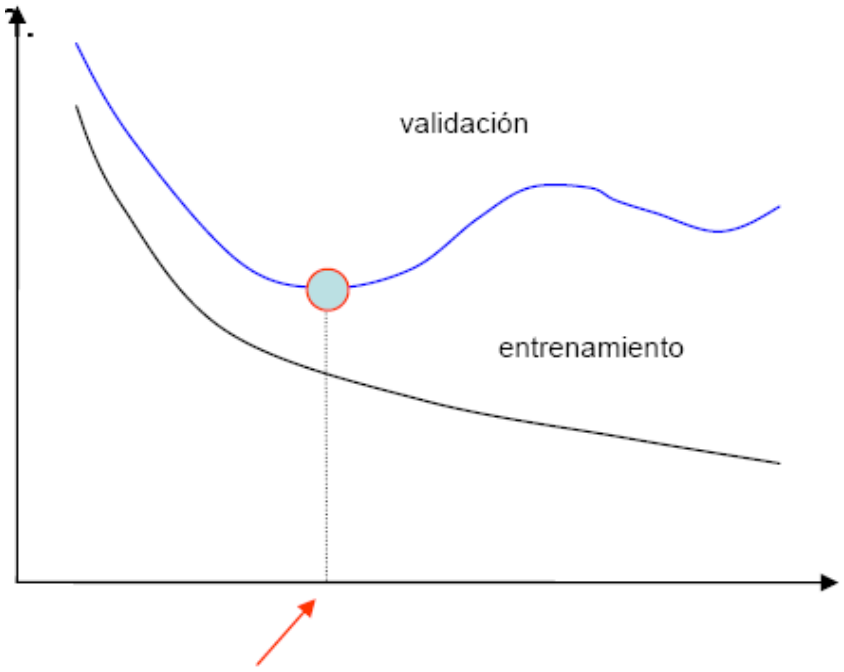

Ilustración 20: "Punto de detención de la validación"

El método de la validación cruzada divide el conjunto de entrenamiento en entrenamiento propiamente dicho y validación (para ajustar los parámetros del modelo). Esta división se puede visualizar en la Ilustración 21: "Esquema del método de validación cruzada".

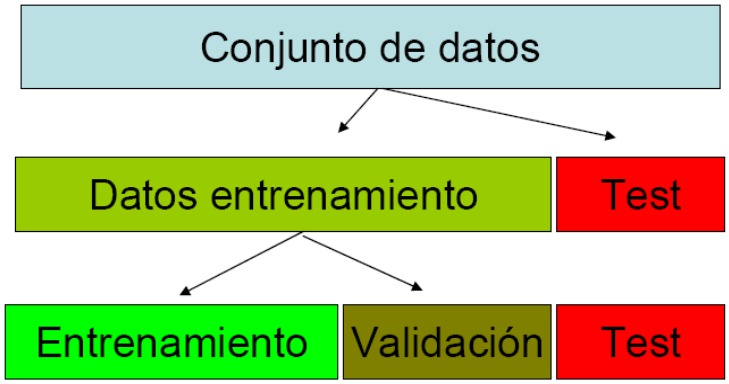

Ilustración 21: "Esquema del método de validación cruzada"

2.8.1.1. El perceptrón

El perceptrón clásico (Rosenblatt) es la forma más simple de una red neuronal utilizada para la clasificación lineal de patrones linealmente separables (patrones que se localizan en los lados opuestos de un hiperplano, también conocido como *dicotomía*) [6]. En la Ilustración 22: "Esquema general de un perceptrón" se puede visualizar esta idea.

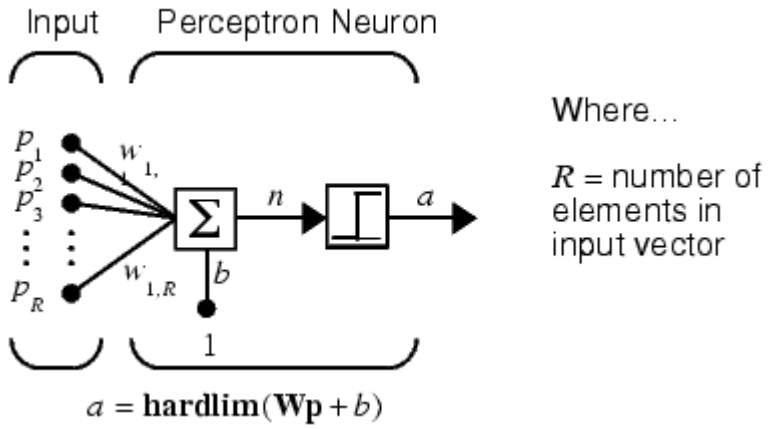

Ilustración 22: "Esquema general de un perceptrón"

Para saber qué tipo de patrones pueden ser reconocidos y qué limites tiene el perceptrón, hay que saber la velocidad del procesamiento paralelo, qué porcentaje de tarea computacional puede ser paralelizada y qué porcentaje es inherentemente secuencial.

Un perceptrón toma una decisión basada en una separación lineal del espacio de entrada. Esto reduce los tipos de problemas con solución con un solo perceptrón. Más separaciones generales del espacio de entrada pueden ayudar a ocuparse de otros problemas sin solución con una sola unidad de umbral. Las funciones usadas para discriminar entre regiones del espacio de entrada se llaman curvas de decisión.

En el reconocimiento de patrones estadístico, se asume que los patrones a ser reconocidos están agrupados en clusters (parte del espacio en el que se agrupan datos

con características semejantes) en el espacio de entrada. Usando una combinación de curvas de decisión se tratará de aislar un cluster de los otros [4].

Supóngase que las variables de entrada al perceptrón están originadas por dos clases linealmente separables. Sea X1 el subconjunto de vectores de entrenamiento x1(1), x1(2),… tal que pertenecen a C1, y sea X2 el subconjunto de vectores de entrenamiento x2(1), x2(2),… tal que pertenecen a C2; la unión de X1 y X2 es el conjunto de entrenamiento completo. Dados los conjuntos de vectores X1 y X2 para el entrenamiento del clasificador, el proceso de entrenamiento involucra el ajuste de pesos w de tal manera que las dos clases C1 y C2 sean separables por un hiperplano. Entonces existe un vector de pesos tal que:

$w^T x > 0$ para cualquier vector de entrada x que pertenezca a la clase C1.

$w^T x \leq 0$ para cualquier vector de entrada x que pertenezca a la clase C2.

[6]

Esta idea [6], se muestra en la Ilustración 23: "Plano de separación de un perceptrón".

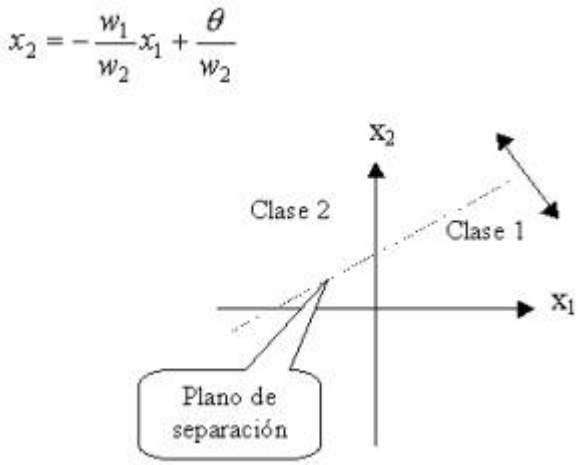

Ilustración 23: "Plano de separación de un perceptrón"

Para un mejor ajuste de los pesos del perceptrón se puede utilizar el algoritmo de retropropagación de errores. Es un método de aprendizaje supervisado de redes neuronales basado en el descenso por gradiente. Este se compone de cuatro pasos:

- Inicialización de pesos
- (Hacia delante) Cálculo de la salida de la neurona para un patrón de entrada
- (Hacia atrás) Cálculo de los errores (propagación del error hacia atrás)
- Modificación de los pesos

Cuando un modelo posee demasiados parámetros puede haber problemas de sobreajuste. Eso conlleva problemas de capacidad de generalización. Se puede ver la capacidad de generalización en la Ilustración 24: "Esquema de la capacidad de generalización en relación con el conjunto de aprendizaje".

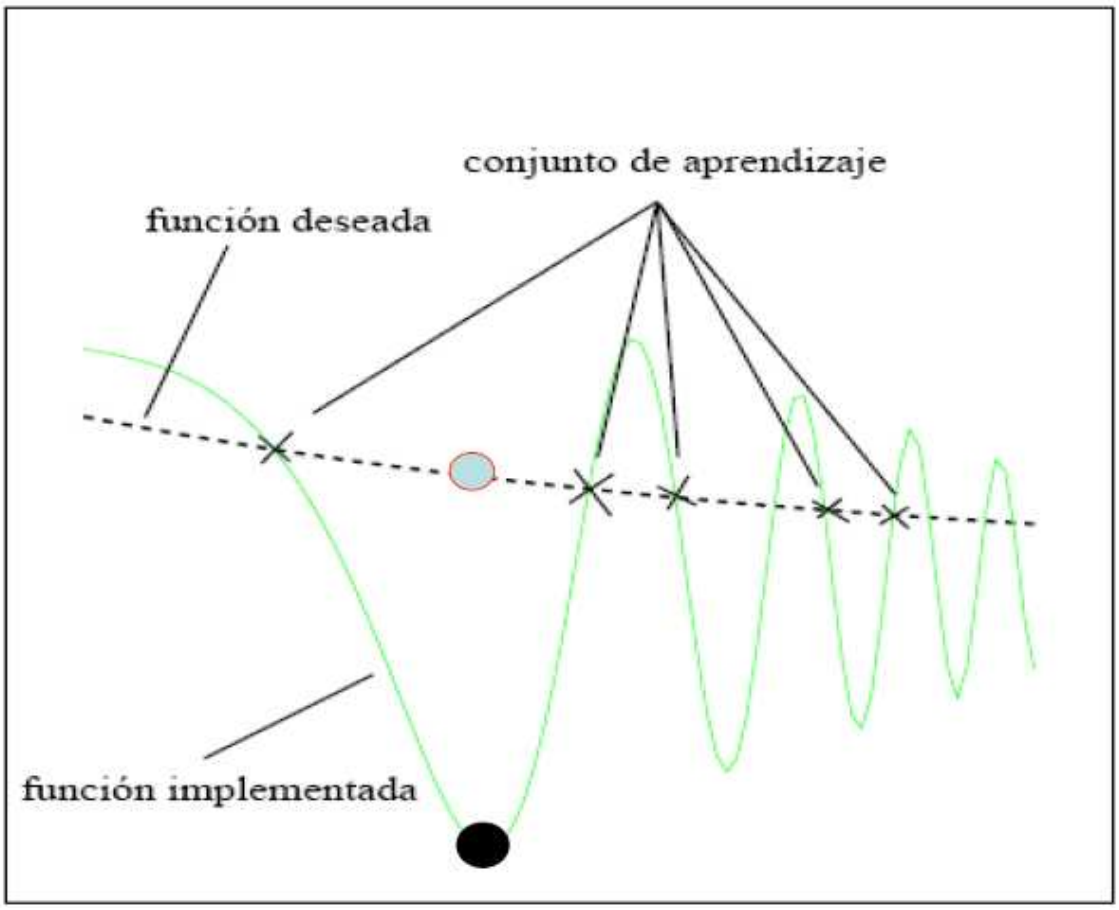

Ilustración 24: "Esquema de la capacidad de generalización en relación con el conjunto de aprendizaje"

2.8.2. Las máquinas de vectores de soporte

La máquina de vectores de soporte o SVM (*Support Vector Machine*) es un tipo de clasificador no lineal binario. Usa un método óptimo para resolver problemas de aprendizaje máquina [16] y puede ser utilizado para clasificación de patrones y para regresión no lineal.

La idea principal es construir un hiperplano como una superficie de decisión en un espacio de características de alta dimensión en el cual se maximice el margen de separación entre los ejemplos positivos y los negativos. La SVM es una implementación aproximada del método de la minimización estructural de riesgos. El coeficiente de error de una máquina de aprendizaje sobre los datos de prueba (coeficiente de error de generalización) está acotado por la suma del coeficiente de error de entrenamiento. En el caso de patrones separables, una SVM produce un valor cero para el primer término y minimiza el segundo término. De acuerdo con esto, la SVM provee una buena generalización para problemas de clasificación de patrones, a pesar de que no incorpora un dominio del conocimiento del problema.

El algoritmo de aprendizaje de vector soporte se puede utilizar para construir los siguientes tipos de máquinas de aprendizaje:

- Máquinas de aprendizaje polinomial
- Redes de función de base radial: si el *kernel* usado es una función de base radial Gaussiana (RBF, *radial basis function*) el correspondiente espacio de características es un espacio de Hilbert de dimensión infinita.
- Perceptrones con dos capas: no se hace el truco del *kernel*, por lo que queda un clasificador que ubica el umbral donde se maximizan los márgenes. Es decir, se diseña como un perceptrón pero el umbral se elige atendiendo a otros criterios.

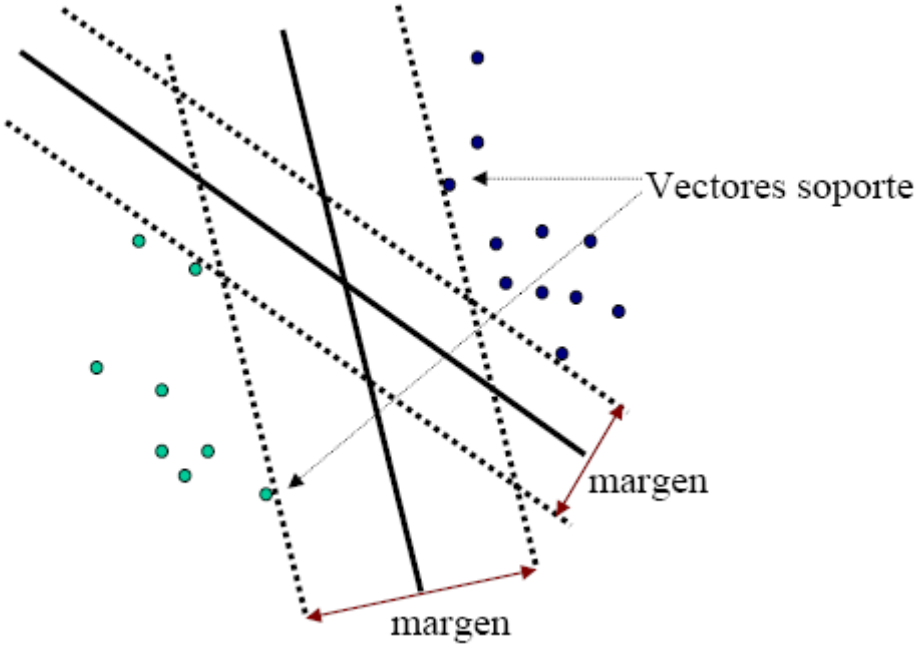

Ilustración 25: "Idea del hiperplano óptimo para patrones linealmente separables"

Para un vector de pesos w y un umbral b dados, la separación entre el hiperplano,

$$w^T x + b = 0$$

Ecuación 1: "Condición del hiperplano óptimo de separación"

y los datos más cercanos se denomina el margen de separación. El objetivo de la máquina de vector de soporte es encontrar un hiperplano en particular tal que se maximice el margen de separación. Si esto se cumple, la superficie de separación se denomina hiperplano óptimo.

Considerando el caso de patrones no separables, no va a ser posible construir un hiperplano separador sin encontrar errores de clasificación. De cualquier modo, se desea encontrar un hiperplano óptimo que minimice la probabilidad de errores de clasificación, promediado sobre el conjunto de entrenamiento. Esta idea se representa en la Ilustración 25: "Idea del hiperplano óptimo para patrones linealmente separables". Se puede decir que el problema queda planteado de la siguiente manera: dado un conjunto de entrenamiento $\{(x_i, d_i)\}_{i=1}^{N}$, hay que hallar los valores óptimos del vector de pesos *w* y del umbral *b* tal que satisfaga las restricciones:

$$d_i\left(w^T x_i + b\right) \geq 1 - \zeta, \text{para} \quad \text{i} = 1,2,..., \text{N}, \xi_i \geq 0 \text{ para todo i}$$

Ecuación 2: "Restricciones del hiperplano óptimo"

y tal que el vector de pesos $\boldsymbol{w}$ y las variables lentas o de holgura ξ_i minimicen la función de coste:

$$\Phi(w, \zeta) = \frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i$$

Ecuación 3: "Función de coste"

donde C es un parámetro positivo especificado por el usuario y ξ es la anchura de la función. El parámetro C controla el compromiso entre la complejidad de la máquina y el número de puntos no separables; puede ser visto como un parámetro de regularización frente a los errores de clasificación. Debe ser seleccionado por el usuario.

La idea básica de una máquina de vector soporte se basa en operaciones matemáticas:

- La transformación no lineal de un vector de entrada en un espacio de características de gran dimensión que está oculto tanto de la entrada como de la salida.
- La construcción de un hiperplano óptimo para separar las características descubiertas anteriormente.

El primer paso se logra con el teorema de Cover sobre la separabilidad de patrones. Ese espacio multidimensional construido por patrones no linealmente separables puede ser transformado en un nuevo espacio de características donde los patrones son linealmente separables. Para ello tiene que cumplir dos condiciones: que la transformación sea no lineal y que la dimensión del espacio de características sea lo suficientemente grande.

El segundo paso define el hiperplano como una función lineal de los vectores determinados a partir del espacio de características y no del espacio de entradas original. El hiperplano se construye de acuerdo con el principio de minimización estructural de riesgo [6].

2.8.2.1. SVM no lineales: el truco de *kernel*

En aprendizaje máquina, el truco de *kernel* es un método que permite usar un clasificador lineal para resolver un problema no lineal mapeando las observaciones no lineales originales en un espacio de más alta dimensión. En este nuevo espacio se emplea el clasificador lineal, lo que hace que una clasificación lineal en el nuevo espacio sea equivalente al empleo de un clasificador no lineal en el espacio original. Esto se puede visualizar en la Ilustración 26: "La transformación de los datos puede hacerlos linealmente separables".

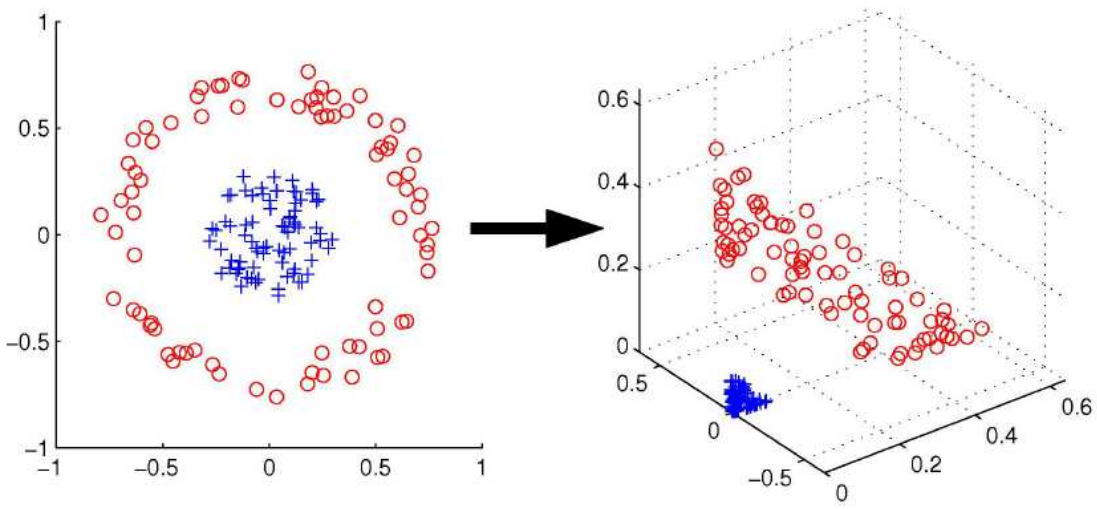

Ilustración 26: "La transformación de los datos puede hacerlos linealmente separables"

Esto se hace usando el teorema de Mercer, que mantiene que cualquier función *kernel* continua, simétrica y semi-definida positiva K(x, y) (cuya expresión se puede ver en la Ecuación 4: "Ecuación de un *kernel* semidefinido positivo") puede ser expresada como un producto escalar en un espacio de dimensión alta.

$$\sum_{i,j} K(x_i, x_j) c_i c_j \geq 0$$

Ecuación 4: "Ecuación de un *kernel* semidefinido positivo"

El truco de *kernel* realiza transformaciones que exclusivamente puedan ser expresadas a partir de un producto entre dos vectores, como se puede ver en la Ecuación

5: "El *kernel* puede ser expresado como un producto vectorial de dimensión probablemente alta". En cualquier parte donde el producto escalar es usado, se reemplaza con la función *kernel*. De ese modo, un algoritmo lineal puede ser fácilmente transformado en uno no lineal. Este algoritmo no lineal es equivalente al lineal en el espacio de φ. Sin embargo, como se usan los *kernels*, la función φ nunca se calcula explícitamente. Esto es deseable, porque el espacio de altas dimensiones puede ser infinitamente dimensional.[5]

$$K(x, y) = \varphi(x) \cdot \varphi(y).$$

Ecuación 5: "El *kernel* puede ser expresado como un producto vectorial de dimensión probablemente alta"

Algunos ejemplos de funciones *kernel* son:

- *Kernel* lineal.
- *Kernel* polinomial de grado d.

$$K(x, y) = (x^T y + 1)^d$$

Ecuación 6: "*Kernel* polinomial de grado d"

- *Kernel* de función de base radial (RBF) con anchura σ. Está estrechamente relacionado con las redes neuronales RBF.

$$K(x, y) = \exp\left(-\|x - y\|^2 \Big/ (2\sigma^2)\right)$$

Ecuación 7: "*Kernel* de función de base radial con anchura σ"

- *Kernel* sigmoide con parámetros κ y θ. No satisface la condición de Mercer para todo κ y θ

$$K(x, y) = \tanh\left(\kappa x^T + \theta\right)$$

Ecuación 8: "*Kernel* sigmoide con parámetros κ y θ"

En este proyecto se utilizará el *kernel* lineal por su sencillez y el RBF por los buenos resultados que reporta.

2.9. Medidas de calidad

Las medidas de calidad se calculan para evaluar el rendimiento de los diferentes algoritmos estudiados en el proyecto, para así poder compararlos y utilizar el óptimo en cada caso. Para este proyecto es necesario introducir los conceptos de las curvas ROC, las de *precision-recall* y la medida F.

2.9.1. Curva ROC

La curva ROC (o curva característica de operación) es la que relaciona dos características en compromiso de decisión, la probabilidad de falsa alarma (P_{FA}) y la probabilidad de detección (P_D).

$$P_{FA} = \Pr(D_1 \mid H_0)$$

Ecuación 9: "Probabilidad de Falsa Alarma"

$$P_D = \Pr(D_1 \mid H_1)$$

Ecuación 10: "Probabilidad de Detección"

donde H_0 y H_1 son las hipótesis de la clase 0 y la clase 1 respectivamente, y D_0 y D_1 son las decisiones de la clase 0 y la clase 1 del clasificador.

La curva se recorre al variar el umbral con el que se compara la salida de un clasificador blando para así obtener una clasificación dura (1 o 0). La P_{FA} es la probabilidad de decidir que una muestra pertenece a la clase positiva cuando no pertenece (detectar D_1 bajo la hipótesis H_0), y la P_D es la probabilidad de detectar bien una caso positivo (detectar D_1 bajo la hipótesis H_1). Ambas medidas toman valores entre cero y uno.

La curva se encuentra por encima de la diagonal principal. El clasificador es tanto mejor cuanto más se acerque a la zona superior izquierda. El área de la ROC es lo que se utiliza como parámetro de calidad [8]. Se puede ver la comparación de varios tipos de curvas en la Ilustración 27: "Ejemplos de curva ROC".

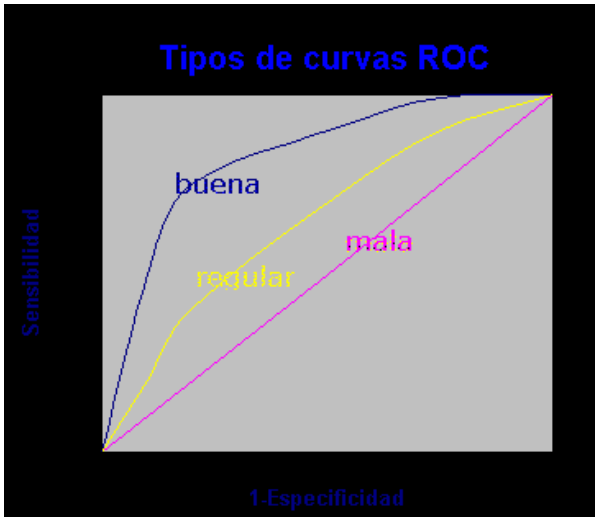

Ilustración 27: "Ejemplos de curva ROC"

2.9.2. Curvas de precision-recall

La *precision* es la proporción, entre los documentos recuperados, de aquellos que son relevantes a la necesidad de información del usuario.

$$precision = \frac{[(\text{documentosRelevantes}) \cap (\text{documentosRecuperados})]}{[\text{documentosRecuperados}]}$$

Ecuación 11: "Precisión"

El *recall* es la fracción de documentos que, siendo relevantes, han sido recuperados satisfactoriamente [5]

$$recall = \frac{[(\text{documentosRelevantes}) \cap (\text{documentosRecuperados})]}{[\text{documentosRelevantes}]}$$

Ecuación 12: "*Recall*"

Mediante estas dos medidas se pueden obtener las curvas de *precision-recall*, que toman valores de 0 a 1 y se cortan en un punto, el cual es el punto óptimo de funcionamiento. Ese punto óptimo corresponde con el umbral óptimo. El umbral óptimo, la precisión y el *recall* se pueden ver en la Ilustración 28: "Ejemplo de curvas de precisión-*recall*".

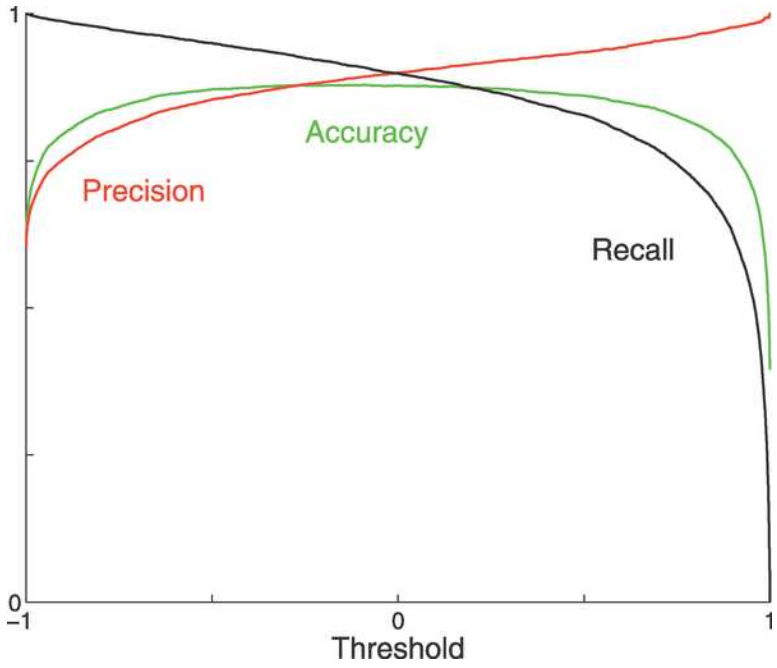

Ilustración 28: "Ejemplo de curvas de precisión-*recall*"

2.9.3. Medida F

La media armónica con pesos de la *precision* y el *recall* es la medida F. La fórmula general es:

$$F_\beta = \left(1+\beta^2\right).\left(precision \cdot recall\right) \Big/ \left(\beta^2 \cdot precision + recall\right)$$

Ecuación 13: "Medida F en función de β"

Para una aplicación de carácter general, sin un conocimiento especial de los requisitos de clasificación, se usa β=1, y la fórmula queda así:

$$F = 2.\left(precision \cdot recall\right) \Big/ \left(precision + recall\right)$$

Ecuación 14: "Medida F con β=1"

2.9.4. Matriz de confusión

La matriz de confusión es una tabla en la que cada columna representa el número de predicciones de cada clase (Di), mientras que cada fila representa a las instancias en la clase real (Hi). La mayor ventaja que ofrece este tipo de medida es que facilita la representación de las relaciones inter-clases en el clasificador (ver, por ejemplo, si el sistema está confundiendo dos clases en particular).

Tabla 1 Ejemplo de una matriz de confusión para un problema con 3 clases

Instancias reales \ Predicciones	**Clase 1**	**Clase 2**	**Clase 3**
Clase 1	P(D1|H1)	P(D2|H1)	P(D3|H1)
Clase 2	P(D1|H2)	P(D2|H2)	P(D3|H2)
Clase 3	P(D1|H3)	P(D2|H3)	P(D3|H3)

Cap. 3: Desarrollo del proyecto, implementación, experimentos, evaluación

3.1. Base de datos empleada en el proyecto

La base de datos empleada se ha obtenido de 'Tu Darmstad Database'[1]. Contiene tres conjuntos de imágenes, cada uno asociado a una clase: vacas, coches y motos. Cada conjunto contiene aproximadamente 100 imágenes. Manualmente se han hecho dos separaciones en cada uno de los conjuntos en las proporciones 60 – 40% para el entrenamiento y el test. A su vez, al entrenar las SVM, se utiliza una validación cruzada 5-*fold* para optimizar los parámetros de éstas máquinas.

Se puede ver un ejemplo de cada clase en la Ilustración 29: "Ejemplo de una imagen del conjunto coches", en la Ilustración 30: "Ejemplo de una imagen del conjunto vacas" y en la Ilustración 31: "Ejemplo de una imagen de la clase motos".

Ilustración 29: "Ejemplo de una imagen del conjunto coches"

[1] Se puede obtener de pascallin.ecs.soton.ac.uk/challenges/VOC/databases.html

Ilustración 30: "Ejemplo de una imagen del conjunto vacas"

Ilustración 31: "Ejemplo de una imagen de la clase motos"

3.2. Desarrollo del proyecto

El proyecto llevará a cabo la implantación del modelo BoW para un problema de reconocimiento de imágenes. A su vez, diversos parámetros abiertos en el problema serán evaluados, tales como la forma de generar los vocabularios de palabras visuales, su tamaño, o el tipo de clasificador empleado para clasificar los histogramas de palabras generados a partir de las imágenes.

Se han realizado seis fases claramente diferenciadas, las cuales se describen en las siguientes secciones.

3.2.1. Extracción de descriptores en parches locales

Para realizar la extracción de descriptores primero se realiza una lectura de los archivos de la base de datos a tratar. A cada imagen se le aplica un reescalado para tener un procesado menos costoso computacionalmente.

Después se aplica el algoritmo *SIFT*. Esta función calcula para cada imagen una serie de descriptores asociados a regiones locales en la imagen que han resultado potencialmente discriminantes (*keypoints* o puntos clave). Los principales pasos que conlleva este algoritmo, como se puede consultar en la sección 2.5, son:

1. Detección de extremos en el espacio escala.
2. Localización de puntos clave.
3. Asignación de la orientación.
4. Generación del descriptor SIFT para cada punto clave o *keypoint*.

Cuando ya se ha terminado el proceso, se han generado para cada imagen unos descriptores de dimensión 128. La cantidad de descriptores por imagen es variable ya que dependen del contenido de la misma.

3.2.2. Cálculo de *codebooks*

Para calcular los *codebooks* se aplica el algoritmo k-*means*. Esta función divide los descriptores hallados previamente y calcula tantos centroides como se hayan especificado como parámetro de la función. Es una división iterativa, que va minimizando la suma, sobre todos los clusters, de la distancia del punto al centroide. Se utilizan las distancias euclídeas [1].

Tras esto se obtienen tantas *codewords* o palabras visuales como se hayan especificado (los *codewords* serán los centroides), los cuales conforman el vocabulario o *codebook* asociado al problema. Cabe esperar que cada concepto o clase a clasificar contenga unas *codewords* identificativas que la distinga del resto de clases y, por otro lado, algunas que correspondan al fondo de la imagen y, por lo tanto, puedan confundir dicha clase con otras (por ejemplo, en una imagen de la clase motos y una de la clase coches se puede confundir el fondo al tener elementos comunes: asfalto de la carretera, casas de una ciudad…).

Cada vez que se calcula un *codebook* el algoritmo k-*means* comienza en un punto aleatorio. Por ello, para los mismos datos y el mismo número de centroides, se obtienen resultados diferentes. Es por eso que el proceso del cálculo de *codebooks* se realiza un cierto número de iteraciones, generando así varios vocabularios, para después elegir el óptimo de todos ellos.

La determinación de que un *codebook* sea el óptimo se toma utilizando las distancias de los centroides a los descriptores. Aquel *codebook* que tenga una menor distancia total será el óptimo, ya que será el que mejor ha hallado los centroides.

En este proyecto se ha experimentado con la creación de tres tipos de *codebooks*:

1. ***Codebook* global**: es aquél generado utilizando muestras pertenecientes a imágenes de todas las categorías. Este *codebook* es común a todos los

clasificadores, con independencia de las categorías que traten de detectar. Además, es el que más concuerda con la idea de un vocabulario, pues es común a todo el problema.

2. ***Codebooks* locales**: Los *codebooks* locales son individuales para cada clase y se generan a partir de muestras de imágenes de dicha clase. Si bien se alejan del sentido teórico de un vocabulario, pueden proporcionar mayor precisión que los globales, asignando más *codewords* a las características discriminantes de la clase de interés.

3. ***Codebook* global modificado**: este *codebook* es de carácter global pero se realiza a partir de los locales. Para generarlo, se concatenan en una matriz las palabras de los diferentes *codebooks* locales y se realiza una nueva etapa de *clustering* que dé lugar al número de palabras deseado. Con esta aproximación se trata de obtener un *codebook* global que mantenga la precisión de los locales, pues aquellas palabras de cada vocabulario local que no estén en los otros vocabularios (palabras asociadas a la clase de interés) se mantendrán en el final. Por otro lado, aquellas palabras que se repitan en varios *codebooks* (asociadas al fondo en general) se unirán en el proceso de *clustering.*

3.2.3. Asignación de *codewords*. Generación de histogramas de descriptores

La asignación de *codewords* consiste en determinar qué *codeword* corresponde a cada descriptor de una imagen. Para ello, se calcula la distancia euclídea entre cada *codeword* ***w*** y cada descriptor ***x***, y el *codeword* de distancia menor será el que corresponda a ese descriptor.

La distancia euclídea *d* se calcula según la Ecuación 15: "Distancia euclídea":

$$d(\mathbf{x},\mathbf{w}) = \sum_{i=1}^{128} (x_i - w_i)^2$$

Ecuación 15: "Distancia euclídea"

Una vez se han asignado todos los *codewords* a los descriptores, se calcula el histograma que modela la aparición de palabras en una imagen. Para que el histograma sea independiente del número de descriptores que tiene cada imagen (que es variable) se realiza una normalización.

3.2.4. Clasificación de imágenes

En este punto se han utilizado tres tipos de máquinas para clasificar las imágenes: a) el perceptrón monocapa con decisión blanda y la máquina de vector de soporte (SVM), b) con la función de kernel lineal y c) con la función de base radial (*Radial Basis Function*).

Además, se han evaluado dos tipos de clasificadores para el caso de la SVM:

- Detectores de conceptos basados en SVMs monoclase: detectan la aparición o no de un concepto en una imagen.

- Clasificadores de carácter multiclase: dada una imagen, eligen la clase a la que pertenece (de entre las clases posibles).

3.2.4.1. Clasificación mediante perceptrón monocapa.

En el primer caso, se utilizan tres perceptrones monocapa como decisores binarios de la presencia o no de cada una de las clases. Para crear una red neuronal se utilizan datos obtenidos del histograma conjunto de las tres clases realizado tras la asignación de *codebooks*, concretamente los valores mínimos y los máximos, y las etiquetas, que son unos vectores cuya longitud se corresponde con el número de imágenes a entrenar, y que valen 1 o 0 en función de si pertenecen a la clase que se va a clasificar o no. Éste es un caso de detección, es decir, se decide si un concepto aparece o no, con independencia de la salida para ese caso de los otros perceptrones.

Si el clasificador es multiclase, no puede utilizarse el perceptrón.

Después la red se entrena tres veces con las etiquetas de cada clase. Así se obtienen tres perceptrones. En la Ilustración 32: "Ejemplo del perceptrón utilizado" se muestra el que podría ser uno de los tres perceptrones. La entrada sería un histograma conjunto

(una matriz de tamaño nº *codewords* x nº clases), la función de entrada sería la red creada y la función de activación es la tangente sigmoidal. La salida son valores que van de -1 a 1 en función de si pertenecen o no a esa clase.

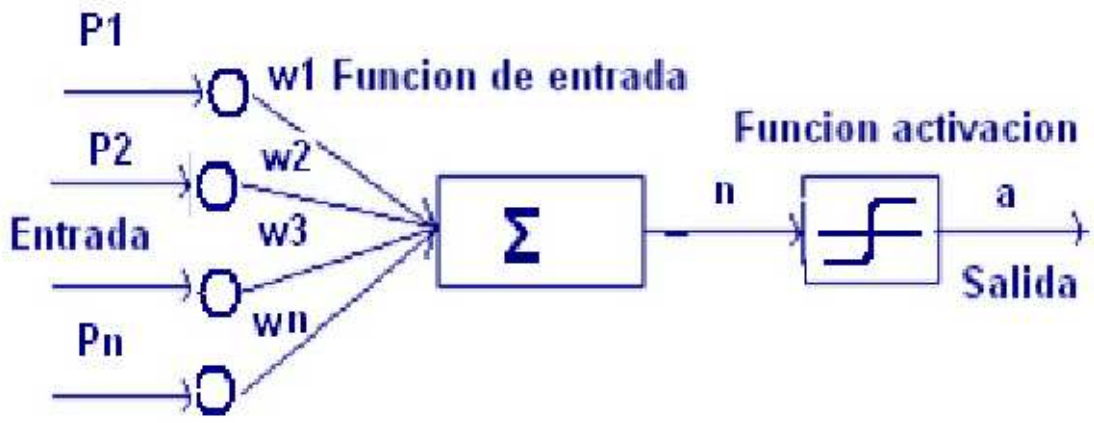

Ilustración 32: "Ejemplo del perceptrón utilizado"

3.2.4.2. Clasificación mediante SVM

En el segundo caso, se utiliza una máquina de vector soporte con *kernel* lineal o con RBF. Para ambas versiones, se debe entrenar la red realizando una validación cruzada para así obtener los valores óptimos de los parámetros configurables.

En el caso de utilizar *kernel* lineal, se realiza un barrido del parámetro del coste aplicado al error (parámetro C en la Ecuación 3: "Función de coste"). En el caso de emplear un *kernel* RBF, además de la validación del parámetro C, es necesario elegir la anchura del *kernel* óptimo para el problema (parámetro σ en la Ecuación 7: "*Kernel* de función de base radial con anchura σ").

Si se trabaja en un problema de detección la clasificación es monoclase, se crean y entrenan tres SVMs distintas a las que se les pasan diferentes parámetros. El primer parámetro es el vector de etiquetas. El segundo es un histograma conjunto (una matriz de tamaño nº *codewords* x nº imágenes) el cual ha sido normalizado para que contenga valores entre 0 y 1. Y el tercero son los parámetros específicos correspondientes a la estructura de la SVM (par {C, σ}).

Si la clasificación es multiclase, se crea y entrena una única SVM multiclase (de hecho, dado el carácter binario de la SVM, internamente se entrenan diferentes SVMs, pero esto resulta transparente al usuario). La diferencia con las redes anteriores es que las etiquetas se pasan en una matriz con valores 1, 2 o 3 en función de la clase a la que pertenezca la imagen. El resto de parámetros se introducen igual.

3.2.5. Simular la red neuronal

Tras obtener las redes resultantes con los datos de entrenamiento, comienza la validación. Para ello, se extraen los descriptores del conjunto de test, se asignan a los *codewords* y con el histograma se simula la red neuronal.

Ésta da como resultado unas variables de clasificación, que contienen unos o ceros dependiendo si se ha decidido que pertenece a la clase o no. Si el clasificador es multiclase, la salida vale 1, 2 o 3 en función de la clase a la que pertenezca la imagen. Con esos datos se podrán calcular diversas medidas de calidad.

3.2.6. Cálculo de medidas de calidad

Una vez que se han obtenido los vectores de clasificación, se pueden calcular ciertas medidas de calidad para evaluar el sistema. Las medidas que se calculan son las probabilidades de falsa alarma (Ecuación 9: "Probabilidad de Falsa Alarma") y detección (Ecuación 10: "Probabilidad de Detección"), la precisión (Ecuación 11: "Precisión") y el *recall* (Ecuación 12: "*Recall*") y la medida F (Ecuación 14: "Medida F con β=1"). Además, si se está utilizando un clasificador multiclase se muestra la matriz de confusión (Tabla 1 Ejemplo de una matriz de confusión para un problema con 3 clases) y la precisión media.

3.3. Implementación del proyecto

El código contiene varias funciones estructuradas de esta manera:

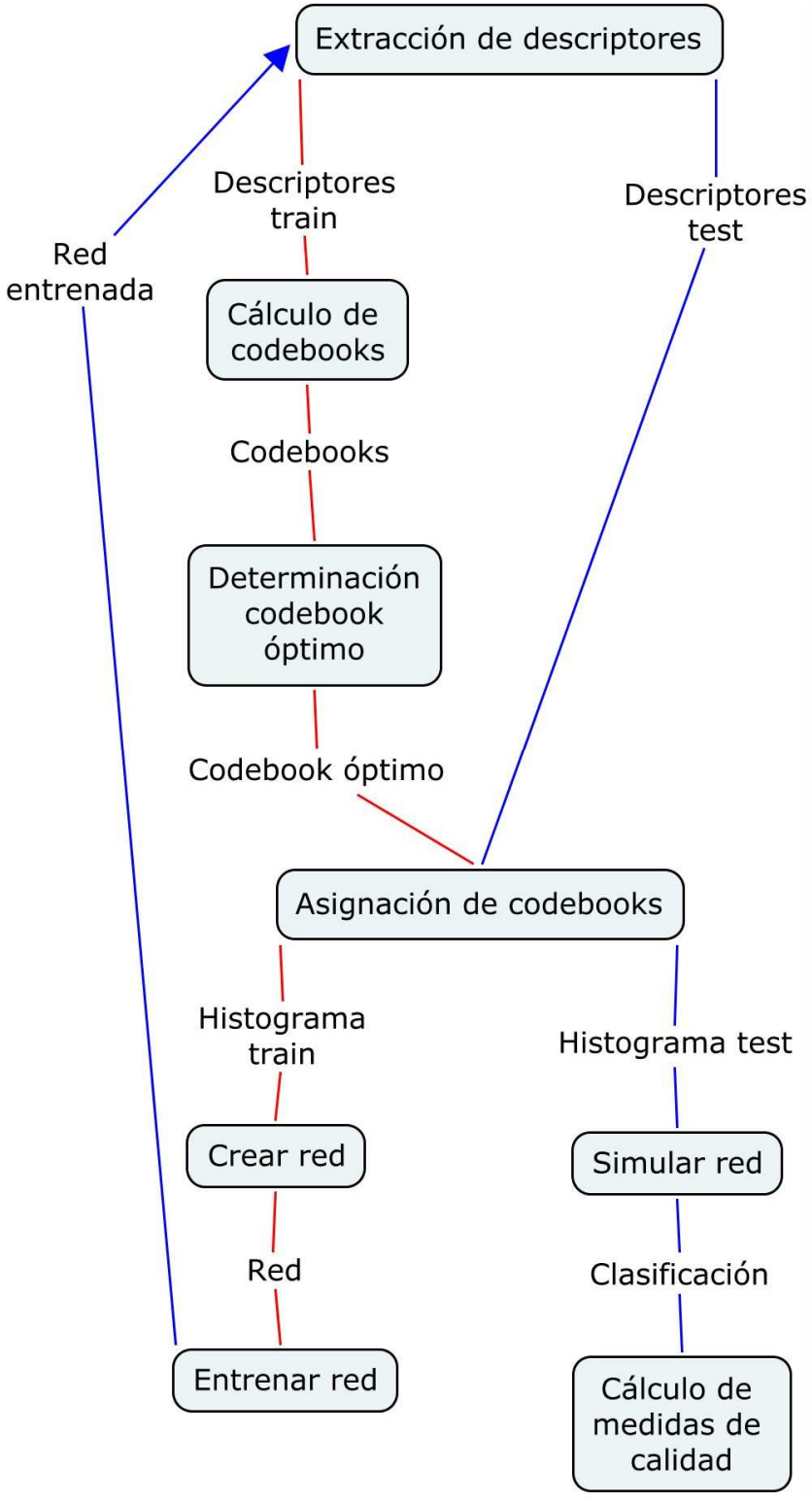

Ilustración 33: "Esquema de la implementación del proyecto"

Como se indica en la Ilustración 33: "Esquema de la implementación del proyecto", se utilizan diferentes funciones a las que cada vez se les pasan unos parámetros determinados. A continuación se describirá cada una de las funciones.

El proyecto se ha implementado en la plataforma de Matlab. Se han empleado diversas *toolboxes* de Matlab, las más importantes son: *Image processing* y *Statistics toolbox.* Además, se han empleado otras librerías externas como VLFeat[2] y SIFT[3].

3.3.1. LectorImagenes.m

Esta función recibe como parámetro un *string* que debe tomar los valores de 'Train' o 'Test'. En función de este parámetro cargará las imágenes que contenga una carpeta u otra (el conjunto total de imágenes se ha dividido y se ha separado en dos carpetas, el 60% están en la carpeta de entrenamiento y el 40% están en la carpeta de test) y se guardarán los datos de salida en unos archivos identificados como de test o de train.

Lo primero que hace la función es comprobar que los descriptores no se han calculado anteriormente (si están almacenados los resultados). Si es así, no continúa el proceso y muestra un mensaje informativo. Si no se han calculado, se procede a hacerlo.

A continuación el programa extrae los archivos de la carpeta en cuestión (las imágenes están separadas en carpetas según la clase a la que pertenezcan) y comprueba que son imágenes obteniendo la extensión del nombre del archivo. Después se leen las imágenes mediante la función *imread* (que pertenece a la *Image processing toolbox* de Matlab). El comando *sift* necesita una imagen en escala de grises en el rango [0,255]. Después, se reescala la imagen para que todas tengan un tamaño menor a 300x300, para que el procesamiento sea más rápido. Todo esto pertenece a la parte de preprocesado de imágenes.

A continuación se aplica la función *SIFT*, que proviene de una *toolbox* SIFT, para extraer las características de cada imagen, y se almacena el valor de los descriptores en

[2] El código está disponible en http://www.vlfeat.org/
[3] El código está disponible en http://www.cs.ubc.ca/~lowe/keypoints/

un archivo de texto plano. Para aplicar la función *sift*, que extrae los puntos de interés y los descriptores asociados a los mismos, se introduce como parámetro la imagen a tratar y devuelve las coordenadas de los parches locales y los descriptores que se están buscando. Los descriptores son vectores de dimensión fija 128, pero el número de descriptores por imagen no es conocido a priori.

3.3.2. Clustering.m

Esta función recibe como parámetros el número de *codewords*. Utilizando los descriptores de entrenamiento calculados en lectorImagenes.m y el número de *codewords* (que será el número de centroides), aplica la función k-*means* (del *Statistics Toolbox* de Matlab), que halla tantos centroides como se hayan especificado, y agrupa los descriptores a estos según el algoritmo de *nearest-neighbor* (vecino más próximo). Esto devuelve los índices de los *codewords*, los *codewords* en sí (que son vectores de 128 de longitud, y todos se almacenan en una matriz) y la suma de las distancias de los descriptores que se han agrupado a ese *codeword* hasta el *codeword* (el centroide). Todo esto se aplica para cada una de las clases y para el conjunto global. Las sumas de las distancias serán el parámetro decisivo para el cálculo del *codebook* óptimo.

La función devuelve los *codewords*, los índices y las sumas de las distancias (de las tres clases y del conjunto, es decir, las características locales y las globales).

3.3.3. CalcularCodebooks.m

Esta función recibe como parámetros el número de *codewords* que debe tener cada *codebook* y el número de iteraciones a realizar (cuántos *codebooks* se han de calcular).

Lo primero que hace es llamar a la función ‘lectorImagenes.m’ para hallar los descriptores del conjunto de entrenamiento. Después llama a la función ‘clustering.m’ para hallar los *codebooks*, y se le llama tantas veces como iteraciones se han definido.

3.3.4. CalcularIteracionOptima.m

Esta función recibe como parámetros el número de *codewords* y el número de iteraciones que se han realizado anteriormente.

Lo primero que hace es cargar los datos de los *codebooks* calculados anteriormente. Se cargan tantos como el número de iteraciones se hayan empleado en el paso anterior (nº de *codebooks* calculados). Después se busca, entre todas las iteraciones, aquella en la que se haya obtenido la mínima distancia, para cada clase y para el conjunto. Aquella iteración será la que tenga el *codebook* óptimo. Por último, sabiendo el número de la iteración que ha sido óptima para cada clase (características locales) y para el conjunto (características globales), se almacenan los *codebooks* como *codebooks* óptimos, para poder ser identificados más tarde.

3.3.5. AsignacionCodewords.m

Esta función recibe como parámetros el *string* 'Train' o 'Test' (para identificar si se está en el entrenamiento o en test) y el número de *codewords*.

Según el valor del parámetro que indica si se usarán los datos de test o de train, se cargan los números de imágenes y los *codebooks* óptimos. En un bucle, se van recorriendo todas las imágenes de cada clase, y en cada imagen, se recorre en otro bucle sus descriptores. Para cada descriptor y cada *codeword* se calcula la distancia euclídea entre ellos. Después se calcula el mínimo de todas las distancias calculadas para cada descriptor, de esa manera se sabrá a qué *codeword* se corresponde cada uno de los descriptores. Cuando se sepa, se almacena el número del *codeword* que se le ha asignado para cada descriptor en un vector llamado 'codewordAsignado'. Como cada imagen tendrá un número distinto de descriptores, este vector tendrá como tamaño máximo el número de descriptores de la imagen que tenga más descriptores, por lo que en el resto de los casos el vector se rellenará con ceros.

3.3.6. Histograma.m

La función histograma recibe como parámetros el *string* 'Train' o 'Test' (para identificar si se está en el entrenamiento o en el test y así cargar unos datos u otros) y el número de *codewords*.

Para cada imagen de cada clase (y de cada subconjunto en función de si se está en entrenamiento o en validación) hallamos, del vector 'codewordsAsignados', aquellos que sean distintos de cero (ya que se incluyeron ceros para que se tuviese una longitud homogénea y que así los ceros no influyan a la hora de calcular el histograma). Después se calcula el histograma mediante la función *hist*, que pertenece a la *Statistics Toolbox* de Matlab. El histograma representará la frecuencia de aparición de un *codeword* en una clase de imágenes. Luego se representan en diagramas de barras. En la Ilustración 34: "Ejemplo de histograma para 700 *codewords*" se puede ver un ejemplo de los resultados del proyecto.

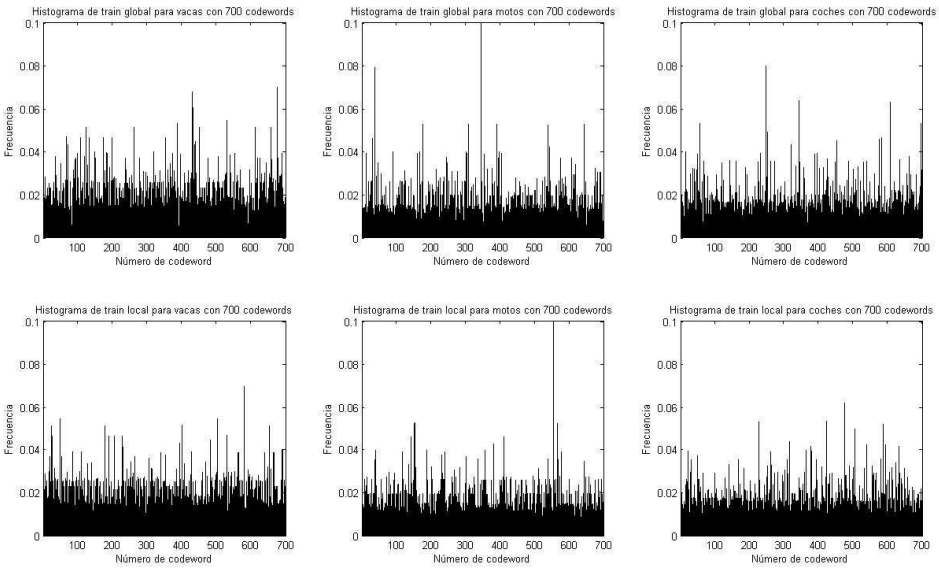

Ilustración 34: "Ejemplo de histograma para 700 *codewords*"

3.3.7. CalcularCodebookConjunto.m

La función calcularCodebookConjunto recibe como parámetros el número de *codewords* y el número de iteraciones.

Primero se calculan tantos *codebooks* conjuntos como iteraciones se hayan especificado. Cuando se tienen todos los *codebooks*, se busca el óptimo entre ellos y se almacena en un archivo.

Después se calculan los histogramas de entrenamiento y de test. Como el cálculo de ambos es idéntico, se recorre un bucle dos veces, una para entrenamiento y otra para test. Después se recorre otro bucle tres veces, una por cada clase, y se cargan los descriptores y sus tamaños. Para cada clase, se recorre de nuevo otro bucle con todas las imágenes que contenga. Por cada imagen de cada clase se calcula su valor del histograma de la misma forma que en la función 'histograma.m'. Por último, se almacenan los histogramas y los *codewords* asignados en sus archivos correspondientes.

3.3.8. EntrenamientoClasificador.m

La función entrenamientoClasificador recibe como parámetros el número de *codewords*, un *string* que representa el tipo de clasificador que se desea utilizar, perceptrón monocapa ('SLP'), máquina de vector soporte de tipo *kernel* lineal ('SVMKL') o de tipo RBF ('SVMRBF') y un *string* que representa el tipo de *codebook* a utilizar: 'Global', 'Local', 'Multiclase', o 'Conjunto' (global modificado).

Primero se establecen los nombres de los archivos que se van a guardar al final y se inicializan todas las variables que se van a emplear.

En el caso de que el clasificador sea 'SLP', se crea la red, empleando la función *newff* perteneciente a la *Neural Network Toolbox* de Matlab. Ésta crea un perceptrón con retropropagación con una función de activación tipo tangente-sigmoidal y como el perceptrón es monocapa, se especifica que sólo tenga una neurona y se le pasa como

parámetro el máximo y el mínimo del histograma del conjunto. Para que se puedan alcanzar los objetivos en el entrenamiento, se han de establecer 1500 épocas en los parámetros de la red global. Para obtener las redes, se entrena tres veces la red global con los histogramas del conjunto globales y las etiquetas correspondientes en cada uno de los tres casos. El caso de 'SLP' no se ejecuta si 'codebook' vale 'Multiclase'.

Si el parámetro tipo vale 'SVMKL', para obtener mejores resultados, se realiza un barrido lineal del parámetro C, que aparece descrito en la Ecuación 3: "Función de coste", que representa la penalización. Se utilizan diferentes valores en función del *codebook* y, además, si el parámetro elegido es el mínimo o el máximo de los establecidos en el vector, se calcularán unos nuevos valores mínimo y máximo y se repetirá el cálculo. El conjunto de datos se divide en 5 subconjuntos para realizar una validación cruzada (que permite establecer los parámetros óptimos de entrenamiento, luego validarlos y más tarde hacer el test). Se realiza un entrenamiento mediante la función *svmtrain* con todos los valores. Para todos ellos se halla la precisión resultante de entrenar la red. Después se halla la máxima precisión en cada caso debido al parámetro C, que es el coste que se le aplica al error. A continuación se entrena de nuevo con los parámetros óptimos y se obtienen las redes.

Si tipo vale 'SVMRBF' el entrenamiento se realiza igual que en el caso anterior, con la única diferencia de que también se realiza un barrido logarítmico del parámetro ξ (la anchura de la función *kernel*), el cual también varía sus valores máximos y mínimos en función del óptimo escogido. Este parámetro también aparece en la Ecuación 3: "Función de coste".

Las funciones *svmtrain* y *svmpredict* (la que se utilizará en la validación) corresponden a la librería LIBSVM[4] para máquina de vector soporte desarrollada por Chih-Chung Chang y Chih-Jen Lin.

[4] La librería se puede consultar y descargar en http://www.csie.ntu.edu.tw/~cjlin/libsvm/

3.3.9. ValidacionClasificador.m

La función validacionClasificador recibe como parámetros el número de *codewords*, un *string* que representa el tipo de clasificador que se desea utilizar, perceptrón monocapa ('SLP'), máquina de vector soporte de tipo *kernel* lineal ('SVMKL') o de tipo RBF ('SVMRBF') y un *string* que representa el tipo de *codebook* a utilizar: 'Global', 'Local', 'Multiclase', o 'Conjunto' (global modificado).

Primero se establecen los nombres de los archivos que se van a guardar al final y se inicializan todas las variables que se van a emplear. También se cargan los datos obtenidos del entrenamiento del clasificador.

En el caso de que el parámetro tipo sea 'SLP', se hace la simulación del perceptrón, mediante la función *sim* perteneciente a la *Neural Network Toolbox* de Matlab. Ésta dará la clasificación final utilizando los histogramas de test.

Si el parámetro tipo vale 'SVMKL' o 'SVMRBF' se recorren todas las imágenes de conjunto de test y se realiza la predicción con la función *svmpredict*.

Como resultado se obtienen los vectores de clasificación, que se almacenarán en un archivo.

3.3.10. CalcularMedidasCalidad.m

Esta función recibe como parámetro el número de *codewords* que se está empleando y un booleano que representa si se está utilizando un clasificador multiclase. Su finalidad es calcular la precisión y el *recall*, la medida F y las probabilidades de error de los tres clasificadores para cada tipo de clasificación.

Primero se establecen los nombres de los archivos a guardar y cargar, se inicializan las variables y se establecen los valores de algunas de ellas. En función del valor de esMulticlase se inicializan unas variables u otras.

Después, se calculan las probabilidades a posteriori. Se considera que se decide cuando el valor de la clasificación vale más que cero cuando se está utilizando un clasificador de tipo slp, cuando vale uno en el caso de utilizar una red neuronal o cuando valga uno, dos y tres (en función de si se decide vacas, motos o coches) si es multiclase. Si el clasificador es multiclase, se calcula la matriz de confusión y la precisión media.

Para obtener la *precision* y el *recall*, se calculan las imágenes recuperadas sumando las probabilidades de que se decida cada clase aun cuando la decisión no ha sido correcta. Los documentos relevantes son el número de imágenes de cada clase. Y los documentos relevantes recuperados son el número de documentos recuperados correctamente. Con todos los datos se calcula la *precision* y el *recall* para cada clase.

Para obtener las probabilidades de falsa alarma y detección se utilizan las probabilidades a priori y a posteriori que se han calculado en un primer momento en función de los resultados de la clasificación. Las ecuaciones para obtenerlas son Ecuación 9: "Probabilidad de Falsa Alarma" y Ecuación 10: "Probabilidad de Detección".

Para calcular la medida F se procede a utilizar la fórmula (Ecuación 14: "Medida F con β=1") con los valores de *precision* y *recall*.

Se ha incluido por seguridad una comprobación al final de cada cálculo de medida de calidad. Si ésta vale 0 o 1, salta un error y termina la ejecución.

3.3.11. PROYECTO.m

Esta función recibe como parámetros el número de *codewords* a emplear y el número de iteraciones a realizar. Su finalidad es llamar en el orden correcto a todas las funciones que comprenden el proyecto con los parámetros adecuados.

Las funciones se llaman en este orden: calcularCodebooks, calcularIteracionOptima, asignacionCodewords (del entrenamiento), histograma (del entrenamiento), asignacionCodewords (de la validación), histograma (de la validación), calcularCodebookConjunto y entrenamientoClasificador, validacionCasificador y con los siguientes valores:

- Clasificador SLP global, local, conjunto
- Clasificador SVMKL global, local, conjunto, multiclase
- Clasificador SVMRBF global, local, conjunto, multiclase

Por último, se llama a la función calcularMedidasCalidad para el caso monoclase y el multiclase.

3.3.12. ComparativaClasificadores.m

Esta función tiene la finalidad de mostrar una comparativa gráfica de los tres tipos de clasificadores utilizando sus características globales.

Primero, se cargan todos los valores de la medida F, y se almacenan en un array. Después, se interpolan todos esos valores para que la curva esté más suavizada. Y por último, se representan las tres gráficas.

3.4. Experimentos

En esta sección se describirán los experimentos realizados con las diferentes configuraciones contempladas en el proyecto. En particular se ha experimentado sobre:

- El tamaño de los *codebooks*: Los *codebooks* que se han calculado tienen tamaños que van desde 10 *codewords* hasta los 2000 *codewords*, con un intervalo entre ellos suficiente e incremental. No se calcularon *codebooks* con más de 2000 *codewords* porque en ese punto la calidad de los clasificadores ya se ha estabilizado lo suficiente, e incrementar el tamaño supondría aumentar la carga computacional sin una mejora en la calidad.

- Tipos de *codebooks:* se han probado diferentes tipos de *codebooks,* tal y como se ha descrito en la sección 3.2.2.

- Estructura y tipo de clasificadores: según lo descrito en la sección 3.2.4.

Es importante remarcar ciertas consideraciones y problemas surgidos a la hora de realizar las pruebas, en general debidos a la carga computacional elevada de los algoritmos involucrados. En los siguientes párrafos se describirán algunos de estos problemas.

El cálculo de descriptores tenía una gran carga de datos, por lo que no se podían emplear matrices, ya que el ordenador se quedaba sin memoria. Se decidió grabar los datos en un fichero de texto plano y luego leer de él.

Quizás lo más costoso haya sido el cálculo de los *codebooks*, que tiene una gran carga computacional y empleó muchas semanas de cálculo en el ordenador.

Tras calcular los *codebooks* y los histogramas se probó a utilizar un perceptrón monocapa simple (y la regla del perceptrón como método de entrenamiento) pero con

este se obtenían unos resultados de clasificación muy pobres, por lo que se decidió aplicar la retropropagación. Así el sistema ganó en calidad.

En cuanto al clasificador basado en máquina de vector soporte, se hicieron múltiples pruebas para establecer un rango de valores adecuados de los parámetros a los cuales se iban a hacer barridos. Para ello, se buscó un rango para que al elegir el valor óptimo no estuviese ni al principio ni al final del rango, y así asegurar que no hubiese un valor menor o mayor respectivamente que fuese mejor que el calculado como óptimo. Esto se hizo con todos los tamaños de *codebooks* empleados en el proyecto.

3.5. Evaluación

Como se ha comentado, los resultados del proyecto se pueden comparar en muchos aspectos. Uno de ellos es el número de *codewords* que debe tener el *codebook* para que la clasificación sea óptima. Otro es el mejor algoritmo a emplear para clasificar, el SLP, el SVM de *kernel* lineal o el SVM de RBF. Para ello se comparará la medida F en función de esos dos parámetros.

3.5.1. Estudio del tamaño del *codebook* sobre detectores monoclase

Para el caso de clasificadores monoclase o detectores de conceptos, los resultados de la comparación de la calidad (parámetro F) se muestran en la Ilustración 35: "Medida F para diferentes tamaños de *codebook* en clasificadores monoclase (detectores de conceptos)":

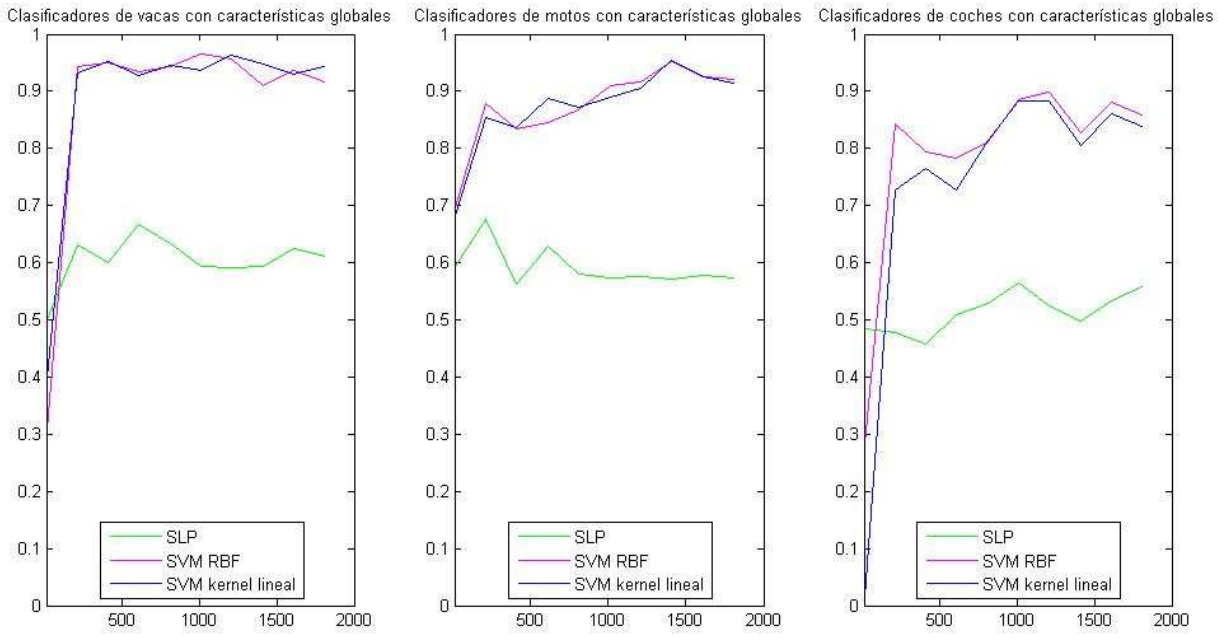

Ilustración 35: "Medida F para diferentes tamaños de *codebook* en clasificadores monoclase (detectores de conceptos)"

Como se puede observar en la figura, aproximadamente, a partir de 1000 *codewords* la calidad de todos los clasificadores se estabiliza. Las fluctuaciones que aparecen son debidas a las inicializaciones aleatorias al crear los *codebooks* (a pesar de haber hecho varias iteraciones y haber escogido el *codebook* óptimo, puede que sea de peor calidad que otros). A pesar de las fluctuaciones, la tendencia es claramente creciente para *codebooks* pequeños (menos de 500 palabras), estabilizándose poco a poco según crece su longitud. A partir de estas gráficas se ha decidido fijar el número de *codewords* a 1000 en el resto de los experimentos tratando de maximizar el compromiso complejidad-rendimiento.

Es importante destacar además, la superioridad de los clasificadores que usan las SVM, con independencia del *kernel* empleado, sobre el SLP. Por lo tanto, parece que la capacidad de generalización de las SVMs, altamente demostrada en la literatura, se hace visible también en este tipo de problemas.

En cuanto a las diferentes clases, se observa que la clase mejor discriminada es la de vacas y la peor la de coches, bajando la calidad ente ellas en término medio un 10% en todos los casos. La razón subyacente es la teórica similitud entre ciertos aspectos visuales presentes en los coches y en las motos, que pueden aumentar la confusión entre ambos.

3.5.2. Estudio de los tipos de *codebook*

En las ilustraciones siguientes se pueden comparar los resultados obtenidos por los detectores de conceptos utilizando las diferentes alternativas propuestas en la generación de *codebooks*. De nuevo, y para enriquecer la comparación entre los diferentes algoritmos de clasificación, se incluyen resultados para el SLP y las SVM con los dos *kernels* empleados. De todos modos, a lo largo de los experimentos, el comportamiento de los diferentes clasificadores es similar al mostrado en el estudio anterior, por lo que no se incidirá mucho más en ellos.

Comparando los distintos tipos de *codebook*, se pueden extraer algunas conclusiones de interés. Observando la Ilustración 36: "Medida F para 1000 *codewords* utilizando

clasificadores monoclase: resultados de la clasificación global", se deduce que los *codebooks* globales obtienen los que se podrían considerar resultados básicos del sistema, dado que son los que más se ajustan a la definición teórica de un vocabulario (común a toda la base de datos y, por lo tanto, a todos los clasificadores).

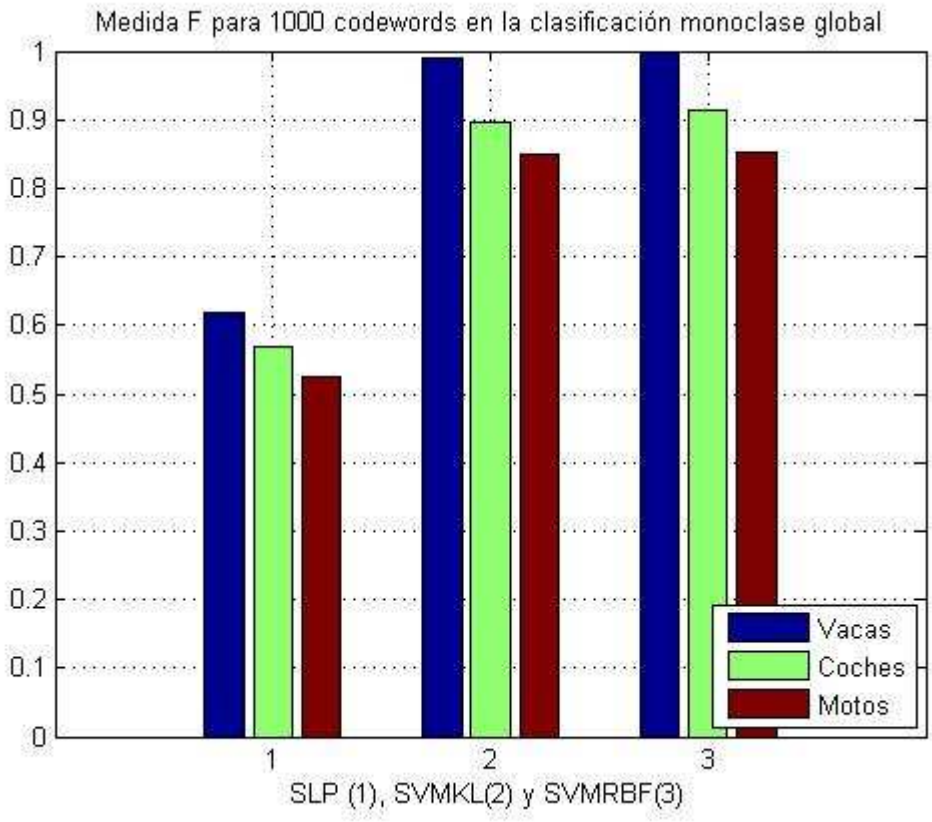

Ilustración 36: "Medida F para 1000 *codewords* utilizando clasificadores monoclase: resultados de la clasificación global"

En la Ilustración 37: "Medida F para 1000 *codewords* utilizando clasificadores monoclase: resultados de la clasificación local", se observa que los *codebooks* locales tienen como objetivo proporcionar una mayor precisión sobre los clasificadores, dado que un mayor número de palabras visuales se asociarán a aspectos discriminantes de las clases. Pero se observa que para la clase vacas esto no ocurre. La razón es que esta clase es muy diferente a la de motos o coches, y como las *codewords* de estas clases no han sido incluidas en el *codebook* de vacas, se asignan casi aleatoriamente al hacer los histogramas. En el caso de coches y motos, la aparición de *codewords* desconocidas es más tenue.

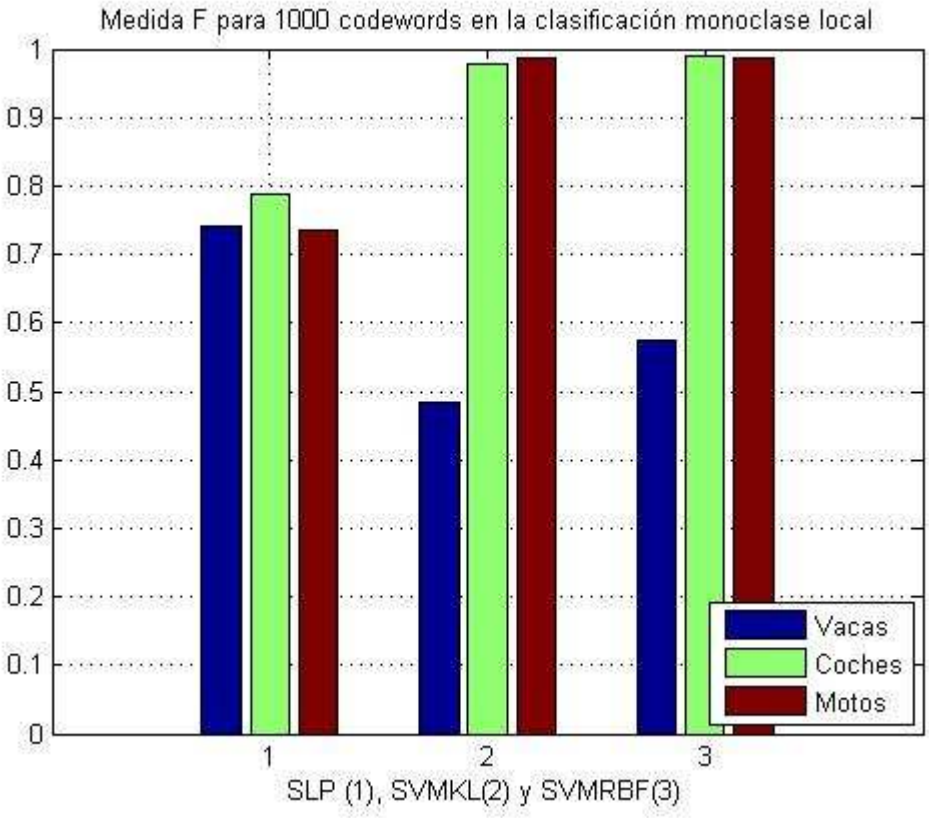

Ilustración 37: "Medida F para 1000 *codewords* utilizando clasificadores monoclase: resultados de la clasificación local"

Por último cabe destacar que la alternativa de *codebook* global modificado (aquél que genera *codebooks* locales que luego se fusionan en uno global) mejora los resultados obtenidos por el *codebook* global. Hay más precisión ya que se representan *codewords* comunes a todas las imágenes (elementos del fondo de las imágenes, por ejemplo).

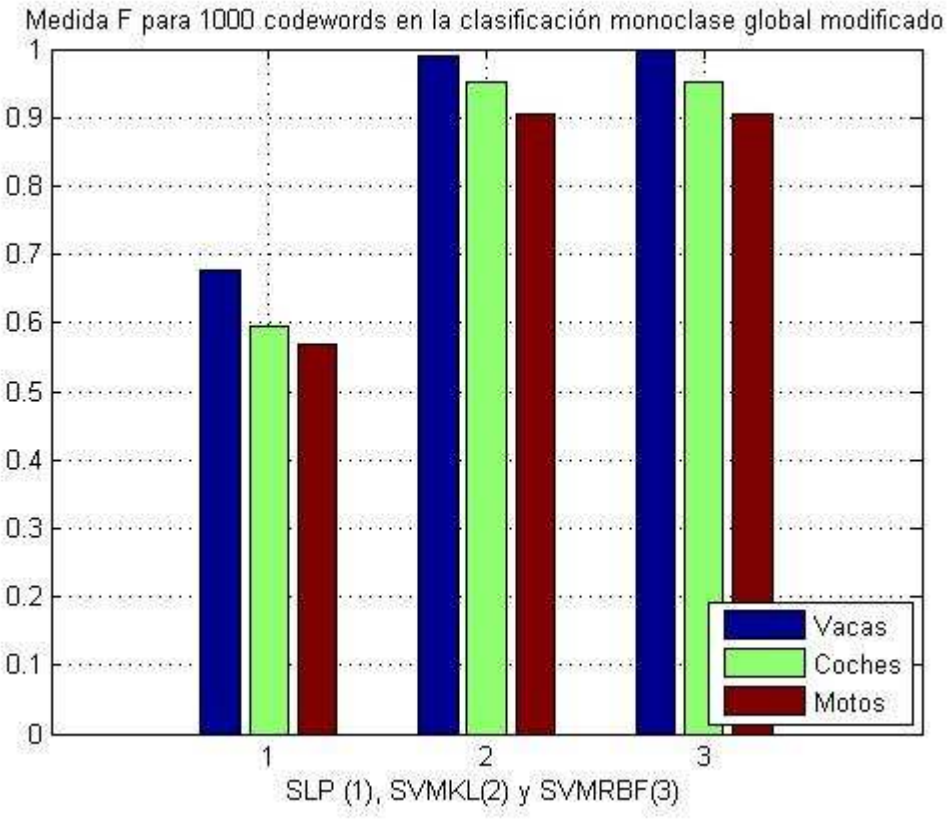

Ilustración 38: "Medida F para 1000 *codewords* en el clasificador global modificado"

Atendiendo a los resultados individuales de cada clase, en general, la clase de Vacas es la mejor discriminada entre las tres lo cual, a tenor de las características de las clases, es lógico (coches y motos comparten ciertas características como la aparición de ruedas y otros elementos comunes del *background*: carretera, aceras…).

3.5.3. Aproximación multiclase

En la aproximación multiclase únicamente se han evaluado la SVMRBF y el SVMKL (como se había comentado con anterioridad el SLP no puede hacer una clasificación multiclase). En la Ilustración 39: "Medida F para 1000 *codewords* en la clasificación multiclase", pueden verse los resultados obtenidos en este caso. En general, se puede concluir que dichos resultados, aunque no parece que obtengan una mejora muy drástica, son ligeramente superiores a los logrados por la aproximación monoclase con *codebooks* globales,

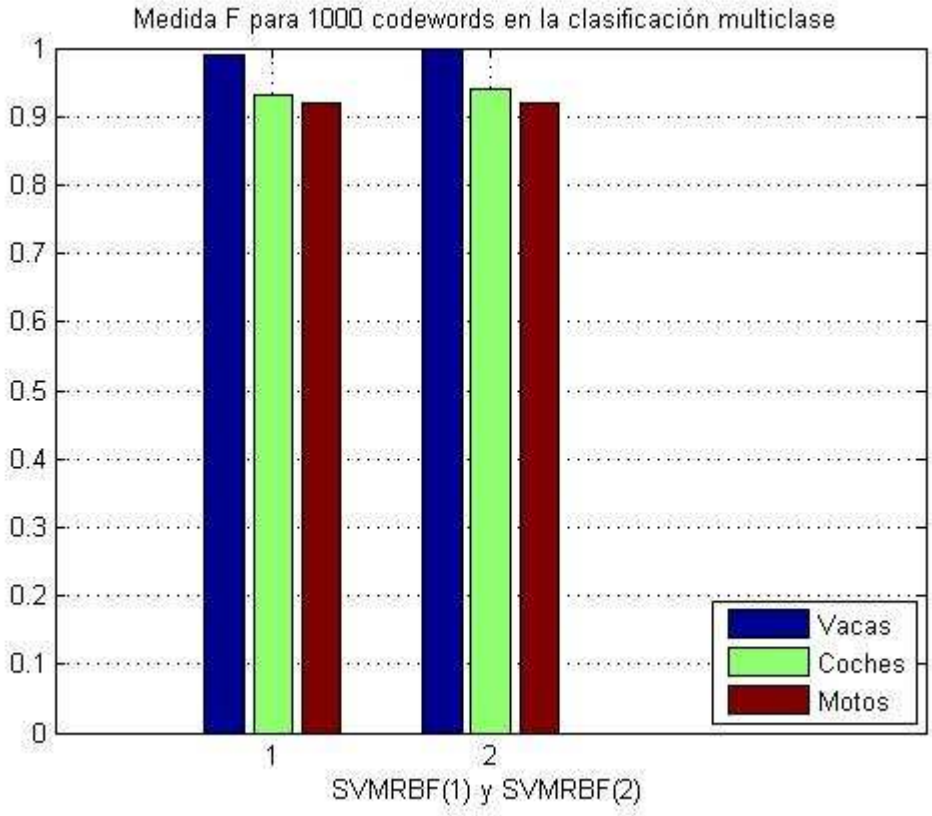

Ilustración 39: "Medida F para 1000 *codewords* en la clasificación multiclase"

En la Tabla 2: "Matriz de confusión para 1000 *codewords* utilizando el predictor SVMKL multiclase" y la Tabla 3: "Matriz de confusión para 1000 *codewords* utilizando el predictor SVMRBF multiclase" no se observan diferencias relevantes entre los dos tipos de clasificadores SVM. En estas tablas, se puede observar con mayor claridad el

mayor solapamiento existente entre las clases coches y motos, explicado a partir de las características visuales que comparten.

Tabla 2: "Matriz de confusión para 1000 *codewords* utilizando el predictor SVMKL multiclase"

MATRIZ DE CONFUSIÓN			
HIPÓTESIS \DECISIONES	Vacas	Motos	Coches
Vacas	46	1	0
Motos	0	48	0
Coches	0	6	34

Tabla 3: "Matriz de confusión para 1000 *codewords* utilizando el predictor SVMRBF multiclase"

MATRIZ DE CONFUSIÓN			
HIPÓTESIS \DECISIONES	Vacas	Motos	Coches
Vacas	47	0	0
Motos	0	48	0
Coches	0	6	34

De todos modos, dados los buenos resultados obtenidos por casi todas las configuraciones, sería necesario trabajar con alguna base de datos más difícil para obtener conclusiones más relevantes.

Cap. 4: Conclusiones y líneas futuras

Tras los resultados obtenidos a partir de las pruebas presentadas en la sección de evaluación, se pueden extraer varias conclusiones acerca del estudio realizado:

La transformada SIFT proporciona unas características muy discriminativas, que tienen robustez frente a cambios de escala y rotaciones, lo cual beneficia a la clasificación, ya que clasificará igualmente bien aunque el objeto no tenga una orientación ideal o esté en otro ángulo. No obstante, la elevada dimensión de dicho descriptor, así como el hecho de trabajar sobre un número variable de parches locales detectados en puntos de interés, obliga a diseñar modelos capaces de manejar dicha variabilidad y con bases de datos anotadas a nivel de imágenes.

Así, el modelo *bag-of-words* permite trabajar con un número de descriptores variable por imagen, ajustándose a las necesidades propias de cada imagen, y a la posible inclusión de descriptores nuevos cada vez. Este modelo proporciona una cierta flexibilidad.

La creación de *codebooks*: es un aspecto crítico ya que la iniciación es aleatoria y nunca se obtendrán resultados idénticos, por lo que conviene realizar muchas inicializaciones para así optar por aquel con mejores resultados.

El tamaño del *codebook* también es un aspecto crítico. Si el *codebook* es pequeño tendrá poco poder expresivo. Según se va aumentando su tamaño, aumentará su poder pero supondrá mayor coste computacional y también aumentan las dimensiones, lo cual es problemático a la hora de entrenar los clasificadores.

En este proyecto se ha experimentado con diferentes métodos de generación de *codebooks* (global, locales y global modificado) y se han observado sus capacidades y limitaciones. En general, se ha visto que la alternativa básica (*codebook* global), además de ser la más lógica y cercana al concepto habitual de un vocabulario, consigue resultados bastante similares al *codebook* global modificado. Los mejores resultados se obtienen empleando el clasificador multicase con el *codebook* global. El *codebook* local

no obtiene buenos resultados para la clasificación de las imágenes pertenecientes a la clase Vacas. Esto no ocurre para las otras dos clases ya que contienen *codewords* más similares y por tanto se asignan mejor en los histogramas.

En cuanto a los clasificadores empleados en el proyecto, los mejores resultados se han obtenido mediante la aproximación basada en máquinas de vectores de soporte (SVM). El perceptrón monocapa ha resultado no disponer de la suficiente capacidad expresiva para el problema a tratar, probablemente debido a la elevada dimensionalidad de los datos (creciente con el tamaño del *codebook*). Entre las alternativas basadas en SVM, no se han observado diferencias significativas entre la utilización de un *kernel* lineal y uno RBF, por lo que el primero sería más adecuado dada su sencillez.

Si bien el coste computacional del preprocesado de las imágenes y del entrenamiento de los clasificadores es considerablemente alto (principalmente debido a la generación de *codebooks* y al entrenamiento de las SVMs), el test tiene tiempos de ejecución aceptables, lo que permitiría su implementación en un sistema real.

La base de datos utilizada es pequeña, pero suficiente para mostrar ciertas diferencias entre las diferentes configuraciones. En todos los casos, incluidas las clasificaciones normales, multiclase y conjunto, la clase mejor clasificada es la de Vacas. Éste es un resultado esperado, ya que es lógico que en imágenes de motos y de coches haya más elementos comunes que con imágenes de vacas (como ruedas, carretera, un fondo de ciudad…) y por tanto se confundan más.

Además de estas conclusiones de carácter técnico, podrían destacarse otras conclusiones de carácter más personal derivadas de la realización del proyecto:

Personalmente, este proyecto me ha servido para conocer más a fondo la complejidad del problema de la clasificación de imágenes por contenido, conocer todo lo que se ha trabajado al respecto, lo que aún resta por investigar, y la multitud de aplicaciones que tiene la recuperación de la información.

En este proyecto se ha experimentado con algunos parámetros abiertos en la aplicación del modelo de *bag-of-words* al reconocimiento de imágenes. No obstante, el trabajo aquí desarrollado admite numerosas líneas de trabajo futuro como las que se detallan a continuación.

Para este proyecto se ha utilizado una base de datos con tres clases y alrededor de unas cien imágenes por clase. Si bien dicha base de datos ha permitido mostrar diferencias entre las distintas configuraciones probadas, se plantea utilizar una base de datos más compleja y realista, con más clases (ej. Caltech 256, con 256 categorías) y con imágenes con una complejidad mayor para detectar los objetos. Una base de datos de estas características se acercaría más al problema real de la clasificación de imágenes y aumentaría las diferencias entre el rendimiento de las distintas propuestas.

El modelo *bag-of-words*, dada su procedencia del análisis de textos, no tiene en cuenta la distribución espacial de los descriptores a lo largo de la imagen (Ej., coches y motos tienen descriptores asociados a las llantas de una rueda, pero en un coche aparecen también descriptores de ventanillas justo encima de la rueda. Relacionar ambos descriptores podría ayudar a discriminar mejor entre coches y motos). Se podrían fusionar la información local obtenida de las palabras visuales y el formato geométrico global dado por una segmentación previa de la imagen.

También se podrían emplear descriptores más complejos: SIFT emplea sólo textura. Se podrían usar color y textura, u otras características combinadas entre sí para mejorar la información de los descriptores.

Dada la elevada dimensionalidad de la transformada SIFT (128 dimensiones), se podrían aplicar técnicas de reducción de dimensionalidad: por ejemplo las técnicas tipo PCA (*Principal Component Analysis*, análisis de componentes principales) determinan el número de factores subyacentes explicativos tras un conjunto de datos, que expliquen la variabilidad de los mismos.

En lo referente a la creación de *codebooks*: existen técnicas más complejas, como la utilización de histogramas blandos de distancias de un descriptor a las palabras del

vocabulario, en vez de meros contadores de aparición (se suman distancias a *codewords* en vez de sumar 1 cada vez en la posición del *codeword* más cercano).

También se podrían aplicar capacidades de regionalización: dividir la imagen en celdas y aplicar los clasificadores en las celdas de forma individual. De todos modos, esta alternativa requeriría bases de datos anotadas a nivel de región (y no de imagen, como la que se ha utilizado).

Cap. 5: Presupuesto

A continuación se detalla el cálculo de los costes aproximados de realización del proyecto descrito en la presente memoria. El presupuesto se dividirá en el cálculo de dos bloques diferenciados, por un lado los costes asociados a materiales empleados y por otro, los costes debidos a honorarios de las personas que han participado en el proyecto.

5.1.1. Coste del material

En este apartado se incluyen tanto los costes de la parte hardware como los de software de cada uno de los elementos que se han empleado.

Un ordenador con un coste aproximado de 600€. Puesto que puede ser reutilizado, su coste puede amortizarse hasta la cantidad de 400€.

Espacio de trabajo con las debidas condiciones de luz, calefacción, mantenimiento, más el mobiliario necesario; tiene un coste asociado de unos 800€/mes. Al tratarse de un laboratorio compartido, tendrá un coste individual asociado de 100€/mes. Puesto que el proyecto ha durado aproximadamente 6 meses, el coste asociado asciende a 600€.

Licencia para utilizar el programa Matlab, con coste estimado de 100€, ya que es un elemento compartido con el departamento.

En la Tabla 4: “Presupuesto de los materiales” están resumidos los costes relacionados con el material empleado.

Tabla 4: “Presupuesto de los materiales”

Material	**Presupuesto**
Ordenador	400€
Lugar de Trabajo	600€
Licencia de Matlab	100€
Total	**1100€**

5.1.2. Coste de honorarios

El coste asociado a los honorarios de las personas que han participado en este proyecto debe dividirse en dos; por un lado el coste asociado al realizador del proyecto, y por otro, los costes correspondientes al a dirección del mismo.

5.1.2.1. Honorarios de realización

Los datos correspondientes a los honorarios pueden extraerse del Colegio Oficial de Ingenieros de Telecomunicación:

Coste de la hora laboral: 72€
Coste de la hora extraordinaria: 84€

A estas cantidades hay que añadir el IVA establecido en el 7%.

El importe correspondiente al trabajo realizado por la autora del proyecto, Sara Pardo Feijoo, con una duración aproximada de 12 meses, una dedicación diaria de 4 horas, y semanal de 5 días, hacen un total de 960 horas laborales. Si se añade el 7% de IVA queda un importe final de 73958 € (IVA incluido); computando el total de horas laborales, sin incluir ninguna hora extraordinaria.

5.1.2.2. Honorarios de dirección

La dirección del proyecto ha corrido a cargo de Iván González Díaz y de Darío García García, y se han considerado unos costes correspondientes a sus honorarios de 15408€, debidos a unas 100 horas trabajadas, retribuidas a 72€ + IVA cada una para cada uno de los tutores.

5.1.3. Presupuesto Final

Al presupuesto global del proyecto, se deben añadir otros costes derivados del mismo difíciles de detallar. Estos costes están referidos a material fungible, costes asociados a trabajadores externos al proyecto…etc. Se cuantificarán estos costes derivados en un 15% del presupuesto final. Se puede ver el desglose de gastos en la Tabla 5: "Presupuesto final".

Tabla 5: "Presupuesto final"

Tipo de gasto	**Presupuesto (€)**
Material	1100
Honorarios Realización	73958
Honorarios Dirección	15408
Total sin costes derivados	90466
Costes derivados (15%)	13567
TOTAL	**104033**

Cap. 6: Apéndices

6.1. Entorno de desarrollo

El entorno de desarrollo utilizado ha sido Matlab, concretamente la versión 7.

6.2. Utilización del programa

Para utilizar el programa, basta con invocar la función PROYECTO.m, indicando como parámetros de entrada el número de *codewords* y el número de iteraciones que se desean realizar. Esa función llama al resto de funciones (explicadas en el apartado 'Implementación del proyecto') en el orden adecuado, y como resultado se tendrán las diferentes gráficas de calidad que servirán para comparar los tres clasificadores entre sí, o los resultados de clasificación obtenidos con un número de *codewords* u otro (de distintas ejecuciones).

Tras invocar la función PROYECTO.m con valores que vayan desde 10 codewords hasta 2000, se puede invocar la función comparativaClasificadores.m, que mostrará una comparativa de todos los resultados obtenidos para así evaluar el proyecto.

6.3. Librería de funciones

6.3.1. VLFeat

VLFeat (VisionLab Features Library) es una colección de algoritmos de visión artificial con un enfoque especial en características de las imágenes como SIFT y MSER. Los algoritmos más importantes son implementados en una biblioteca C (C-90) ligera y portátil y se han hecho accesibles a través de la biblioteca API, programas de líneas de comandos, y programas MATLAB [19][19].

6.3.2. SIFT

La transformación de características invariante a la escala (SIFT) ata un detector y un descriptor de características. Un detector de características toma una imagen I y selecciona un número de *frames* (puntos clave, puntos de interés, regiones), que son regiones de imagen estables (que las mismas regiones son seleccionadas a un conjunto de trasformaciones de imagen prescritas, debido al ruido, cambios en los puntos de vista, en la iluminación, etc.). Un descriptor de características calcula, desde la apariencia local de un *frame*, un descriptor, una etiqueta distintiva que ayuda a identificar el *frame*.

Tanto el detector como el descriptor son accesibles por un programa sencillo de MATLAB llamado sift [19].

6.3.3. LIBSVM

LIBSVM es un software integrado para clasificación con vectores de soporte (C_SVC, nu-SVC), regresión (Epsilon-SVR, nu-SVR) y estimación de distribución (SVM monoclase). Soporta la clasificación multiclase [20]. De esta librería se utilizan las funciones *svmtrain* y *svmpredict*.

6.4. Código fuente

6.4.1. LectorImagenes.m

```
function lectorImagenes (testOTrain)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LECTORIMAGENES (testOTrain)
% La función lectorImagenes calcula los descriptores con el algoritmo
% sift, de todas las imágenes contenidas en las carpetas del conjunto
% de test o de train, según se haya especificado en el parámetro de
% entrada.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

nombreFicheroTexto=['descriptoresVacas',testOTrain,'.txt';'descriptore
sMotos',testOTrain,'.txt';'descriptoresCoche',testOTrain,'.txt'];
nombreCarpeta =
['Imagenes\vacas\',testOTrain;'Imagenes\motos\',testOTrain;'Imagenes\c
oche\',testOTrain];
nombreFicheroNumeroImagenes = ['numeroImagenes',testOTrain,'.mat'];
nombreFicheroDatosDescriptores =
['descriptores/','descriptores',testOTrain,'.mat'];
nombreFicheroDatosTamanos =
['descriptores/','tamanoDescriptores',testOTrain,'.mat'];

%Sólo se calculan los descriptores si no se han calculado antes.
carpetaDescriptores = dir ('descriptores');
if (numel(carpetaDescriptores)<6)
    for i = 1:3     %i=1 es Vacas, i=2 es Motos, i=3 es Coches
        carpetaImagenes = dir (nombreCarpeta(i,:));
        concatenar='\';
        %Se halla el número de archivos de cada carpeta
        numeroArchivos = numel (carpetaImagenes);
        %Se abre el fichero de texto correspondiente
        fi=fopen (nombreFicheroTexto(i,:),'w+');
        %No todos los archivos son imágenes, por lo que se van
        %contando
        numeroImagenes(i)=0;
        for j=1:numeroArchivos
            %Se halla el directorio de cada imagen
            nombreArchivo = getfield (carpetaImagenes (j,1), 'name');
            directorioImagen= [nombreCarpeta(i,:) concatenar
nombreArchivo];
            %Para que sea imagen, debe contener en su nombre 'jpg' o
            %'png'
            if findstr(nombreArchivo, 'jpg')
                imagen =imread (directorioImagen);
                %En el caso de que la imagen sea a color, se pasa a
                % escala de grises
                [x,y,z]=size (imagen);
                if z==3
                    imagen =  rgb2gray(imagen);
                end
                %Se reescalan las imágenes para que sea menos costoso
                %el procesado
```

```
                while (x>300)||(y>300)
                    imagen =imresize (imagen, 0.5);
                    [x,y,z]=size (imagen);
                end
                %Se aplica el algoritmo sift
                imagen = single(imagen);
                [frames, descriptors]=sift (imagen);
                fwrite (fi,descriptors, 'uint8');
                [m,n]=size (descriptors);
                numeroImagenes(i)=numeroImagenes(i)+1;
                tamanoDescriptores (i,numeroImagenes(i))=n;
            elseif findstr(nombreArchivo, 'png')
                imagen =imread (directorioImagen);
                %En el caso de que la imagen sea a color, se pasa a
                %escala de grises
                [x,y,z]=size (imagen);
                if z==3
                    imagen =  rgb2gray(imagen);
                end
                %Se reescalan las imágenes para que sea menos costoso
                %el procesado
                while (x>300)||(y>300)
                    imagen =imresize (imagen, 0.5);
                    [x,y,z]=size (imagen);
                end
                %Se aplica el algoritmo sift
                imagen = single(imagen);
                [frames, descriptors]=sift (imagen);
                fwrite (fi,descriptors, 'uint8');
                [m,n]=size (descriptors);
                numeroImagenes(i)=numeroImagenes(i)+1;
                tamanoDescriptores (i,numeroImagenes(i))=n;
            end
        end
        %Se halla el tamaño de los descriptores para poder leerlos
        %luego del fichero de texto
        tamanoTotalDescriptores (i)= sum (tamanoDescriptores(i,:));

        fclose (fi);
        fi=fopen (nombreFicheroTexto(i,:),'r');
        %Para cada clase, se leen los descriptores del fichero de
        %texto
        descriptores = zeros (tamanoTotalDescriptores(i),128);
        for k=1:tamanoTotalDescriptores (i)
            descriptores (k,:)= fread (fi,128,'uint8');
        end
        errorCerrar(i) = fclose (fi);
        switch i
            case 1
                descriptoresVacas = descriptores;
                numeroImagenesVacas = numeroImagenes(i);

tamanoDescriptoresVacas=tamanoDescriptores(i,1:numeroImagenesVacas);
            case 2
                descriptoresMotos = descriptores;
                numeroImagenesMotos = numeroImagenes(i);

tamanoDescriptoresMotos=tamanoDescriptores(i,1:numeroImagenesMotos);
            case 3
```

```
                descriptoresCoches = descriptores;
                numeroImagenesCoches = numeroImagenes(i);

tamanoDescriptoresCoches=tamanoDescriptores(i,1:numeroImagenesCoches);
        end
    end
    %Se guardan los datos, si no se ha producido ningún error, en
    %archivos .mat
    if (errorCerrar(1)==0 || errorCerrar(2)==0 || errorCerrar(3)==0)
        save
(nombreFicheroDatosDescriptores,'descriptoresCoches','descriptoresMoto
s', 'descriptoresVacas');
        save (nombreFicheroDatosTamanos,
'tamanoDescriptoresMotos','tamanoDescriptoresCoches','tamanoDescriptor
esVacas');
        save (nombreFicheroNumeroImagenes, 'numeroImagenesVacas',
'numeroImagenesMotos', 'numeroImagenesCoches');
        disp ('LOS DESCRIPTORES ESTÁN GUARDADOS EN LA CARPETA
"DESCRIPTORES"');
    else
        error('Error al cerrar algún fichero de texto');
    end

elseif
((numel(carpetaDescriptores)>6)||(numel(carpetaDescriptores)==6))
    disp(['LOS DESCRIPTORES DE ',testOTrain,' YA ESTABAN
CALCULADOS']);
end
```

6.4.2. Clustering.m

```
function [codewords,indices,sumaDistanciasTotal, codewordsConjunto,
indicesConjunto,sumaDistanciasConjuntoTotal]=clustering
(numeroCodewords)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% [codewords,indices,sumaDistanciasTotal, codewordsConjunto,
% indicesConjunto,sumaDistanciasConjuntoTotal]=CLUSTERING
%(numeroCodewords)
%
% La función clustering halla los codewords de todo el conjunto de
% imágenes. Para ello, aplica la función kmeans, del toolbox de
% procesamiento de imágenes de Matlab, a los descriptores de todas las
% imágenes de un conjunto, con tantos centroides como codewords hemos
% especificado. Esto devuelve los codewords, junto con sus índices y
% la suma de las distancias al descriptor en cuestión.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load 'descriptores/descriptoresTrain.mat';
%Se ejecuta el kmeans para todas las clases
disp ('Haciendo el kmeans de vacas');
[indicesVacas,codewordsVacas, sumaDistanciasVacas]=kmeans
(descriptoresVacas,numeroCodewords,'EmptyAction','singleton');
disp ('Hecho el kmeans de vacas');
disp ('Haciendo el kmeans de motos');
[indicesMotos,codewordsMotos, sumaDistanciasMotos]=kmeans
(descriptoresMotos,numeroCodewords,'EmptyAction','singleton');
disp ('Hecho el kmeans de motos');
disp ('Haciendo el kmeans de coches');
[indicesCoches,codewordsCoches, sumaDistanciasCoches]=kmeans
(descriptoresCoches,numeroCodewords,'EmptyAction','singleton');
disp ('Hecho el kmeans de coches');

%Se almacenan los datos en un vector conjunto para utilizarlos en el
%codebook global
codewords= [codewordsVacas;codewordsMotos;codewordsCoches];
indices = [indicesVacas;indicesMotos;indicesCoches];
%Se juntan todos los descriptores para hacer el codebook global
descriptores =
[descriptoresVacas;descriptoresMotos;descriptoresCoches];
%Se ejecuta el kmeans para las clases mezcladas
disp ('Haciendo el kmeans global');
[indicesConjunto, codewordsConjunto, sumaDistanciasConjunto]=kmeans
(descriptores,numeroCodewords,'EmptyAction','singleton');
disp ('Hecho el kmeans global');

%Se calculan las distancias totales
sumaDistanciasConjuntoTotal= sum(sumaDistanciasConjunto);
sumaDistanciasVacasTotal   = sum(sumaDistanciasVacas);
sumaDistanciasMotosTotal   = sum(sumaDistanciasMotos);
sumaDistanciasCochesTotal  = sum(sumaDistanciasCoches);
sumaDistanciasTotal =
[sumaDistanciasVacasTotal;sumaDistanciasMotosTotal;sumaDistanciasCoche
sTotal];
```

6.4.3. CalcularCodebooks.m

```
function calcularCodebooks(numeroCodewords,numeroIteraciones);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CALCULARCODEBOOKS (numeroCodewords, numeroIteraciones)
%
% La función calcularCodebooks calcula los codebooks para todas las
% clases de imágenes tantas veces como se haya especificado en el
% parámetro 'numeroIteraciones', y utilizando tantas codewords como se
% haya especificado en el parámetro 'numeroCodewords'. Todos los datos
% se guardan en archivos .mat en la carpeta codebooks
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Primero se leen todas las imágenes de entrenamiento para hallar sus
%descriptores
lectorImagenes('train');

nC=int2str(numeroCodewords);

carpeta = dir ('codebooks');
numeroArchivos = numel (carpeta);

%Ejecuta el código tantas veces como iteraciones se hayan especificado
%al llamar a la función
for numeroIteracion = 1:numeroIteraciones
    nI=int2str(numeroIteracion);
    calcularCodebook = true;
    for i=1:numeroArchivos
        nombreArchivo = getfield (carpeta (i,1), 'name');
        if (strcmp(nombreArchivo,['codebookGlobal',nC,nI,'.mat']))
            disp ([nI,'º CODEBOOK GLOBAL DE ',nC,' PALABRAS YA ESTABA
GENERADO EN LA CARPETA "CODEBOOKS"']);
            calcularCodebook = false;
        elseif (strcmp(nombreArchivo ,['codebookLocal',nC,nI,'.mat']))
            disp ([nI,'º CODEBOOK LOCAL DE ',nC,' PALABRAS YA ESTABA
GENERADO EN LA CARPETA "CODEBOOKS"']);
            calcularCodebook = false;
        end
    end
    if (calcularCodebook)

[codewordsLocal,indicesLocal,distanciasLocal,codewordsGlobal,indicesGl
obal,distanciasGlobal]=clustering (numeroCodewords);
        archivo1 = ['codebooks/codebookGlobal',nC,nI,'.mat'];
        archivo2 = ['codebooks/codebookLocal',nC,nI,'.mat'];

        save
(archivo1,'codewordsGlobal','indicesGlobal','distanciasGlobal');
        save
(archivo2,'codewordsLocal','indicesLocal','distanciasLocal');
        disp ([nI,'º CODEBOOK DE ',nC,' PALABRAS GENERADOS EN LA
CARPETA "CODEBOOKS"']);
    end
end
```

6.4.4. CalcularIteracionOptima.m

```
function calcularIteracionOptima (numeroCodewords,numeroIteraciones);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CALCULARITERACIONOPTIMA (numeroCodewords)
% La función calculariteracionoptima busca los codebooks calculados
% para el número de codewords especificado en el parámetro, y halla
% cuál de ellos ha obtenido una menor distancia a las codewords.
% El que tenga la menor
% distancia será el óptimo
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Estas variables son strings que se compararán con el nombre los
% archivos encontrados en la carpeta, y si coinciden (por el número
% de codewords) se prosigue
archivoGlobal = ['codebooks/codebookGlobal',int2str(numeroCodewords)];
archivoLocal = ['codebooks/codebookLocal',int2str(numeroCodewords)];

for i=1:numeroIteraciones
   nombreArchivoGlobal = [archivoGlobal,int2str(i),'.mat'];
   load (nombreArchivoGlobal);
   %Se almacena el valor de las distancias en un vector
   distanciasGlobalIteraciones (i,:) = distanciasGlobal;

   nombreArchivoLocal = [archivoLocal,int2str(i),'.mat'];
   load (nombreArchivoLocal);
   %Se almacena el valor de las distancias en un vector
   distanciasLocalIteraciones (i,:) = distanciasLocal;
end

%Se calcula el mínimo de todas las distancias y a la vez se obtiene la
%iteración que ha dado un óptimo resultado
[minimaDistanciaVacas,  iteracionVacas]  = min
(distanciasLocalIteraciones(:,1));
[minimaDistanciaMotos,  iteracionMotos]  = min
(distanciasLocalIteraciones(:,2));
[minimaDistanciaCoches, iteracionCoches] = min
(distanciasLocalIteraciones(:,3));
[minimaDistanciaGlobal, iteracionGlobal] = min
(distanciasGlobalIteraciones);

load 'descriptores/descriptoresTrain.mat';
numeroDescriptoresVacas  = length (descriptoresVacas);
numeroDescriptoresCoches = length (descriptoresCoches);
numeroDescriptoresMotos  = length (descriptoresMotos);

%Se carga el codebook adecuado para la iteración de la clase vacas que
%sea óptima. Se guardan en otras variables las codewords y los índices
%óptimos
nombreArchivoCodebookOptimo =
['codebooks/codebookLocal',int2str(numeroCodewords),int2str(iteracionV
acas),'.mat'];
load (nombreArchivoCodebookOptimo);
codewordsOptimoVacas = codewordsLocal(1:numeroCodewords,:);
```

```
indicesOptimoVacas = indicesLocal (1:numeroDescriptoresVacas);

%Se carga el codebook adecuado para la iteración de la clase motos que
%sea óptima. Se guardan en otras variables las codewords y los índices
%óptimos
nombreArchivoCodebookOptimo=['codebooks/codebookLocal',int2str(numeroC
odewords),int2str(iteracionMotos),'.mat'];
load (nombreArchivoCodebookOptimo);
codewordsOptimoMotos =
codewordsLocal((numeroCodewords+1):(numeroCodewords*2),:);
indicesOptimoMotos = indicesLocal
((numeroDescriptoresVacas+1):(numeroDescriptoresMotos+numeroDescriptor
esVacas+1));

%Se carga el codebook adecuado para la iteración de la clase coches
%que sea óptima. Se guardan en otras variables las codewords y los
%índices óptimos
nombreArchivoCodebookOptimo =
['codebooks/codebookLocal',int2str(numeroCodewords),int2str(iteracionC
oches),'.mat'];
load (nombreArchivoCodebookOptimo);
codewordsOptimoCoches = codewordsLocal((2*numeroCodewords+1):end,:);
indicesOptimoCoches = indicesLocal
((numeroDescriptoresMotos+numeroDescriptoresVacas+1):end);

%Se carga el codebook adecuado para la iteración de todas las clases
%que sea óptima. Se guardan en otras variables las codewords y los
%índices óptimos
nombreArchivoCodebookOptimo =
['codebooks/codebookGlobal',int2str(numeroCodewords),int2str(iteracion
Global),'.mat'];
load (nombreArchivoCodebookOptimo);
codewordsOptimoGlobal = codewordsGlobal;
indicesOptimoGlobal= indicesGlobal;

archivo =
['codebooksOptimos/codebookOptimo',int2str(numeroCodewords),'.mat'];

save
(archivo,'codewordsOptimoVacas','codewordsOptimoMotos','codewordsOptim
oCoches','indicesOptimoVacas','indicesOptimoCoches','indicesOptimoMoto
s','codewordsOptimoGlobal','indicesOptimoGlobal');
disp ('SE HAN CALCULADO LAS ITERACIONES ÓPTIMAS');
```

6.4.5. AsignacionCodewords.m

```
function asignacionCodewords (testOTrain,numeroCodewords);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ASIGNACIONCODEWORDS (testOTrain,numeroCodewords)
% La función asignacionCodewords calcula las distancias eucídeas entre
% los descriptores de cada imagen y las codewords óptimas que se han
% calculado. De todas esas distancias, se hallará la mínima, y se
% asociará el descriptor al codeword que haya dado la mínima distancia
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Se establece el nombre de los archivos en función de que sean
% de test o de train y del número de codewords y se cargan
switch testOTrain
    case 'Test'
        lectorImagenes ('Test');
end
load (['descriptores/descriptores',testOTrain,'.mat']);
load (['descriptores/tamanoDescriptores',testOTrain,'.mat']);
load (['numeroImagenes',testOTrain,'.mat']);
load
(['codebooksOptimos/codebookOptimo',int2str(numeroCodewords),'.mat']);

archivoGuardar = ['histogramas/codewordsAsignados',testOTrain,
int2str(numeroCodewords),'.mat'];

%Código referido a la clase Vacas
primerValor = 1;
ultimoValor = 0;
% Se asigna, para cada imagen, los descriptores a la codeword que esté
% a menor distancia
for i=1:numeroImagenesVacas
    % Se extraen los descriptores de la imagen i de la matriz
    % descriptoresVacas
    numeroDescriptoresVacas = tamanoDescriptoresVacas(i);
    primerValor = ultimoValor+1;
    ultimoValor = numeroDescriptoresVacas + ultimoValor;
    descriptor = descriptoresVacas(primerValor:ultimoValor,:);
    % Cálculo para cada descriptor y cada codeword de la distancia
    % euclídea
    for k=1:numeroDescriptoresVacas
        for j=1:numeroCodewords
            distanciasVacasLocal (j) = sum ((codewordsOptimoVacas(j,:)
- descriptor(k,:)).^2);
            distanciasVacasGlobal (j) = sum
((codewordsOptimoGlobal(j,:) - descriptor(k,:)).^2);
        end
        % Se halla el codeword con mínima distancia
        [distancia,codeword] = min (distanciasVacasLocal);
        codewordAsignadoVacasLocal (k,i) = codeword;
        [distancia,codeword] = min (distanciasVacasGlobal);
        codewordAsignadoVacasGlobal (k,i)=codeword;
    end
end
```

```
%Código referido a la clase Motos
primerValor = 1;
ultimoValor = 0;
for i=1:numeroImagenesMotos
    % Se extrae el descriptor adecuado de la matriz descriptoresMotos
    numeroDescriptoresMotos = tamanoDescriptoresMotos(i);
    primerValor = ultimoValor+1;
    ultimoValor = numeroDescriptoresMotos + ultimoValor;
    descriptor = descriptoresMotos(primerValor:ultimoValor,:);
    % Cálculo para cada descriptor y cada codeword de la distancia
    % euclídea
    for k=1:numeroDescriptoresMotos
        for j=1:numeroCodewords
            distanciasMotosLocal (j) = sum ((codewordsOptimoMotos(j,:)
- descriptor(k,:)).^2);
            distanciasMotosGlobal (j) = sum
((codewordsOptimoGlobal(j,:) - descriptor(k,:)).^2);
        end
        % Se halla el codeword con mínima distancia
        [distancia,codeword] = min (distanciasMotosLocal);
        codewordAsignadoMotosLocal (k,i) = codeword;
        [distancia,codeword] = min (distanciasMotosGlobal);
        codewordAsignadoMotosGlobal (k,i)=codeword;
    end
end

%Código referido a la clase Coches
primerValor = 1;
ultimoValor = 0;
for i=1:numeroImagenesCoches
    % Se extrae el descriptor adecuado de la matriz descriptoresCoches
    numeroDescriptoresCoches = tamanoDescriptoresCoches(i);
    primerValor = ultimoValor+1;
    ultimoValor = numeroDescriptoresCoches + ultimoValor;
    descriptor = descriptoresCoches(primerValor:ultimoValor,:);
    % Cálculo para cada descriptor y cada codeword de la distancia
    % euclídea
    for k=1:numeroDescriptoresCoches
        for j=1:numeroCodewords
            distanciasCochesLocal (j) = sum
((codewordsOptimoCoches(j,:) - descriptor(k,:)).^2);
            distanciasCochesGlobal (j) = sum
((codewordsOptimoGlobal(j,:) - descriptor(k,:)).^2);
        end
        % Se halla el codeword con mínima distancia
        [distancia,codeword] = min (distanciasCochesLocal);
        codewordAsignadoCochesLocal (k,i) = codeword;
        [distancia,codeword] = min (distanciasCochesGlobal);
        codewordAsignadoCochesGlobal (k,i)=codeword;
    end
end

save
(archivoGuardar,'codewordAsignadoVacasLocal','codewordAsignadoCochesLo
cal','codewordAsignadoMotosLocal','codewordAsignadoVacasGlobal','codew
ordAsignadoCochesGlobal','codewordAsignadoMotosGlobal');
disp ('LOS CODEWORDS ESTÁN ASIGNADOS');
```

6.4.6. Histograma.m

```
function histograma (testOTrain,numeroCodewords);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% HISTOGRAMA (testOTrain,numeroCodewords)
% La función histograma calcula los histogramas correspondientes a los
% codewords asignados y los dibuja por pantalla, para así hallar las
% conclusiones oportunas
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

nC = int2str(numeroCodewords);
%Según el valor de los parámetros, se cargan unos datos u otros
archivoCargar1=
['histogramas\codewordsAsignados',testOTrain,nC,'.mat'];
archivoCargar2 = 'numeroImagenes',testOTrain,'.mat';
load 'descriptores\tamanoDescriptores',testOTrain,'.mat';
archivoGuardar = ['histogramas\histograma',testOTrain,nC,'.mat'];

load (archivoCargar1);
load (archivoCargar2);

% Cálculo de los histogramas para la clase vacas
for i=1:numeroImagenesVacas
    descriptoresDistintosCeroVacasLocal  = find
(codewordAsignadoVacasLocal(:,i));
    descriptoresDistintosCeroVacasGlobal = find
(codewordAsignadoVacasGlobal(:,i));
    histogramaVacasLocal (i,:) = (hist(codewordAsignadoVacasLocal
(descriptoresDistintosCeroVacasLocal
,i),1:numeroCodewords))/tamanoDescriptoresVacas(i);
    histogramaVacasGlobal(i,:) = (hist(codewordAsignadoVacasGlobal
(descriptoresDistintosCeroVacasGlobal,i),1:numeroCodewords))/tamanoDes
criptoresVacas(i);
end

% Cálculo de los histogramas para la clase Motos
for i=1:numeroImagenesMotos
    descriptoresDistintosCeroMotosLocal  = find
(codewordAsignadoMotosLocal(:,i));
    descriptoresDistintosCeroMotosGlobal = find
(codewordAsignadoMotosGlobal(:,i));
    histogramaMotosLocal (i,:) = (hist(codewordAsignadoMotosLocal
(descriptoresDistintosCeroMotosLocal
,i),1:numeroCodewords))/tamanoDescriptoresMotos(i);
    histogramaMotosGlobal(i,:) = (hist(codewordAsignadoMotosGlobal
(descriptoresDistintosCeroMotosGlobal,i),1:numeroCodewords))/tamanoDes
criptoresMotos(i);
end

% Cálculo de los histogramas para la clase Coches
for i=1:numeroImagenesCoches
    descriptoresDistintosCeroCochesLocal  = find
(codewordAsignadoCochesLocal(:,i));
```

```
    descriptoresDistintosCeroCochesGlobal = find
(codewordAsignadoCochesGlobal(:,i));
    histogramaCochesLocal  (i,:)= (hist(codewordAsignadoCochesLocal
(descriptoresDistintosCeroCochesLocal,
i),1:numeroCodewords))/tamanoDescriptoresCoches(i);
    histogramaCochesGlobal (i,:)= (hist(codewordAsignadoCochesGlobal
(descriptoresDistintosCeroCochesGlobal,i),1:numeroCodewords))/tamanoDe
scriptoresCoches(i);
end

save
(archivoGuardar,'histogramaVacasLocal','histogramaVacasGlobal','histog
ramaMotosLocal','histogramaMotosGlobal','histogramaCochesLocal','histo
gramaCochesGlobal');

hold on;
%Se dibujan todos los histogramas
subplot (2,3,1),bar(1:numeroCodewords,histogramaVacasGlobal');
axis ([1 numeroCodewords 0 0.1]);
xlabel ('Número de codeword');
ylabel ('Frecuencia');
title(['Histograma de ',testOTrain,' global para vacas con ',nC,'
codewords']);

subplot (2,3,2),bar(1:numeroCodewords,histogramaMotosGlobal');
axis ([1 numeroCodewords 0 0.1]);
xlabel ('Número de codeword');
ylabel ('Frecuencia');
title(['Histograma de ',testOTrain,' global para motos con ',nC,'
codewords']);

subplot (2,3,3),bar(1:numeroCodewords,histogramaCochesGlobal');
axis ([1 numeroCodewords 0 0.1]);
xlabel ('Número de codeword');
ylabel ('Frecuencia');
title(['Histograma de ',testOTrain,' global para coches con ',nC,'
codewords']);

subplot (2,3,4),bar(1:numeroCodewords,histogramaVacasLocal');
axis ([1 numeroCodewords 0 0.1]);
xlabel ('Número de codeword');
ylabel ('Frecuencia');
title(['Histograma de ',testOTrain,' local para vacas con ',nC,'
codewords']);

subplot (2,3,5),bar(1:numeroCodewords,histogramaMotosLocal');
axis ([1 numeroCodewords 0 0.1]);
xlabel ('Número de codeword');
ylabel ('Frecuencia');
title(['Histograma de ',testOTrain,' local para motos con ',nC,'
codewords']);

subplot (2,3,6),bar(1:numeroCodewords,histogramaCochesLocal');
axis ([1 numeroCodewords 0 0.1]);
xlabel ('Número de codeword');
ylabel ('Frecuencia');
title(['Histograma de ',testOTrain,' local para coches con ',nC,'
codewords']);
```

6.4.7. CalcularCodebookConjunto.m

```
function calcularCodebookConjunto (numeroCodewords,
numeroIteraciones);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function CALCULARCODEBOOKCONJUNTO (numeroCodewords,
% numeroIteraciones);
%
% La función CALCULARCODEBOOKCONJUNTO hace un codebook mezclando las
% codewords de vacas, coches y motos. Con ese codebook se realizan
% todas las acciones pertinentes hasta el cálculo de medidas de
% calidad
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% CÁLCULO DE LOS CODEBOOKS
nC = int2str(numeroCodewords);
archivoCargar1  = ['codebooksOptimos\codebookOptimo',nC,'.mat'];
load (archivoCargar1);
codewordsTotal = [codewordsOptimoVacas; codewordsOptimoCoches;
codewordsOptimoMotos];

for i=1:numeroIteraciones
    nI = int2str(i);
    archivoGuardar1 = ['codebooks\codebookConjunto',nC,nI,'.mat'];
    [indices,codewords,distancias]=kmeans
(codewordsTotal,numeroCodewords,'EmptyAction','singleton');
    save (archivoGuardar1, 'indices', 'codewords','distancias');
    disp(['Ya está calculado el ',nI,'º codebook']);
    distanciasConjunto(i) = sum(distancias);
    indicesConjunto   (i,:) = indices;
end

% ITERACIÓN ÓPTIMA
%Se calcula el mínimo de todas las distancias y a la vez se obtiene la
%iteración que ha dado un óptimo resultado
[minimaDistanciaConjunto, iteracionConjunto] = min
(distanciasConjunto);

nIO = int2str (iteracionConjunto);
archivoCargar2 =  ['codebooks\codebookConjunto',nC,nIO,'.mat'];
load (archivoCargar2);
codewordOptimoConjunto = codewords;
indicesOptimoConjunto = indicesConjunto (iteracionConjunto,:);

archivoGuardar2 =
['codebooksOptimos/codebookConjuntoOptimo',int2str(numeroCodewords),'.
mat'];
save (archivoGuardar2
,'codewordOptimoConjunto','indicesOptimoConjunto');
disp ('SE HAN CALCULADO LAS ITERACIONES ÓPTIMAS');

% ASIGNACIÓN DE CODEWORDS + HISTOGRAMA ENTRENAMIENTO
for t = 1:2 % t=1 es el train.t=2 es el test
    if t==1
```

```
        load 'descriptores/descriptoresTrain.mat';
        load 'descriptores/tamanoDescriptoresTrain.mat';
        load 'numeroImagenesTrain.mat';
        archivoGuardar3 =
['histogramas/codewordsAsignadosConjuntoTrain',
int2str(numeroCodewords),'.mat'];
        archivoGuardar4 =
['histogramas\histogramaConjuntoTrain',nC,'.mat'];
    elseif t==2
        load 'descriptores/descriptoresTest.mat';
        load 'descriptores/tamanoDescriptoresTest.mat';
        load 'numeroImagenesTest.mat';
        archivoGuardar3 =
['histogramas/codewordsAsignadosConjuntoTest',
int2str(numeroCodewords),'.mat'];
        archivoGuardar4 =
['histogramas\histogramaConjuntoTest',nC,'.mat'];
    end
    numeroImagenes =
[numeroImagenesVacas;numeroImagenesMotos;numeroImagenesCoches];

    % Se asigna, para cada imagen, los descriptores a la codeword que
    % esté a menor distancia
    for clase = 1:3
        primerValor = 1;
        ultimoValor = 0;
        switch clase
            case 1
                tamanoDescriptores = tamanoDescriptoresVacas;
                descriptores = descriptoresVacas;
            case 2
                tamanoDescriptores = tamanoDescriptoresMotos;
                descriptores = descriptoresMotos;
            case 3
                tamanoDescriptores = tamanoDescriptoresCoches;
                descriptores = descriptoresCoches;
        end
        for i=1:numeroImagenes(clase)
            % Se extraen los descriptores de la imagen i de la matriz
            % descriptores
            primerValor = ultimoValor+1;
            ultimoValor = tamanoDescriptores(i) + ultimoValor;
            descriptor = descriptores(primerValor:ultimoValor,:);
            % Cálculo para cada descriptor y cada codeword de la
            % distancia euclídea
            for k=1:tamanoDescriptores(i)
                for j=1:numeroCodewords
                    distanciasConjunto (clase,j) = sum
((codewordOptimoConjunto(j,:) - descriptor(k,:)).^2);
                end
                % Se halla el codeword con mínima distancia
                [distancia,codeword] = min
(distanciasConjunto(clase,:));
                codewordAsignadoConjunto (clase,k,i) = codeword;
            end
            descriptoresDistintosCeroConjunto  = find
(codewordAsignadoConjunto(clase,:,i));
            if clase == 1 %Vacas
```

```
                histogramaVacasConjunto(i,:)  =
(hist(codewordAsignadoConjunto
(clase,descriptoresDistintosCeroConjunto,i),1:numeroCodewords))/tamano
Descriptores(i);
            elseif clase == 2 %Motos
                histogramaMotosConjunto(i,:)  =
(hist(codewordAsignadoConjunto
(clase,descriptoresDistintosCeroConjunto,i),1:numeroCodewords))/tamano
Descriptores(i);
            elseif clase == 3 %Coches
                histogramaCochesConjunto(i,:) =
(hist(codewordAsignadoConjunto
(clase,descriptoresDistintosCeroConjunto,i),1:numeroCodewords))/tamano
Descriptores(i);
            end
        end
    end
    codewordAsignadoVacasConjunto  = codewordAsignadoConjunto (1,:,:);
    codewordAsignadoCochesConjunto = codewordAsignadoConjunto (2,:,:);
    codewordAsignadoMotosConjunto  = codewordAsignadoConjunto (3,:,:);
    save
(archivoGuardar3,'codewordAsignadoVacasConjunto','codewordAsignadoCoch
esConjunto','codewordAsignadoMotosConjunto');
    save
(archivoGuardar4,'histogramaVacasConjunto','histogramaMotosConjunto','
histogramaCochesConjunto');
    if t==1
        disp ('LOS CODEWORDS DE TRAIN ESTÁN ASIGNADOS');
    elseif t==2
        disp ('LOS CODEWORDS DE TEST ESTÁN ASIGNADOS');
    end
    clear histogramaVacasConjunto;
    clear histogramaMotosConjunto;
    clear histogramaCochesConjunto;
end
```

6.4.8. EntrenamientoClasificador.m

```
function entrenamientoClasificador (numeroCodewords, clasificador,
codebook);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ENTRENAMIENTOCLASIFICADOR (numeroCodewords, clasificador, codebook)
% La función entrenamientoclasificador crea las redes neuronales y las
% entrena para hallar las clasificaciones mediante los histogramas
% calculados anteriormente. En función del parámetro clasificador,
% se usará un perceptrón monocapa (si tipo vale 'slp'), una maquina
% de vector soporte tipo kernel lineal (si tipo vale 'svmkl') o una
% máquina de vector soporte tipo rbf (si tipo vale 'svmrbf'). El
% parámetro codebook determina si queremos utilizar el 'Global', el
% 'Local', el 'Multiclase' o el 'Conjunto'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Se establecen los nombres de los archivos necesarios
nC = int2str (numeroCodewords);
numeroCalculos = 0;

switch codebook
    case 'Conjunto'
        archivoCargar =
['histogramas/histogramaConjuntoTrain',nC,'.mat'];
    otherwise
        archivoCargar = ['histogramas/histogramaTrain',nC,'.mat'];
end

archivoGuardar =
['clasificadores/redes',clasificador,codebook,nC,'.mat'];
load (archivoCargar);
load 'numeroImagenesTrain.mat';

%Las etiquetas son vectores que contienen unos en las posiciones
%correspondientes a la clase a clasificar y ceros en el resto
unosVacasTrain   = ones  (1,numeroImagenesVacas);
unosCochesTrain  = ones  (1,numeroImagenesCoches);
unosMotosTrain   = ones  (1,numeroImagenesMotos);
cerosVacasTrain  = zeros (1,numeroImagenesVacas);
cerosCochesTrain = zeros (1,numeroImagenesCoches);
cerosMotosTrain  = zeros (1,numeroImagenesMotos);

etiquetasVacasTrain  = [unosVacasTrain  cerosMotosTrain
cerosCochesTrain];
etiquetasCochesTrain = [cerosVacasTrain cerosMotosTrain
unosCochesTrain];
etiquetasMotosTrain  = [cerosVacasTrain unosMotosTrain
cerosCochesTrain];

%Código para los histogramas globales
switch codebook
    case 'Conjunto'
```

```
        histogramaTrain =
[histogramaVacasConjunto;histogramaMotosConjunto;histogramaCochesConju
nto];
    case 'Local'
        histogramaTrain = [histogramaVacasLocal;
histogramaMotosLocal;   histogramaCochesLocal];
    otherwise
        histogramaTrain = [histogramaVacasGlobal;
histogramaMotosGlobal;  histogramaCochesGlobal];
end

%A la red hay que pasarle, como parámetros de entrada, una matriz con
% los valores máximos y mínimos del histograma para cada codeword, y
% el número de neuronas para el perceptron, en nuestro caso 1
minimoHistogramaTrain = min (histogramaTrain);
maximoHistogramaTrain = max (histogramaTrain);
elementosEntradaTrain = [minimoHistogramaTrain;maximoHistogramaTrain];

switch clasificador
    case 'SLP'
        net = newff (elementosEntradaTrain',[1]);
        %Se establece el parámetro de epochs como 1500 para que se
        % alcancen los objetivos
        net.trainParam.epochs=1500;
        %Se hallan las redes finales entrenando con los histogramas
        netVacas  = train (net,histogramaTrain',etiquetasVacasTrain);
        netMotos  = train (net,histogramaTrain',etiquetasMotosTrain);
        netCoches = train (net,histogramaTrain',etiquetasCochesTrain);
        %Se cierran las figuras del entrenamiento
        close;
        %Se guardan los datos
        save (archivoGuardar, 'netVacas','netCoches','netMotos');

    case 'SVMKL'
        switch codebook
            case 'Multiclase'
                etiquetasTrain = [ones(1,numeroImagenesVacas)
2*ones(1,numeroImagenesMotos) 3*ones(1,numeroImagenesCoches)];
                %Se hace la validación cruzada
                %Barrido lineal del parámetro C
                minimoC = 0.0001;
                maximoC = 20;
                indiceCBienCalculado = false;
                while
(indiceCBienCalculado==false)&&(numeroCalculos<10)
                    numeroCalculos = numeroCalculos +1;
                    c = linspace(minimoC,maximoC,20);
                    for i=1:20
                        parametros = sprintf('-s 0 -t 0 -c %i -v
5',c(i));
                        precisionEntrenamiento (i)  = svmtrain
(etiquetasTrain',  histogramaTrain, parametros);
                    end

                    %Se encuentran los parámetros c óptimos para cada
                    %caso
                    [precisionMaxima,indiceC]   =
max(precisionEntrenamiento);
                    cOptimo = c(indiceC);
```

```
                    save
(['cyg\cyg',clasificador,codebook,nC,'.mat'],'indiceC','cOptimo','mini
moC','maximoC');
                    if (mean(indiceC)<5)
                        msgbox('El parámetro C corresponde a un índice
muy bajo. Se va a recalcular.');
                        minimoC = minimoC/2;
                        maximoC = minimoC*2;
                        indiceCBienCalculado = false;
                    elseif (mean(indiceC)>15)
                        msgbox('El parámetro C corresponde a un índice
muy alto. Se va a recalcular.');
                        minimoC = maximoC/2;
                        maximoC = maximoC*2;
                        indiceCBienCalculado = false;
                    else
                        indiceCBienCalculado = true;
                    end
                end
                %Se obtiene el modelo de la red con el c óptimo
                cadena = sprintf ('-s 0 -t 0 -c %i',cOptimo);
                net  = svmtrain (etiquetasTrain',  histogramaTrain,
cadena);
                save (archivoGuardar, 'net');
            otherwise
                %Se hace la validación cruzada
                %Barrido lineal del parámetro C
                indiceCBienCalculado = false;
                switch codebook
                    case 'Local'
                        minimoC = 0.1;
                        maximoC = 50;
                    otherwise
                        minimoC = 0.1;
                        maximoC = 50;
                end
                while
(indiceCBienCalculado==false)&&(numeroCalculos<10)
                    numeroCalculos = numeroCalculos +1;
                    c = linspace (minimoC,maximoC, 20);
                    precisionVacasEntrenamiento=zeros(1,20);
                    precisionMotosEntrenamiento=zeros(1,20);
                    precisionCochesEntrenamiento=zeros(1,20);
                    for i=1:20
                        parametros = sprintf('-s 0 -t 0 -c %i -v
5',c(i));
                        precisionVacasEntrenamiento (i)  = svmtrain
(etiquetasVacasTrain',  histogramaTrain, parametros);
                        precisionMotosEntrenamiento (i)  = svmtrain
(etiquetasMotosTrain',  histogramaTrain, parametros);
                        precisionCochesEntrenamiento(i)  = svmtrain
(etiquetasCochesTrain', histogramaTrain, parametros);
                    end

                    %Se encuentran los parámetros c óptimos para cada
                    %caso
                    [precisionMaximaVacas, indiceC(1)] =
max(precisionVacasEntrenamiento);
```

```
                    [precisionMaximaMotos, indiceC(2)] =
max(precisionMotosEntrenamiento);
                    [precisionMaximaCoches,indiceC(3)] =
max(precisionCochesEntrenamiento);
                    cOptimo = c(indiceC);
                    save
(['cyg\cyg',clasificador,codebook,nC,'.mat'],'indiceC','cOptimo','mini
moC','maximoC');
                    if (mean(indiceC)<5)
                        msgbox('El parámetro C corresponde a un índice
muy bajo. Se va a recalcular.');
                        minimoC = minimoC/2;
                        maximoC = minimoC*2;
                        indiceCBienCalculado = false;
                    elseif (mean(indiceC)>15)
                        msgbox('El parámetro C corresponde a un índice
muy alto. Se va a recalcular.');
                        minimoC = maximoC/2;
                        maximoC = 2*maximoC;
                        indiceCBienCalculado = false;
                    else
                        indiceCBienCalculado = true;
                    end
                end
                %Se obtiene el modelo de la red con el c óptimo
                cadenaVacas  = sprintf ('-s 0 -t 0 -c %i',cOptimo(1));
                cadenaMotos  = sprintf ('-s 0 -t 0 -c %i',cOptimo(2));
                cadenaCoches = sprintf ('-s 0 -t 0 -c %i',cOptimo(3));
                netVacas  = svmtrain (etiquetasVacasTrain',
histogramaTrain, cadenaVacas);
                netMotos  = svmtrain (etiquetasMotosTrain',
histogramaTrain, cadenaMotos);
                netCoches = svmtrain (etiquetasCochesTrain',
histogramaTrain, cadenaCoches);
                save (archivoGuardar,
'netVacas','netCoches','netMotos');
        end
    case 'SVMRBF'
        switch codebook
            case 'Multiclase'
                %Barrido lineal de parámetros
                minimoG = 0.1;
                maximoG = 5;
                minimoC = 0.1;
                maximoC = 50;
                indiceGBienCalculado = false;
                indiceCBienCalculado = false;
                while (indiceCBienCalculado==false ||
indiceGBienCalculado==false)&&(numeroCalculos<10)
                    numeroCalculos = numeroCalculos +1;
                    g = linspace(minimoG,maximoG,20);
                    c = linspace(minimoC,maximoC,20);
                    etiquetasTrain = [ones(1,numeroImagenesVacas)
2*ones(1,numeroImagenesMotos) 3*ones(1,numeroImagenesCoches)];
                    precisionEntrenamiento=zeros(20,20);
                    for i=1:20
                        %Barrido logarítmico del parámetro g
                        for j=1:20
                            parametros = sprintf('-s 0 -t 2 -c %i -g
%i -v 5',c(j),g(i));
```

```
                            precisionEntrenamiento(i,j,:)  = svmtrain
(etiquetasTrain',  histogramaTrain, parametros);
                        end
                    end

                    %Se encuentran los parámetros c óptimos para cada
                    %caso
                    [precisionMaxima,indiceC]   = max
(max(precisionEntrenamiento));
                    indiceG = find
(precisionEntrenamiento(:,indiceC)==precisionMaxima,1);
                    cOptimo = c(indiceC);
                    gOptimo = g(indiceG);
                    save
(['cyg\cyg',clasificador,codebook,nC,'.mat'],'indiceC','indiceG','cOpt
imo','gOptimo','minimoC','maximoC','minimoG','maximoG');
                    if (mean(indiceC)<5)
                        msgbox('El parámetro C corresponde a un índice
muy bajo. Se va a recalcular.');
                        minimoC = minimoC/2;
                        maximoC = minimoC*2;
                        indiceCBienCalculado = false;
                    elseif (mean(indiceC)>15)
                        msgbox('El parámetro C corresponde a un índice
muy alto. Se va a recalcular.');
                        minimoC = maximoC/2;
                        maximoC = 2*maximoC;
                        indiceCBienCalculado = false;
                    else
                        indiceCBienCalculado = true;
                    end
                    if (mean(indiceG)<5)
                        msgbox('El parámetro G corresponde a un índice
muy bajo. Se va a recalcular.');
                        minimoG = minimoG/2;
                        maximoG = minimoG*2;
                        indiceGBienCalculado = false;
                    elseif (mean(indiceG)>15)
                        msgbox('El parámetro G corresponde a un índice
muy alto. Se va a recalcular.');
                        minimoG = maximoG/2;
                        maximoG = 2*maximoG;
                        indiceGBienCalculado = false;
                    else
                        indiceGBienCalculado = true;
                    end
                end
                %Se obtiene el modelo de la red con g y c óptimos
                cadena = sprintf('-s 0 -t 2 -g %i -c
%i',gOptimo,cOptimo);
                net  = svmtrain (etiquetasTrain',  histogramaTrain,
cadena);
                save (archivoGuardar, 'net');

            otherwise
                indiceGBienCalculado = false;
                indiceCBienCalculado = false;
                switch codebook
                    case 'Global'
```

```
                        minimoG = -7;
                        maximoG = 1;
                        minimoC = -3;
                        maximoC = 3.5;
                    case 'Local'
                        minimoG = -4;
                        maximoG = 1;
                        minimoC = -2;
                        maximoC = 2.6;
                    case 'Conjunto'
                        minimoG = -5;
                        maximoG = 1;
                        minimoC = -3;
                        maximoC = 3;
                end
                while (indiceCBienCalculado==false ||
indiceGBienCalculado==false)&&(numeroCalculos<10)
                    numeroCalculos = numeroCalculos +1;
                    g = logspace (minimoG,maximoG,20);
                    c = logspace (minimoC,maximoC,20);

                    precisionVacasEntrenamiento  =
zeros(length(g),length(c));
                    precisionMotosEntrenamiento  =
zeros(length(g),length(c));
                    precisionCochesEntrenamiento =
zeros(length(g),length(c));
                    for i=1:length(g)
                        %Barrido logarítmico del parámetro g
                        for j=1:length(c)
                            parametros = sprintf('-s 0 -t 2 -c %i -g
%i -v 5',c(j),g(i));
                            precisionVacasEntrenamiento(i,j)  =
svmtrain (etiquetasVacasTrain',  histogramaTrain, parametros);
                            precisionMotosEntrenamiento(i,j)  =
svmtrain (etiquetasMotosTrain',  histogramaTrain, parametros);
                            precisionCochesEntrenamiento(i,j) =
svmtrain (etiquetasCochesTrain', histogramaTrain, parametros);
                        end
                    end

                    %Se encuentran los parámetros c óptimos para cada
                    %caso
                    [precisionMaximaVacas,indiceC(1)]  = max
(max(precisionVacasEntrenamiento));
                    [precisionMaximaMotos,indiceC(2)]  = max
(max(precisionMotosEntrenamiento));
                    [precisionMaximaCoches,indiceC(3)] = max
(max(precisionCochesEntrenamiento));
                    indiceG(1) = find
(precisionVacasEntrenamiento(:,indiceC(1)) ==precisionMaximaVacas, 1);
                    indiceG(2) = find
(precisionMotosEntrenamiento(:,indiceC(2)) ==precisionMaximaMotos, 1);
                    indiceG(3) = find
(precisionCochesEntrenamiento(:,indiceC(3))==precisionMaximaCoches,1);
                    cOptimo = c(indiceC);
                    gOptimo = g(indiceG);
```

```
                    save
(['cyg\cyg',clasificador,codebook,nC,'.mat'],'indiceC','indiceG','cOpt
imo','gOptimo','minimoC','minimoG','maximoC','maximoG');
                    if (mean(indiceC)<5)
                        msgbox('El parámetro C corresponde a un índice
muy bajo. Se va a recalcular.');
                        minimoC = minimoC/2;
                        maximoC = minimoC*2;
                        indiceCBienCalculado = false;
                    elseif (mean(indiceC)>15)
                        msgbox('El parámetro C corresponde a un índice
muy alto. Se va a recalcular.');
                        minimoC = maximoC/2;
                        maximoC = 2*maximoC;
                        indiceCBienCalculado = false;
                    else
                        indiceCBienCalculado = true;
                    end
                    if (mean(indiceG)<5)
                        msgbox('El parámetro G corresponde a un índice
muy bajo. Se va a recalcular.');
                        minimoG = minimoG/2;
                        maximoG = minimoG*2;
                        indiceGBienCalculado = false;
                    elseif (mean(indiceG)>15)
                        msgbox('El parámetro G corresponde a un índice
muy alto. Se va a recalcular.');
                        minimoG = maximoG/2;
                        maximoG = 2*maximoG;
                        indiceGBienCalculado = false;
                    else
                        indiceGBienCalculado = true;
                    end
                    indiceCBienCalculado = true;
                    indiceGBienCalculado= true;
                end
                %Se obtiene el modelo de la red con g y c óptimos
                cadenaVacas  = sprintf('-s 0 -t 2 -g %i -c
%i',gOptimo(1),cOptimo(1));
                cadenaMotos  = sprintf('-s 0 -t 2 -g %i -c
%i',gOptimo(2),cOptimo(2));
                cadenaCoches = sprintf('-s 0 -t 2 -g %i -c
%i',gOptimo(3),cOptimo(3));
                netVacas  = svmtrain (etiquetasVacasTrain',
histogramaTrain, cadenaVacas);
                netMotos  = svmtrain (etiquetasMotosTrain',
histogramaTrain, cadenaMotos);
                netCoches = svmtrain (etiquetasCochesTrain',
histogramaTrain, cadenaCoches);
                save (archivoGuardar,
'netVacas','netCoches','netMotos');
        end
end
```

6.4.9. ValidacionClasificador.m

```
function validacionClasificador (numeroCodewords, clasificador,
codebook);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% VALIDACIONCLASIFICADOR(numeroCodewords,tipo,esConjunto,esMulticlase)
% La función validacionclasificador simula las redes neuronales
% para hallar las clasificaciones mediante los histogramas calculados
% anteriormente.  En función del parámetro clasificador, usaremos
% un perceptrón monocapa (si tipo vale 'SLP'), una maquina de vector
% soporte tipo kernel lineal (si tipo vale 'SVMKL') o una máquina de
% vector soporte tipo rbf (si tipo vale 'SVMRBF'). El parámetro
% codebook determina si queremos utilizar el 'Global', el 'Local', el
% 'Multiclase' o el 'Conjunto'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Se establecen los nombres de los archivos para guardar datos
nC = int2str (numeroCodewords);
archivoGuardar1 =
['clasificadores/clasificaciones',clasificador,codebook,nC,'.mat'];
archivoCargar1 =
['clasificadores/redes',clasificador,codebook,nC,'.mat'];
disp(archivoCargar1)
%Se empieza a hacer el test
switch codebook
    case 'Conjunto'
        archivoCargar2 =
['histogramas/histogramaConjuntoTest',nC,'.mat'];
    otherwise
        archivoCargar2 = ['histogramas/histogramaTest',nC,'.mat'];
end
load (archivoCargar1);
load (archivoCargar2);

load 'numeroImagenesTest.mat';

%Las etiquetas son vectores que contienen unos en las posiciones
%correspondientes a la clase a clasificar y ceros en el resto
unosVacasTest   = ones (1,numeroImagenesVacas);
unosCochesTest  = ones (1,numeroImagenesCoches);
unosMotosTest   = ones (1,numeroImagenesMotos);
cerosVacasTest  = zeros (1,numeroImagenesVacas);
cerosCochesTest = zeros (1,numeroImagenesCoches);
cerosMotosTest  = zeros (1,numeroImagenesMotos);

etiquetasVacasTest  = [unosVacasTest  cerosMotosTest cerosCochesTest];
etiquetasMotosTest  = [cerosVacasTest unosMotosTest  cerosCochesTest];
etiquetasCochesTest = [cerosVacasTest cerosMotosTest unosCochesTest];

switch codebook
    case 'Conjunto'
```

```
        histogramaTest = [histogramaVacasConjunto;
histogramaMotosConjunto;  histogramaCochesConjunto];
    case 'Local'
        histogramaTest = [histogramaVacasLocal;  histogramaMotosLocal;
histogramaCochesLocal];
    otherwise
        histogramaTest = [histogramaVacasGlobal;
histogramaMotosGlobal;  histogramaCochesGlobal];
end

maximoHistogramaTest = max (histogramaTest);
cerosEnMaximo = find (maximoHistogramaTest==0);
maximoHistogramaTest(cerosEnMaximo) = 0.0001;
[F C]=size(histogramaTest);
numeroImagenesTest =
numeroImagenesVacas+numeroImagenesMotos+numeroImagenesCoches;

switch clasificador
    case 'SLP'
        %Se hace la simulación del perceptrón, que dará la
        %clasificación final
        clasificacionVacas  = sim (netVacas , histogramaTest');
        clasificacionMotos  = sim (netMotos , histogramaTest');
        clasificacionCoches = sim (netCoches, histogramaTest');
        save (archivoGuardar1,
'clasificacionVacas','clasificacionCoches','clasificacionMotos');

    otherwise
        switch codebook
            case 'Multiclase'
                etiquetasTest  = [unosVacasTest  2*unosMotosTest
3*unosCochesTest];
                [clasificacion , precision] = svmpredict
(etiquetasTest' ,histogramaTest,net);
                save (archivoGuardar1, 'clasificacion');
            otherwise
                [clasificacionVacas , precisionVacas ] = svmpredict
(etiquetasVacasTest' ,histogramaTest,netVacas);
                if(sum(clasificacionVacas>0)==0)
                    disp('Salidas de vacas todas a ceros')
                    pause;
                end
                [clasificacionMotos , precisionMotos ] = svmpredict
(etiquetasMotosTest' ,histogramaTest,netMotos);
                if(sum(clasificacionMotos>0)==0)
                    disp('Salidas de motos todas a ceros')
                    pause;
                end
                [clasificacionCoches, precisionCoches] = svmpredict
(etiquetasCochesTest',histogramaTest,netCoches);
                if(sum(clasificacionCoches>0)==0)
                    disp('Salidas de coches todas a ceros')
                    pause;
                end
                save (archivoGuardar1,
'clasificacionVacas','clasificacionCoches','clasificacionMotos');
        end
end
```

6.4.10. CalcularMedidasCalidad.m

```
function calcularMedidasCalidad (numeroCodewords,esMulticlase);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CALCULARMEDIDASCALIDAD (NUMEROCODEWORDS, ESMULTICLASE)
% La función calcularMedidasCalidad calcula la precisión y recall, la
% medida F y las probabilidades de falsa alarma y detección. El
% parámetro esMulticlase determina si la red neuronal utiliza una
% clasificación multiclase.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

nC = int2str (numeroCodewords);
load ('numeroImagenesTest.mat');

numeroImagenesTest =
numeroImagenesVacas+numeroImagenesMotos+numeroImagenesCoches;
probabilidadHv = numeroImagenesVacas/numeroImagenesTest;
probabilidadHm = numeroImagenesMotos/numeroImagenesTest;
probabilidadHc = numeroImagenesCoches/numeroImagenesTest;
relevantes =
[numeroImagenesVacas;numeroImagenesMotos;numeroImagenesCoches];
nombrePredictor = ['SLP   '; 'SVMKL ';'SVMRBF'];

if (esMulticlase == false)
    for cod=1:3   %cod cambia el codebook empleado
        switch cod
            case 1  %Codebooks globales
                codebook ='Global';
            case 2  %Codebooks globales modificados
                codebook = 'Conjunto';
            case 3 %Codebooks locales
                codebook = 'Local';
        end
        archivoCargarSLP    = ['clasificadores/clasificacionesSLP',
codebook,nC,'.mat'];
        archivoCargarSVMKL  = ['clasificadores/clasificacionesSVMKL',
codebook,nC,'.mat'];
        archivoCargarSVMRBF =
['clasificadores/clasificacionesSVMRBF',codebook,nC,'.mat'];
        archivoGuardarSLP    = ['probabilidades/fSLP'
,codebook,nC,'.mat'];
        archivoGuardarSVMKL  = ['probabilidades/fSVMKL'
,codebook,nC,'.mat'];
        archivoGuardarSVMRBF =
['probabilidades/fSVMRBF',codebook,nC,'.mat'];
        for clas=1:3 %clas es el clasificador (SLP,SVMKL,SVMRBF)
            if clas==1
                load (archivoCargarSLP);
            elseif (clas==2)
                load (archivoCargarSVMKL);
            elseif (clas==3)
                load (archivoCargarSVMRBF);
            end
```

```
            if clas==1 %SLP monoclase
                probabilidadDvHv = numel(find (clasificacionVacas
(1:numeroImagenesVacas) >= 0));
                probabilidadDmHv = numel(find (clasificacionMotos
(1:numeroImagenesVacas) >= 0));
                probabilidadDcHv = numel(find (clasificacionCoches
(1:numeroImagenesVacas) >= 0));

                probabilidadDvHm = numel(find (clasificacionVacas
((numeroImagenesVacas+1):(numeroImagenesVacas+numeroImagenesMotos)) >=
0));
                probabilidadDmHm = numel(find (clasificacionMotos
((numeroImagenesVacas+1):(numeroImagenesVacas+numeroImagenesMotos)) >=
0));
                probabilidadDcHm = numel(find (clasificacionCoches
((numeroImagenesVacas+1):(numeroImagenesVacas+numeroImagenesMotos)) >=
0));

                probabilidadDvHc = numel(find (clasificacionVacas
((numeroImagenesVacas+numeroImagenesMotos+1):end) >= 0));
                probabilidadDmHc = numel(find (clasificacionMotos
((numeroImagenesVacas+numeroImagenesMotos+1):end) >= 0));
                probabilidadDcHc = numel(find (clasificacionCoches
((numeroImagenesVacas+numeroImagenesMotos+1):end) >= 0));

            elseif (clas==2||clas==3) %SVMs monoclase
                probabilidadDvHv = numel(find (clasificacionVacas
(1:numeroImagenesVacas) == 1));
                probabilidadDmHv = numel(find (clasificacionMotos
(1:numeroImagenesVacas) == 1));
                probabilidadDcHv = numel(find (clasificacionCoches
(1:numeroImagenesVacas) == 1));

                probabilidadDvHm = numel(find (clasificacionVacas
((numeroImagenesVacas+1):(numeroImagenesVacas+numeroImagenesMotos)) ==
1));
                probabilidadDmHm = numel(find (clasificacionMotos
((numeroImagenesVacas+1):(numeroImagenesVacas+numeroImagenesMotos)) ==
1));
                probabilidadDcHm = numel(find (clasificacionCoches
((numeroImagenesVacas+1):(numeroImagenesVacas+numeroImagenesMotos)) ==
1));

                probabilidadDvHc = numel(find (clasificacionVacas
((numeroImagenesVacas+numeroImagenesMotos+1):end) == 1));
                probabilidadDmHc = numel(find (clasificacionMotos
((numeroImagenesVacas+numeroImagenesMotos+1):end) == 1));
                probabilidadDcHc = numel(find (clasificacionCoches
((numeroImagenesVacas+numeroImagenesMotos+1):end) == 1));
            end
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % PROBABILIDADES DE FALSA ALARMA Y DETECCIÓN

            probabilidadFAVacas (clas,cod) =
(probabilidadDvHm*probabilidadHm +
probabilidadDvHc*probabilidadHc)/numeroImagenesTest;
```

```
            probabilidadFAMotos (clas,cod) =
(probabilidadDmHv*probabilidadHv +
probabilidadDmHc*probabilidadHc)/numeroImagenesTest;
            probabilidadFACoches(clas,cod) =
(probabilidadDcHv*probabilidadHv +
probabilidadDcHm*probabilidadHm)/numeroImagenesTest;
            probabilidadPerdidaVacas (clas,cod) = 1-
(probabilidadDvHv/numeroImagenesVacas);
            probabilidadPerdidaMotos (clas,cod) = 1-
(probabilidadDmHm/numeroImagenesMotos);
            probabilidadPerdidaCoches(clas,cod) = 1-
(probabilidadDcHc/numeroImagenesCoches);
            if (probabilidadFAVacas (clas,cod)<0 ||
probabilidadFAVacas (clas,cod)>1 ||probabilidadFACoches (clas,cod)<0||
probabilidadFACoches (clas,cod)>1||probabilidadFAMotos (clas,cod)<0 ||
probabilidadFAMotos (clas,cod)>1)
                error ('Ninguna de las probabilidades de falsa alarma
o pérdida pueden valer más de 1 o menos de 0');
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % PRECISIÓN-RECALL Y MEDIDA F

            relevantesRecuperados   =
[probabilidadDvHv;probabilidadDmHm;probabilidadDcHc];
            recuperados    =
[(probabilidadDvHv+probabilidadDvHm+probabilidadDvHc);(probabilidadDmH
v+probabilidadDmHm+probabilidadDmHc);(probabilidadDcHv+probabilidadDcH
m+probabilidadDcHc)];

            %Cálculo de las precisiones y del recall
            for clase=1:3 %Vacas, motos, coches
                if (recuperados(clase) == 0)
                    error (['No se ha recuperado ninguna imagen de la
clase ',int2str(clase),', clasificador ',int2str(clas)]);
                end
                precision(clas,cod,clase)  = relevantesRecuperados
(clase) /recuperados(clase);
                if (precision(clas,cod,clase)<0 ||
precision(clas,cod,clase)>1)
                    error ('Ninguna de las precisiones pueden valer
más de 1 o menos de 0');
                end
                if (recuperados (clase)==0 && relevantesRecuperados
(clase)==0)
                    precision(clas,cod,clase)  == 0;
                end
                recall (clas,cod,clase) = relevantesRecuperados
(clase)  /relevantes (clase);
                if (recall(clas,cod,clase)<0 ||
recall(clas,cod,clase)>1)
                    error ('Ninguno de los recall pueden valer más de
1 o menos de 0');
                end
                f(clas,cod,clase)   = 2*(precision(clas,cod,clase)
*recall(clas,cod,clase)  )/(precision(clas,cod,clase)
+recall(clas,cod,clase));
                if (isnan (f(clas,cod,clase)))
                    f(clas,cod,clase) = 0;
```

```
                end
            end %for clase
            fVacas    = f (clas,cod,1);
            fMotos    = f (clas,cod,2);
            fCoches   = f (clas,cod,3);
            if (clas==1)
                save (archivoGuardarSLP,
'fVacas','fCoches','fMotos');
            elseif clas==2
                save (archivoGuardarSVMKL,
'fVacas','fCoches','fMotos');
            elseif clas==3
                save
(archivoGuardarSVMRBF,'fVacas','fCoches','fMotos');
            end
        end %for clas
        for clase=1:3
            if f(2,cod,clase)>f(3,cod,clase)
                disp (['Para el codebook ',int2str(cod),' los
resultados del SVMKL son mejores que los del SVMRBF']);
            end
        end
    end %for cod
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %PRESENTACIÓN DE RESULTADOS
    % Representación de la probabilidad de falsa alarma y la de
    % pérdida
    disp ('*****************PROBABILIDADES DE FALSA ALARMA Y
PÉRDIDA******************');
    disp ('CLASIFICADOR         PROB. F.A.
PROB. PÉR.');
    disp ('             Vacas       Motos      Coches       Vacas
Motos      Coches');
    for cod=1:3
        if (cod==1)
            disp ('CODEBOOK GLOBAL');
        elseif (cod==2)
            disp ('CODEBOOK GLOBAL MODIFICADO');
        elseif (cod==3)
            disp ('CODEBOOK LOCAL');
        end
        for clas=1:3
            disp ([nombrePredictor(clas,:),'
',num2str(probabilidadFAVacas(clas,cod)),'
',num2str(probabilidadFAMotos(clas,cod)),'
',num2str(probabilidadFACoches(clas,cod)),     '
',num2str(probabilidadPerdidaVacas(clas,cod)), '
',num2str(probabilidadPerdidaMotos(clas,cod)), '
',num2str(probabilidadPerdidaCoches(clas,cod))]);
        end
    end
    if (numel(find(f==0))>0)
        error('Alguna medida F es 0');
    end
    % Representación de precisión y recall
    disp ('*****************RESULTADOS DE PRECISION Y RECALL
******************');
    disp ('CLASIFICADOR         PRECISION
RECALL');
```

```
    disp ('             Vacas       Motos      Coches       Vacas
Motos      Coches');
    for cod=1:3
        if (cod==1)
            disp ('CODEBOOK GLOBAL');
        elseif (cod==2)
            disp ('CODEBOOK GLOBAL MODIFICADO');
        elseif (cod==3)
            disp ('CODEBOOK LOCAL');
        end
        for clas=1:3
            disp ([nombrePredictor(clas,:),'
',num2str(precision(clas,cod,1)),'
',num2str(precision(clas,cod,2)),'
',num2str(precision(clas,cod,3)),     '
',num2str(recall(clas,cod,1)), '     ',num2str(recall(clas,cod,2)), '
',num2str(recall(clas,cod,3))]);
        end
        %Representación de la medida F para el clasificador monoclase
        figure;
        bar ([f(1,cod,1) f(1,cod,2) f(1,cod,3);f(2,cod,1) f(2,cod,2)
f(2,cod,3);f(3,cod,1) f(3,cod,2) f(3,cod,3)]);
        if cod==1
            title (['Medida F para ',nC,' codewords en la
clasificación monoclase global']);
        elseif cod==2
            title (['Medida F para ',nC,' codewords en la
clasificación monoclase global modificado']);
        elseif cod==3
            title (['Medida F para ',nC,' codewords en la
clasificación monoclase local']);
        end
        axis ([0 4 0 1]);
        legend ('Vacas', 'Coches','Motos','Location', 'SouthEast');
        xlabel (['SLP (1), SVMKL(2) y SVMRBF(3)']);
        grid on;
    end

elseif (esMulticlase)
    archivoCargarSVMKL =
['clasificadores/clasificacionesSVMKLMulticlase',nC,'.mat'];
    archivoGuardarSVMKL=
['probabilidades/fSVMKLMulticlase',nC,'.mat'];
    archivoCargarSVMRBF =
['clasificadores/clasificacionesSVMRBFMulticlase',nC,'.mat'];
    archivoGuardarSVMRBF=
['probabilidades/fSVMRBFMulticlase',nC,'.mat'];
    for clas=2:3
        if (clas==2)
            load (archivoCargarSVMKL);
        elseif (clas==3)
            load (archivoCargarSVMRBF);
        end
        probabilidadDvHv = numel(find (clasificacion
(1:numeroImagenesVacas) == 1));
        probabilidadDmHv = numel(find (clasificacion
(1:numeroImagenesVacas) == 2));
        probabilidadDcHv = numel(find (clasificacion
(1:numeroImagenesVacas) == 3));
```

```
        probabilidadDvHm = numel(find (clasificacion
((numeroImagenesVacas+1):(numeroImagenesVacas+numeroImagenesMotos)) ==
1));
        probabilidadDmHm = numel(find (clasificacion
((numeroImagenesVacas+1):(numeroImagenesVacas+numeroImagenesMotos)) ==
2));
        probabilidadDcHm = numel(find (clasificacion
((numeroImagenesVacas+1):(numeroImagenesVacas+numeroImagenesMotos)) ==
3));

        probabilidadDvHc = numel(find (clasificacion
((numeroImagenesVacas+numeroImagenesMotos+1):end) == 1));
        probabilidadDmHc = numel(find (clasificacion
((numeroImagenesVacas+numeroImagenesMotos+1):end) == 2));
        probabilidadDcHc = numel(find (clasificacion
((numeroImagenesVacas+numeroImagenesMotos+1):end) == 3));

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % MATRIZ DE CONFUSIÓN

        disp('**********************MATRIZ DE
CONFUSIÓN**********************');
        disp(['HIPÓTESIS  \DECISIONES
',nombrePredictor(clas,:)]);
        disp('            Vacas      Motos       Coches');
        disp(['Vacas         ',int2str(probabilidadDvHv),'
',int2str(probabilidadDmHv),'
',int2str(probabilidadDcHv)]);
        disp(['Motos         ',int2str(probabilidadDvHm),'
',int2str(probabilidadDmHm),'
',int2str(probabilidadDcHm)]);
        disp(['Coches        ',int2str(probabilidadDvHc),'
',int2str(probabilidadDmHc),'
',int2str(probabilidadDcHc)]);

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % PROBABILIDADES DE FALSA ALARMA Y DETECCIÓN

        probabilidadFAVacas (clas)= (probabilidadDvHm*probabilidadHm +
probabilidadDvHc*probabilidadHc)/numeroImagenesTest;
        probabilidadPerdidaVacas (clas)= 1-
(probabilidadDvHv/numeroImagenesVacas);

        probabilidadFAMotos(clas) = (probabilidadDmHv*probabilidadHv +
probabilidadDmHc*probabilidadHc)/numeroImagenesTest;
        probabilidadPerdidaMotos(clas) = 1-
(probabilidadDmHm/numeroImagenesMotos);

        probabilidadFACoches(clas) = (probabilidadDcHv*probabilidadHv
+ probabilidadDcHm*probabilidadHm)/numeroImagenesTest;
        probabilidadPerdidaCoches(clas) = 1-
(probabilidadDcHc/numeroImagenesCoches);

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % PRECISIÓN-RECALL Y MEDIDA F
```

```
        relevantesRecuperados   =
[probabilidadDvHv;probabilidadDmHm;probabilidadDcHc];
        recuperados    =
[(probabilidadDvHv+probabilidadDvHm+probabilidadDvHc);(probabilidadDmH
v+probabilidadDmHm+probabilidadDmHc);(probabilidadDcHv+probabilidadDcH
m+probabilidadDcHc)];

        %Cálculo de las precisiones y del recall
        for clase=1:3 %Vacas, motos, coches
            precision(clas,clase)  = relevantesRecuperados (clase)
/recuperados(clase);
            if (precision(clas,clase)<0 || precision(clas,clase)>1)
                error ('Ninguna de las precisiones pueden valer más de
1 o menos de 0');
            end
            if (recuperados (clase)==0 && relevantesRecuperados
(clase)==0)
                precision(clas,clase)  == 0;
            end
            recall (clas,clase) = relevantesRecuperados (clase)
/relevantes (clase);
            if (recall(clas,clase)<0 || recall(clas,clase)>1)
                error ('Ninguno de los recall pueden valer más de 1 o
menos de 0');
            end
            f(clas,clase)   = 2*(precision(clas,clase)
*recall(clas,clase)  )/(precision(clas,clase)   +recall(clas,clase));
            if (isnan (f(clas,clase)))
                f(clas,clase) = 0;
            end
        end
        fVacas   = f (clas,1);
        fCoches  = f (clas,2);
        fMotos   = f (clas,3);
        if clas==2
            save (archivoGuardarSVMKL, 'fVacas','fCoches','fMotos');
        elseif clas==3
            save (archivoGuardarSVMRBF,'fVacas','fCoches','fMotos');
        end
    end

    % Representación de la probabilidad de falsa alarma y la de
    % pérdida
    disp ('****************PROBABILIDADES DE FALSA ALARMA Y
PÉRDIDA*****************');
    disp ('CLASIFICADOR         PROB. F.A.
PROB. PÉR.');
    disp ('MULTICLASE      Vacas       Coches      Motos       Vacas
Coches      Motos');
    for clas=2:3
        disp ([nombrePredictor(clas,:),'
',num2str(probabilidadFAVacas(clas)),'
',num2str(probabilidadFAMotos(clas)),'
',num2str(probabilidadFACoches(clas)),     '
',num2str(probabilidadPerdidaVacas(clas)), '
',num2str(probabilidadPerdidaMotos(clas)), '
',num2str(probabilidadPerdidaCoches(clas))]);
    end
    % Representación de la precisión y el recall
```

```
    disp ('****************RESULTADOS DE PRECISION Y
RECALL******************');
    disp ('CLASIFICADOR         PRECISION
RECALL');
    disp ('MULTICLASE      Vacas       Motos      Coches       Vacas
Motos      Coches');
    for clas=2:3
        disp ([nombrePredictor(clas,:),'
',num2str(precision(clas,1)),'      ',num2str(precision(clas,2)),'
',num2str(precision(clas,3)),     '     ',num2str(recall(clas,1)), '
',num2str(recall(clas,2)), '     ',num2str(recall(clas,3))]);
    end

    %Representación de la medida F para el clasificador multiclase
    figure;
    bar ([f(2,1) f(2,2) f(2,3);f(3,1) f(3,2) f(3,3)]);
    title (['Medida F para ',nC,' codewords en la clasificación
multiclase ']);
    axis ([0 4 0 1]);
    legend ('Vacas', 'Coches','Motos','Location', 'SouthEast');
    xlabel ([nombrePredictor(clas,:),'(1) y
',nombrePredictor(clas,:),'(2)']);
    grid on;
end
```

6.4.11. PROYECTO.m

```
function PROYECTO (numeroCodewords, numeroIteraciones);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PROYECTO (NUMEROCODEWORDS, NUMEROITERACIONES)
% La función PROYECTO llama a todas las funciones con los parámetros
% adecuados para realizar toda la ejecución del proyecto.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

calcularCodebooks(numeroCodewords, numeroIteraciones);
calcularIteracionOptima(numeroCodewords, numeroIteraciones);
asignacionCodewords('Train',numeroCodewords);
histograma('Train',numeroCodewords);
asignacionCodewords ('Test',numeroCodewords);
histograma ('Test',numeroCodewords);
calcularCodebookConjunto (numeroCodewords, numeroIteraciones);

entrenamientoClasificador (numeroCodewords,'SLP','Global');
entrenamientoClasificador (numeroCodewords,'SLP','Local');
entrenamientoClasificador (numeroCodewords,'SLP','Conjunto');
entrenamientoClasificador (numeroCodewords,'SVMKL','Global');
entrenamientoClasificador (numeroCodewords,'SVMKL','Local');
entrenamientoClasificador (numeroCodewords,'SVMKL','Multiclase');
entrenamientoClasificador (numeroCodewords,'SVMKL','Conjunto');
entrenamientoClasificador (numeroCodewords,'SVMRBF','Global');
entrenamientoClasificador (numeroCodewords,'SVMRBF','Local');
entrenamientoClasificador (numeroCodewords,'SVMRBF','Multiclase');
entrenamientoClasificador (numeroCodewords,'SVMRBF','Conjunto');

validacionClasificador (numeroCodewords,'SLP','Global');
validacionClasificador (numeroCodewords,'SLP','Local');
validacionClasificador (numeroCodewords,'SLP','Conjunto');
validacionClasificador (numeroCodewords,'SVMKL','Global');
validacionClasificador (numeroCodewords,'SVMKL','Local');
validacionClasificador (numeroCodewords,'SVMKL','Multiclase');
validacionClasificador (numeroCodewords,'SVMKL','Conjunto');
validacionClasificador (numeroCodewords,'SVMRBF','Global');
validacionClasificador (numeroCodewords,'SVMRBF','Local');
validacionClasificador (numeroCodewords,'SVMRBF','Multiclase');
validacionClasificador (numeroCodewords,'SVMRBF','Conjunto');

calcularMedidasCalidad (numeroCodewords, false);
calcularMedidasCalidad (numeroCodewords, true);
```

6.4.12. ComparativaClasificadores.m

```
function comparativaClasificadores

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% La función COMPARATIVACLASIFICADORES compara los 3 tipos de
% clasificadores empleados para las 3 clases y para la utilización de
% características globales. Se comparan mediante la medida F
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%No se utilizan todos los datos porque quedarían algunos picos en la
%gráfica. Estos picos son producto de las inicializaciones aleatorias.
x=[10:20:250 300:50:1550 1750 2000];

for i=1:41
    cadena=sprintf ('probabilidades/fSLP%i.mat',x(i));
    load (cadena);
    fVacasGlobalSLP (i) = fVacasGlobal;
    fMotosGlobalSLP (i) = fMotosGlobal;
    fCochesGlobalSLP(i) = fCochesGlobal;

    cadena=sprintf ('probabilidades/fSVMKL%i.mat',x(i));
    load (cadena);
    fVacasGlobalSVMKL (i) = fVacasGlobal;
    fMotosGlobalSVMKL (i) = fMotosGlobal;
    fCochesGlobalSVMKL(i) = fCochesGlobal;

    cadena=sprintf ('probabilidades/fSVMRBF%i.mat',x(i));
    load (cadena);
    fVacasGlobalSVMRBF (i) = fVacasGlobal;
    fMotosGlobalSVMRBF (i) = fMotosGlobal;
    fCochesGlobalSVMRBF(i) = fCochesGlobal;

end

xi = 10:200:2000;

%Se usa la interpolación para suavizar las curvas
fVacasGlobalSLP  = interp1(x,fVacasGlobalSLP ,xi);
fMotosGlobalSLP  = interp1(x,fMotosGlobalSLP ,xi);
fCochesGlobalSLP = interp1(x,fCochesGlobalSLP,xi);

fVacasGlobalSVMKL  = interp1(x,fVacasGlobalSVMKL ,xi);
fMotosGlobalSVMKL  = interp1(x,fMotosGlobalSVMKL ,xi);
fCochesGlobalSVMKL = interp1(x,fCochesGlobalSVMKL,xi);

fVacasGlobalSVMRBF  = interp1(x,fVacasGlobalSVMRBF ,xi);
fMotosGlobalSVMRBF  = interp1(x,fMotosGlobalSVMRBF ,xi);
fCochesGlobalSVMRBF = interp1(x,fCochesGlobalSVMRBF,xi);

%Representación de la comparación
subplot (1,3,1), plot (xi,fVacasGlobalSLP,'g',
xi,fVacasGlobalSVMRBF,'m',xi,fVacasGlobalSVMKL,'b');
axis ([10 2000 0 1]);
legend ('SLP', 'SVM RBF', 'SVM kernel lineal','Location','South');
```

```
title 'Clasificadores de vacas con características globales';

subplot (1,3,2), plot
(xi,fMotosGlobalSLP,'g',xi,fMotosGlobalSVMRBF,'m',xi,fMotosGlobalSVMKL
,'b');
axis ([10 2000 0 1]);
legend ('SLP', 'SVM RBF', 'SVM kernel lineal','Location','South');
title 'Clasificadores de motos con características globales';

subplot (1,3,3), plot
(xi,fCochesGlobalSLP,'g',xi,fCochesGlobalSVMRBF,'m',xi,fCochesGlobalSV
MKL,'b');
axis ([10 2000 0 1]);
legend ('SLP', 'SVM RBF', 'SVM kernel lineal','Location','South');
title 'Clasificadores de coches con características globales';
```

6.5. Referencias

[1] Matlab Help. "The Mathworks", 2005

[2] David G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints International Journal of Computer Vision, 60, 2 (2004), pp. 91-110.

[3] Shigeo Abe, "Pattern classification. Neuro-fuzzy methods and their comparison", 2001

[4] R. Rojas, "Neural networks", 1996

[5] Wikipedia, http://en.wikipedia.org/wiki/Machine_learning

[6] Edgar Nelson Sánchez Camperos, Alma Yolanda Alanís García, "Redes neuronales: conceptos fundamentales y aplicaciones a control automático", 2006

[7] Tzay Y. Young, King-Sun Fu, "Handbook of pattern recognition and image processing", 1986

[8] Tratamiento digital de la información, transparencias de clase.

[9] Nuno Vasconcelos, "From pixels to semantic spaces: advances in content-based image retrieval", IEEE Computer, vol. 40 , no. 7, pp. 20-26, July 2007

[10] Michael S. Lew, Nicu Sebe, Chabane Djeraba, Ramesh Jain, "Content-based Multimedia Information Retrieval: State of the Art and Challenges", ACM, Transactions on Multimedia Computing, Communicatios, and Applications (TOMCCAP), v.2 n.1, p. 1-19, February 2006

[11] Rittrenda Datta, Dhiraj Joshi, Jia Li y James Z. Wang, "Image retrieval: ideas, influences and trends of the new age", ACM, Computing Surveys (CSUR), vol.40, Issue 2, April 2008

[12] Dana H. Ballard, Christopher M. Brown, “Computer vision”, Prentice hall, 1982

[13] Lai Wang, Latifur Khan, Li Liu y Weili Wu, “Automatic image annotation and retrieval using weighted feature selection”, vol. 29, no. 1, p. 55-71, Springer, April 2004

[14] Gustavo Carneiro, Antoni B. Chan, Pedro J. Moreno y Nuno Vasconcelos, “Supervised learning of semantic classes for image annotation and retrieval”, IEEE Transactions on pattern analysis and machine intelligence, vol. 29, n. 3, p. 394-410, March 2007

[15] A. Grace Selvarani y Dr. S. Annadurai “Medical image retrieval by combining low level features and Dicom features” IEEE International conference on computational intelligence and multimedia applications (ICCIMA), vol. 1, pp.587-589, 2007.

[16] Hu Xuelong, Zhang Yuhui y Yang Li, “A new method for semi-automatic image annotation” The eighth international conference on electronic measurement and instruments, CEMI, p. 866-869, July 2007

[17] Luis Llano, Andrés Hoyos, Francisco Arias, Juan Velásquez, “Comparación del desempeño de funciones de activación en redes Feedforward para aproximar funciones de datos con y sin ruido”, Revista Avances en sistemas e informática, Septiembre de 2007

[18] Changhu Wang, Feng Jing, Lei Zhang, Hong-Jiang Zhang, “Content-based image annotation refinement”, International Conference on Computer Vision and Pattern recognition (CVPR), June 2007

[19] A. Vedaldi, B. Fulkerson “An introduction to the VisionLab Features Library”

[20] Chih-Chung Chang and Chih-Jen Lin "LIBSVM -- A Library for Support Vector Machines", http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[21] T. Hofmann, "Unsupervised learning by Probabilistic Latent Semantic Analysis," Machine Learning, vol. 42, no. 1/2, pp. 177–196, 2001.

[22] D. M. Blei, A. Y. Ng, M. I. Jordan, and J. Lafferty, "Latent Dirichlet Allocation," Journal of Machine Learning Research, vol. 3, p. 2003, 2003.

[23] M. Szummer y R. Picard, "Indoor-outdoor image classification", Proc. IEEE Int'l Workshop Conent-Based Access of Image and Video Databases (CAIVD '98) in Int'l Conf. Computer Vision, January, 1998.

[24] A. Vailaya, A.Jain y H. Zhang, "On image classification: city vs. landscape", IEEE Workshop on Content – Based Access of Image and Video Libraries, p. 3-8, 1998.

[25] N. Haering, Z. Myles y N. Lobo "Locating deciduous trees", IEEE Workshop on Content – Based Access of Image and Video Libraries, p. 18-25, June 1997.

[26] D. Forsyth y M. Fleck, "Body plans", CVPR, p. 678-683, 1997.

[27] Y. Li y L. Shapiro, "Consistent line clusters for building recognition in CBIR", ICPR02, vol. 3, p.952-956.

[28] Y. Mori, H. Takahashi, and R. Oka, "Image-to-word transformation based on dividing and vector quantizing images with words", in MISRM'99 First International Workshop on Multimedia Intelligent Storage and Retrieval Management, 1999.

[29] J. Jeon, V. Lavrenko, R. Manmatha, "Automatic Image Annotation and Retrieval using Cross-Media Relevance Models," 26th Annual Int. ACM SIGIR Conference, Toronto, Canada, 2003.

[30] P. Duygulu, K. Barnard, N. de Freitas, D. Forsyth, "Object recognition as machine translation: learning a lexicon for a fixed image vocabulary," in Proceedings of Seventh European Conference on Computer Vision (ECCV), Vol. 4, pp. 97-112, 2002.

[31] K. Barnard, P. Duygulu, N. de Freitas, D. Forsyth, D. Blei, M. Jordan, "Matching words and pictures," Journal of Machine Learning Research, Vol.3, pp.1107-1135, 2003.

[32] D. Blei, M. Jordan, "Modeling annotated data," 26th Annual Int. ACM SIGIR Conf., Toronto, Canada, 2003.

[33] J. Li and J. Z. Wang, "Automatic linguistic indexing of pictures by a statistical modeling approach," IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 25, pp.1075-1088, 2003.

[34] K. Barnard y D. Forsyth, "Learning the semantics of words and pictures", Computer Vision, ICCV 2001. Proceedings, Eighth IEEE International Conference on, vol. 2, pp. 408-415.

[35] D. Blei y M. Jordan "Modeling annotated data" 26th International Conference on Research and Development in Information Retrieval (SIGIR), 2003.

[36] P. Carbonetto, N. de Freitas y K. Barnard "A statistical model for general contextual object recognition", Proceedings of ECCV, 2004.

[37] P. Duygulu, K. Barnard y D. F. N. Freitas "Object recognition as machine translation: learning a lexicon for a fixed image vocabulary", Proceedings of the 7th European Conference on Computer Vision-Part IV, p.97-112, May, 2002 .

[38] S. Feng, R. Manmatha y V. Lavrenko "Multiple Bernoulli relevance models for image and video annotation", Computer Vision and Pattern Recognition, 2004.

CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on, vol. 2, pp. 1002-1009, July 2004.

[39] P. Carbonetto, H. Kueck y N. Freitas “A constrained semi-supervised learning approach to data association”, 8th European Conference on Computer Vision ECCV04, vol. 3, pp.1-12, May 2004.

[40] V. Lavrenko, R. Manmatha y J. Jeon “A model for learning the semantics of pictures”, Conference On Image And Video Retrieval, Proceedings of the 2008 international conference on Content-based image and video retrieval, pp. 427-436, 2008.

[41] J. Pan, H. Yang, C. FAloutsos and P. Duygulu “GCap: Graph-based automatic image captioning”, cvprw, vol. 9, pp.146, 2004 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'04) Volume 9, 2004.

[42] C. Harris and M. Stephens (1988). "A combined corner and edge detector". *Proc. of the 4th Alvey Vision Conference*: 147-151.

[43] D. Lowe (1999). "Object recognition with informative features and linear classification". Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on (2003), pp. 281-288 vol.1.

[44] T. Kadir and M. Brady (2001). "Scale, saliency and image description". International Journal of Computer Vision 45 (2): 83–105. doi:10.1023/A:1012460413855.

[45] Doina Ana Cernea. “SOAF: un sistema de indexado semántico de OA basado en anotaciones colaborativas”, 2007.

[46] Smeulder, A. W., Worring, M., Santini, S., Gupta, A., and Jain, R. 2000. “Content-based image retrieval at the end of the early years”. IEEE Trans. Pattern Anal. Mach. Intell. 22, 12, 1349-1380.