Trabajo Fin de Grado

Grado en Ingeniería de las Tecnologías de Telecomunicación

Aplicación del Análisis en Componentes Independientes (ICA) a la clasificación de imágenes

Autor: José Camacho Reyes

Tutora: Susana Hornillo Mellado

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Aplicación del Análisis en Componentes Independientes (ICA) a la clasificación de imágenes

Autor:
José Camacho Reyes

Tutor:
Susana Hornillo Mellado
Profesora Contratada Doctora

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Proyecto Fin de Carrera: Aplicación del Análisis en Componentes Independientes (ICA) a la clasificación de imágenes

Autor: José Camacho Reyes

Tutor: Susana Hornillo Mellado

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

A mis padres por confiar en mí, haberme permitido siempre tomar mis propias deciciones, aportarme lo mejor que puedo dar como persona y perdonar mis equivocaciones.

A mis hermanos por tener una actitud positiva frente a los acontecimientos, así como resiliencia y sentido de superación.

A mi abuela Carmen que, aunque nos dejara hace unos años, nunca olvidaré y siempre tendré presente.

A mi primo Luciano por ser un ejemplo que seguir y modelo en el que mirarme desde la infancia.

A mi tío Manolo por darme su visión de la vida y compartir conmigo innumerables experiencias.

A mis amigos, en particular, quiero tener unas palabras de agradecimiento hacia Antonio, Lito y Sergio por saber escuchar y aportarme cada uno su grano de arena con su muy valorada amistad.

Agradecer a mi tutora, Susana, su aporte a la mejora del trabajo y comprensión en mis vaivenes con el mismo.

A todas las demás personas que han contribuido en mi vida de una forma u otra y me han acompañado en este camino sigan estando o se hayan ido, gracias, por sumar en mi desarrollo por este camino llamado vida.

Por último, pedir dicsculpas por dejar a personas importantes sin nombrar explícitamente. Solo deseo añadir que el verdadero agradecimiento se muestra en las acciones, en los detalles, hacia aquellos seres que consiguen nuestra estima más profunda.

José Camacho Reyes

Sevilla, 2019

Resumen

El presente trabajo tiene como objeto el desarrollo de un clasificador de texturas y para ello se usa la base de datos de Columbia-Utrecht (CUReT). El desarrollo del clasificador se apoya en el paper publicado por Manik Varma y Andrew Zisserman: "A Statistical Approach to Texture Classification from Single Images".

Como aporte original se emplea el uso de una técnica matemática, llamada, análisis en componentes independientes (ICA) para conseguir el banco de filtros que permitirán obtener las características de las texturas. Esto permite poder comparar con los filtros que, los mencionados autores, utilizan en su trabajo. El presente documento aporta un nuevo enfoque puesto que los filtros obtenidos a partir de ICA son filtros dependientes de las propias texturas.

Abstract

The purpose of this work is the development of a textures classifier using the Columbia-Utrecht database (CUReT). The development of the classifier is based on the paper published by Manik Varma and Andrew Zisserman: "A Statistical Approach to Texture Classification from Single Images".

As original contribution, the use of a mathematical technique, analysis of independent components (ICA), is used to get the bank of filters that will allow us to get the characteristics of the textures. This allows us to compare with the filters that the mentioned authors use in their work providing a new approach because the filters obtained from ICA are filters dependent on the textures themselves.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

ICA	Independent Component Analysis
kurt	Kurtosis
A	Matriz de bases/matriz de mezclas
W	Matriz inversa de A
X	Matriz de observaciones
y	Variable aleatoria
BSS	Blind Source Separation
J	Negentropía/Neguentropía
w	Matriz de permutación
H	Entropía de una variable aleatoria
V	Matriz ortogonal de autovectores
D	Matriz ortogonal de autovalores
P	Matriz de permutación
I	Información mutua
CUReT	Base de datos Columbia-Utrecht
cos	Función coseno
cosh	Función coseno hiperbólico
tanh	Función tangente hiperbólica
χ^2	Distancia chi-cuadrado
kNN	k-Nearest-Neighbors
PCA	Principal component analysis
E [·]	Operador esperanza matemática
:	Tal que
<	Menor o igual
>	Mayor o igual
$\propto$	Signo de proporcionalidad

1 INTRODUCCIÓN

"Nobody ever figures out what life is all about, and it doesn't matter. Explore the world. Nearly everything is really interesting if you go into it deeply enough. "

- Richard Feynman-

En el momento de escribir el presente documento, se están viviendo una serie de cambios sociales propiciados por diferentes avances tecnológicos (lo que muchos pensadores denominan una nueva revolución industrial). Sin entrar en detalles, se puede ver como empresas tecnológicas, nacidas en la última década, han revolucionado el paradigma social con productos que se nutren, en última instancia, por algoritmos cuyo propósito es clasificar patrones, imágenes o, en definitiva, características. Todo esto tiene sus cimientos edificados sobre los famosos conceptos de *big data*, *machine learning* o *deep learning (*entre otros).

Se entiende, por tanto, que en este contexto la clasificación de características tome cierta relevancia y sea motivo de estudio por parte de organismos tanto públicos como privados. No es casualidad que empresas que necesitan y/o usan estos algoritmos para sus diversos productos (y el posicionamiento de sus servicios) sean, actualmente, las empresas con mayor valor de mercado. Entre ellas podemos citar a: Alphabet (que engloba a Google), Apple, Amazon o Facebook.

Dentro de esta contaste "guerra", por la clasificación de patrones para la obtención de resultados y conclusiones, podemos hablar de las texturas y su procesamiento de cara a, valga la redundancia, la clasificación; concepto que descansa sobre diversas técnicas matemáticas. Existen muchos campos donde tiene sentido el tratamiento de las texturas y la búsqueda de características en las mismas (en el siguiente punto se dará una breve pincelada de los mismos).

En definitiva, el estudio y manejo de las texturas y, por tanto, la extracción de sus características es un tema muy de actualidad, y con mucha importancia en diversos sectores económicos que además terminan repercutiendo en aplicaciones con gran impacto social.

La contribución del presente documento explorará el uso de la técnica matemática: *Independent Component Analysis,* la cual, viene siendo usada desde hace ya algún tiempo. En este documento se le dará relación con la clasificación de texturas para su posterior reconocimiento. Veremos en los siguientes puntos que esta técnica tiene, todavía, mucho que decir en este campo del conocimiento.

Como nota interesante se puede señalar que en esta nueva revolución digital — como denominan diversos gurús y expertos tecnológicos — Europa parece quedar al margen. El ecosistema americano ha engendrado las mayores empresas tecnológicas a nivel mundial. Sin embargo, no están solos, los países asiáticos le siguen en innovación y desarrollo con grandes contribuciones a disciplinas del conocimiento como son la inteligencia artificial, las ciencias de la computación y las telecomunicaciones.

En estos momentos, y en todos en general, la ciencia básica (no solo la aplicada) es clave para poder avanzar hacia una mayor comprensión de los bloques básicos del conocimiento que permiten poder desarrollar nueva tecnología y aplicaciones de esta. Por tanto, no está de más, destacar la importancia de los trabajos de investigación y que este trabajo, valga la redundancia, se encuadra dentro de este punto.

1.1 Campos de aplicación.

Existen diversas parcelas donde tiene lugar el tratamiento de las texturas. Los fines pueden ser muy diversos y entre otros podemos encontrarnos con [1]:

1. Tratamiento de imágenes médicas: Las imágenes se obtienen de diversas técnicas como pueden ser los rayos X, ultrasonidos, tomografías, fotografías microscópicas de biopsias, etc. En varias de estas imágenes las propiedades que presentan, las texturas, son muy importantes para poder dar un diagnóstico acertado. Por ejemplo, en diversos tipos de cáncer, como puede ser el cáncer de mama, el procesamiento de las imágenes ayuda a la detección precoz [2][3].
2. La teledetección: Consiste en poder computar características de un objeto alejado. Entre posibles aplicaciones se pueden citar: la clasificación de terreno, la clasificación de nubes, la cartografía del mar y el reconocimiento de patrones sísmicos. [4].
3. En la industria: Se usa para, por ejemplo, detectar productos con defectos en la cadena de fabricación. De forma manual este proceso se hace tedioso y dificultoso por eso es muy interesante buscar la automatización del proceso. En [5] se describen tres enfoques diferentes de adquisición de imágenes para la inspección visual automática de superficies metálicas.
4. Segmentación de documentos: Se usa para segmentar regiones de texturas homogéneas para poder aplicar el reconocimiento óptico de caracteres/ *Optical Character Recognition* (OCR) que permite digitalizar documentos o extraer parte de ellos [6].
5. Búsqueda de imágenes basadas en el contenido: Permite localizar imágenes con un contenido específico dentro de Internet entre otras aplicaciones [7].
6. Dar forma a partir de una textura: Esto permite reconstruir formas tridimensionales a partir de imágenes bidimensionales.

1.2 Clasificación de las texturas.

Antes de entrar en la explicación de las diversas técnicas empleadas en el reconocimiento de texturas, y cómo están estructuradas; surge la necesidad de partir de una explicación más o menos fidedignas sobre qué entendemos por textura. El hablar de una explicación "más o menos" fidedigna no es casual ya que, aunque parezca extraño, no existe una opinión única sobre qué entendemos por textura. Los conceptos que se barajan son, en muchos casos, subjetivos y no llevan a una definición que se acepte, ampliamente, al carecer de un modelo matemático unívoco [8]. Sin embargo, podemos decir que existen unas propiedades inherentes al concepto de textura como puede ser su asociación a un área que depende del tamaño de la imagen y se percibe como una cierta combinación de patrones menores. La razón por la cual no haya una definición estándar explica, en cierto modo, el por qué hay varios enfoques para acercarnos al reconocimiento de las texturas según queramos obtener unas características u otras y esto repercutirá en la forma de modelarlas. En [9] se detalla más sobre el concepto de textura.

1.2.1 El proceso de clasificación de texturas.

Un sistema general de clasificación de objetos de una imagen, sin ningún tipo de retroalimentación, se muestra en la figura 1. Al término objeto se le puede denominar patrón de forma más genérica. Cuando se habla de reconocimiento de patrones el término patrón no implica, necesariamente, una repetición y se usa para incluir todos los objetos que podrían querer clasificarse como, por ejemplo: gatos, fresas y mesas. Una clase es un conjunto de objetos, o patrones, que son similares, pero no idénticos, completamente, y son distinguibles de otras clases.

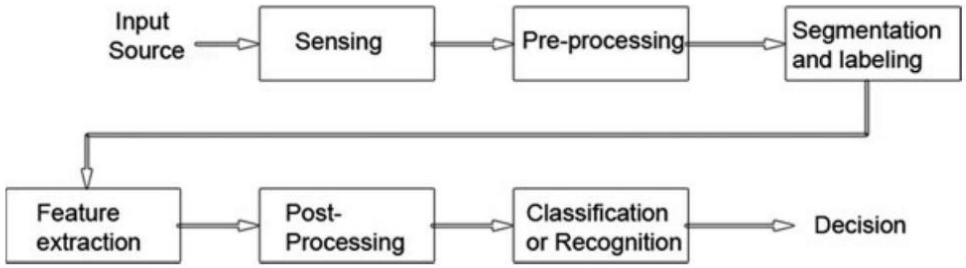

Figura 1: Proceso general de clasificación de una imagen [10].

Las etapas que se describen en el típico sistema de clasificación de patrones mostrado en la figura 1 son [10]:

1. Etapa de detección/adquisición: Se usa un transductor como puede ser una cámara. La señal que se adquiere, en nuestro contexto una imagen, debe de ser de calidad suficiente para que las "características" puedan medirse de forma adecuada. En el caso de una cámara las características son, entre otras, las siguientes: la resolución, el rango dinámico, la sensibilidad, la distorsión, la relación señal a ruido (*SNR*), si está enfocado o no, etc.
2. Preprocesamiento: Se usa para poder condicionar la imagen de cara a la segmentación. Se puede, por ejemplo, filtrar una imagen por una máscara gaussiana para suavizar la imagen, aplicar algún tipo de máscara de mediana para poder eliminar el ruido de tipo sal y pimienta o ecualizar el histograma para poder tener una iluminación uniforme en la imagen.
3. Segmentación: Se trata de dividir una imagen en regiones. Por un lado, se puede querer tener segmentados los objetos de interés y, por otro lado, el fondo. Existen dos enfoques principales:
    a. Los métodos basados en regiones: son aquellos que encuentran regiones conectadas basadas en alguna similitud entre los píxeles de dentro de ellas. Las características más básicas para definir las regiones es el nivel de brillo o gris pero también se puede usar otras características como el color. Sin embargo, tenemos el problema de encontramos con una sobre segmentación de la imagen si necesitamos que los pixeles de una región sean muy similares o, por el contrario, si permitimos poca semejanza en los pixeles de una región, nos encontraremos con una fusión de los objetos. Los métodos que se basan en regiones requieren que se establezca algún tipo de umbral (ya sea global o un umbral local adaptativo — para ello se suele usar un umbral óptimo calculando, por ejemplo, el umbral de Otsu —). Otros métodos basados en regiones incluyen regiones de crecimiento (se usan, en enfoques ascendentes de regiones, los llamados "píxeles semillas") y regiones de decrecimiento (el método opuesto al de crecimiento de semillas) [11].
    b. Los métodos basados en límites: Usan un detector de bordes como el detector de Canny y, además, buscan conseguir límites continuos ya que para tener segmentado un objeto de una imagen es necesario que los borden formen figuras cerradas.
4. Posprocesamiento: Se puede usar como paso previo a la extracción de características. Por ejemplo, con ello se puede eliminar objetos grandes o pequeños en función a un cierto valor que actúe de límite o, bien, ciertos tipos de agujeros presentes en los objetos o en el fondo se pueden rellenados aplicando alguna operación morfológica. Las operaciones morfológicas se encuadran dentro de la teoría de conjuntos y pueden ser aplicadas a cualquier tipo de imágenes. Las más básicas son la dilatación y la erosión de ellas derivan, entre otras operaciones, las conocidas como *opening* y *closing* [12].

5. La extracción de características: Las características son propiedades de los objetos y sus valores, se entienden, que para una clase en concreto deben de ser similares y diferentes de los valores de los objetos de otras clases (o del fondo de la imagen). El paso representado en la figura 1, como extracción de características, permite la reducción de los datos gracias a la medida de algunas

características concretas de los objetos previamente identificados en la segmentación. Los valores de las características pueden ser valores continuos (valores numéricos) o categóricos (con valores etiquetados). La longitud y el área son ejemplos de valores continuos. Por otra parte, los valores categóricos son u ordinales (donde el orden en el etiquetado es significativo — por ejemplo, los rangos en el escalafón militar o el nivel de satisfacción —) o bien, nominal donde el orden del etiquetado no es significativo (por ejemplo, el nombre). La elección de las características adecuadas dependerá de la imagen particular y la aplicación que se busca. Sin embargo, en general, se buscarán una serie de propiedades:

- Robustez: Normalmente, las características deberán ser invariantes a la orientación(rotación), cambio de escala, cambio de las condiciones de iluminación de la imagen y, en la mayor medida, se buscan que sean invariantes a la presencia de ruido y valores espurios (esto requerirá, por lo general, algún tipo de preprocesamiento de la imagen como se comentó en la etapa 2).
- Discriminación: Los rangos de valores de los objetos de las diferentes clases deberían de ser diferentes y, preferentemente, bien separados y que no den lugar a solape entre ellos.
- Valores de confianza: los objetos de la misma clase deberían de tener valores similares.
- Independientes: Es decir, se buscarán características no correlacionadas. Por ejemplo, existen características correlacionadas como pueden ser la longitud y el área.

En [10] y [13] se indica que las características son representaciones de alto nivel de la estructura y la forma (de los objetos que forman una imagen) y deberían de escogerse para preservar la información que es importante para la tarea particular en cuestión. Dentro de las características podemos decir que tenemos aquellas que describen lo que contienen los objetos y aquellas que describen su forma.

Entre las características que describen el contenido de los objetos tenemos:

- Características obtenidas a partir del histograma de un objeto usando el procesamiento de la región de interés
- La textura de un objeto, para ello se usa momentos estadísticos del histograma de niveles de grises del objeto o su dimensión fractal.

Entre las características que describen la forma tenemos:

- El tamaño o área de un objeto que se obtiene directamente del número de píxeles que comprenden un objeto.
- La circularidad que se obtiene mediante el perímetro y el área del objeto en cuestión.
- La transformación del eje medio que produce una imagen en escala de grises donde cada punto del esqueleto tiene una intensidad que representa su distancia a un límite en el objeto original. Se puede usar para la reconstrucción del objeto original.
- El número de Euler que es el número total de objetos de la imagen restando el número total de agujeros en esos objetos.
- Distintos momentos estadísticos en 2-D para aplicar a una imagen digital. Si los momentos son definidos en torno a la media se les denominan momentos centrales. El segundo momento central es la varianza y la *kurtosis*, que como se verá está muy relacionado con *ICA*, es el cuarto momento central(normalizado).

Las características se representan mediante un vector, x, que contienen las características medidas (x_1, x_2, …, x_n) para un objeto particular o una región. Los vectores de características se pueden mostrar como puntos en el

espacio de características. Para n características, el espacio de características será n–dimensional siendo cada característica una dimensión. Los objetos de una misma clase deberían de ser agrupados juntos en el espacio de características y estar muy bien separados de las distintas clases. En la parte de clasificación la meta es conseguir asignar cada vector de características a una de las clases.

Es importante hacer notar que el número de características influye notablemente en el tiempo de computación que se manejará. Por tanto, en la medida de lo posible hay que jugar con poder tener un número de características suficiente que permita llegar a una clasificación, pero sin dejar de lado el tiempo computacional que se invertirá en ello. A veces más características pueden llevar a una mayor carga computacional, pero sin resultados verdaderamente mejores. Por tanto, es un buen punto de partida el intentar encontrar unas características que de verdad arrojen luz para poder clasificar. Hay técnicas como *PCA (Principal Component Analysis)*, y la usada en el presente documento *ICA*, que buscan las características que más información aportan. Por ejemplo, *PCA,* como método muy conocido de extracción de característica, arroja un vector de características y (no correlacionadas) que forman una combinación lineal de las características originales x.

6. Clasificación: Según [12] el objetivo de la clasificación es identificar características, patrones o estructuras dentro de una imagen y usarlos para asignar la imagen a una clase en concreto. También engloba el hecho de clasificar una imagen completa en una clase en particular.

1.2.2 Enfoques en la Clasificación

Las técnicas de clasificación se pueden agrupar en dos tipos, principalmente, supervisadas y no supervisadas. Las técnicas de clasificación supervisadas se basan en el hecho de tener patrones de ejemplo, o vectores de características, que ya han sido asignados, previamente, a una clase. Con una parte de este conjunto formamos un grupo de entrenamiento que nos permitirá diseñar un sistema de clasificación al cual probaremos, su veracidad, con otra parte de los vectores de características que no usamos en la parte de entrenamiento (denominada también aprendizaje lo que se busca es reducir el error de clasificación dentro del conjunto usado para el entrenamiento). En la clasificación supervisada, el objetivo es utilizar ejemplos, un conjunto de entrenamiento, que permita diseñar un clasificador que se generalice bien a nuevos ejemplos. Por otra parte, la clasificación no supervisada no se basa en que existan ejemplos de una clase de patrón conocida. Los ejemplos, en este caso, no están etiquetados y se busca identificar grupos directamente dentro del conjunto de datos y características que nos permita distinguir un grupo de otro [12].

Cabe señalar que la parte que se denomina aprendizaje es parte del muy conocido *machine learning* o inteligencia artificial que tan en boca está en el momento de escribir estas líneas. Básicamene, esto se refiere a alguna forma de adaptar el algoritmo de clasificación para que se vaya comportando mejor, es decir, vaya dando unos mejores resultados con menores errores en la clasificación del conjunto usado para el entrenamiento. Esta última parte añadiría un punto de complejidad a la figura 1, donde se tiene un modelo más simple, porque se está hablando de retroalimentar el sistema con el fin de reducir el error en la clasificación.

Según [14] los cuatro enfoques más conocidos en el reconocimiento de patrones son:

1. *Template matching* (correspondencia de patrón): Es una técnica para detectar una imagen que más se parezca a una plantilla o patrón dado. Esta plantilla representa el objeto que se quiere detectar. El patrón para reconocer se compara con la plantilla almacenada teniendo en cuenta las posibles rotaciones, translaciones y cambio de escalas. Básicamente, se irá desplazando la plantilla por cada píxel de la imagen y computando la diferencia. Donde la diferencia sea menor, ahí, se tendrá la porción de la imagen con mayor similitud. En la figura 2 se representa este proceso.

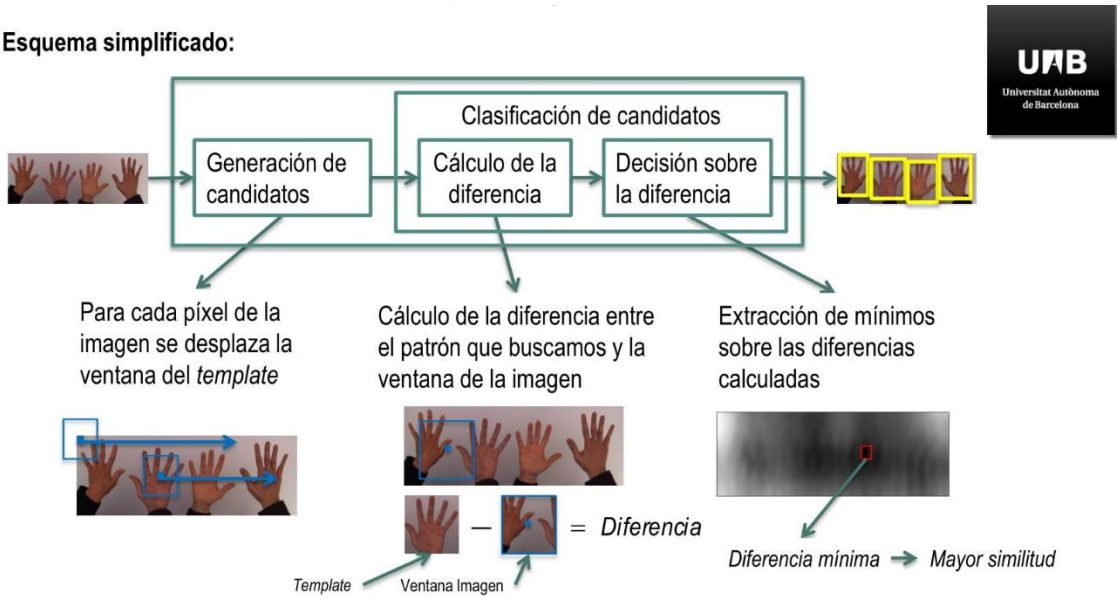

Figura 2: Representación del proceso Template matching.[1]

2. Enfoque estadístico: Nos encontramos según [13] con dos enfoques generales, el enfoque paramétrico y el no paramétrico. Los métodos paramétricos se valen de distribuciones de probabilidad y estiman los parámetros derivados de ellos, como son la media y la desviación estándar, para proporcionar una representación compacta de las clases. Como ejemplo se puede citar el análisis discriminante. Los métodos paramétricos suelen ser lentos en la parte de entrenamiento, pero una vez entrenados son rápidos en la fase de test. Por otro lado, los métodos no paramétricos estiman las distribuciones de probabilidad (estimación no paramétrica) o evitan las probabilidades y van directamente a computar las funciones de decisión.

   El análisis discriminante, ejemplo de enfoque paramétrico, es una técnica estadística que se utiliza para clasificar mediante límites de decisión entre las clases. Estos límites se denominan funciones discriminantes que se obtienen de clasificadores como el clasificador de Bayes que se sustenta, de forma teórica, en el famoso teorema de Bayes y en la teoría de la probabilidad. Como ejemplo de técnicas no paramétricas tenemos el clasificador *k-nearest-neighbors* (k-NN). (En el apéndice A tenemos un breve acercamiento a este clasificador).

3. Enfoque sintáctico o estructural: En muchos problemas de reconocimiento que involucran patrones complejos es más apropiado adoptar una perspectiva jerárquica donde un patrón es visto como una composición de sub-patrones más simples y estos a su vez están formados por sub-patrones aún más simples. Los patrones más simples/elementales para ser reconocidos se llaman primitivas y los patrones complejos están representados en términos de interrelación entre estas primitivas. Este tipo de enfoque aporta una descripción de cómo el patrón dado es construido a partir de las primitivas. Este paradigma ha tenido aplicación en situaciones donde los patrones tienen una estructura determinada que puede ser obtenida aplicando un conjunto de reglas (como por ejemplo imágenes de texturas como es el caso del presente trabajo).

   En [15] se indica que una textura puede ser vista como un conjunto de primitivas denominadas *texels* en una relación espacial particular. Los *texels* con regiones de imágenes que puede ser extraídas a través de algunos procedimientos aplicando algún tipo de umbral a la imagen.

   En [16] se indica que, en lugar del término *texel*, la mayoría de los trabajos usan el término *texton* para referirse a pequeños objetos o áreas específicas que juntos forman una textura. El término *texton* fue acuñado por Julesz [17] pero ya J.J. Gibson en [18] hablaba de un elemento de textura (*texton* o *texel*) como el bloque de construcción básico que define una textura en particular. Sin embargo, en [19] se comenta que, aunque ambos términos se usan indistintamente existe un matiz: los *texels* tienen

[1] La figura pertenece a la UAB, departamento de Ciencias de la Computación, profesora María Vanrell.

analogía con los píxeles y definen una partición de la textura sin que haya una superposición espacial. Por otro lado, los *textones* tienen características estadísticas y se calculan en cada píxel sin preocuparse por la superposición. En el presente trabajo nuestro enfoque de primitiva será el uso de los *textones*. Por tanto, se adelanta que en este trabajo el acercamiento al reconocimiento de patrones, en particular texturas, se hará mediante *textones*.

4. Redes neuronales: Estos modelos se basan en el sistema nervioso humano. Las neuronas son células autónomas especializadas en transmitir, procesar y almacenar información lo cual es indispensable para que podamos responder a los estímulos. Básicamente, el modelo de una neurona es transmitir impulsos de electricidad a través de un largo hilo que se denomina axón. El axón se divide en miles de ramas donde pueden "disparar" a través de una brecha (sinapsis) a las dendritas de otras neuronas. Existe un modelo matemático para la neurona denominado perceptrón (siendo este el modelo más simple) y busca emular a una neurona biológica [10].

Una red de neuronas artificiales es un sistema paralelo que es capaz de resolver problemas que un sistema lineal no puede resolver. Como su contraparte biológica, la red de neuronas artificiales es un sistema adaptativo (los parámetros pueden variar durante el entrenamiento para adaptarse al problema). La principal diferencia de este enfoque mediante neuronas artificiales con los otros enfoques es la capacidad que tienen las neuronas de aprender relaciones complejas de entradas y salidas. Cuando se van sumando capas a las redes neuronales, aumentan en capacidad para los cálculos, pero esto, a su vez, aumenta la dificultad de las operaciones y, por consiguiente, el tiempo computacional. Las redes neuronales proveen de algoritmos no lineales para la extracción de características (usando capas ocultas) y para la clasificación (usando perceptrones multicapa). Existen muchos modelos dentro de las redes de neuronas artificiales que tienen su equivalente o similar en el enfoque de patrones estadístico [14].

En la siguiente figura se puede ver un modelo de neurona biológica (a) y un modelo de neurona artificial (b). Podemos ver que la parte del detector de umbral se dispara cuando se supera un cierto valor y tiene su equivalencia en el proceso de la sinapsis de la neurona biológica.

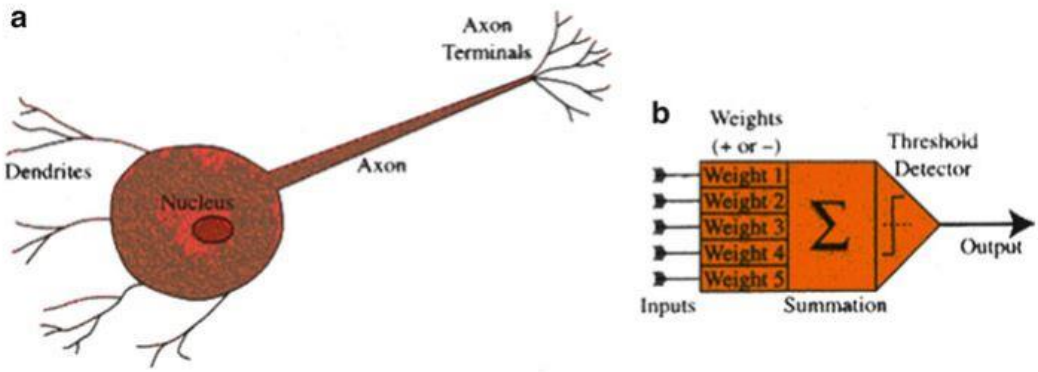

Figura 3: Modelo de neurona biológica y neurona artificial [10].

1.3 Contexto del trabajo.

El presente trabajo consiste en la creación de un clasificador apoyado en el uso de los denominados *textones*. Para ello, como base de datos de texturas, se trabaja con la base de datos Columbia-Utrecht (en adelante, *CUReT*) que contiene 61 tipos diferentes de texturas y cada textura consta, a su vez, de 205 imágenes obtenidas bajo diferentes condiciones de visión e iluminación. De estas 205 imágenes, por clase de textura, se trabaja con una versión reducida de la misma. Esta versión reducida consta de 92 imágenes por clase de textura que son seleccionadas como sigue: para cada textura en la base de datos hay 118 imágenes donde el ángulo de visión acimutal es menor a 60 grados. De éstas, elegimos 92 imágenes de las cuales se puede obtener una región suficientemente grande para todas las clases de texturas, en este caso, se trabajará con recortes de texturas de 200x200 pixeles convertidos a escala de grises y luego normalizados para que presenten media cero y una desviación estándar unitaria.

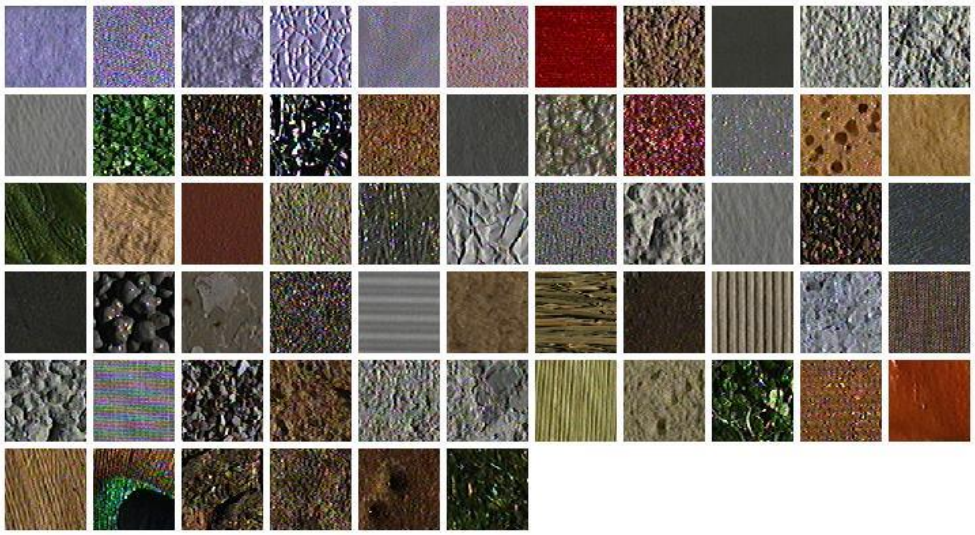

Figura 4: Base de datos CUReT[4]. En este trabajo se trabaja con imágenes monocromáticas.

1.4 Objetivo.

En [20] los autores hacen una comparativa entre las ratios de clasificación para un conjunto de cuatro bancos de filtros usados por el algoritmo de clasificación que proponen. Como novedad este trabajo aporta el uso de bancos de filtros obtenidos usando el algoritmo *ICA* (Independent Component Analysis). Estos bancos de filtros serán dependientes de la base de datos de imágenes utilizadas por el algoritmo tanto en la parte de entrenamiento y en la parte de testeo.

2 ESTRUCTURA DEL TRABAJO

"It's not only the question, but the way you try to solve it. "

-Maryam Mirzajani-

Como hemos comentado en el punto anterior, el desarrollo de este clasificador se nutre del trabajo realizado por Manik Varma y Andrew Zisserman[20]. El clasificador, básicamente, consta de una parte de aprendizaje y otra de clasificación. La parte de aprendizaje la podemos dividir, a su vez, en dos partes diferenciadas. En la primera parte se genera el diccionario de textones [21], en la segunda se generan los modelos, a partir de las imágenes de entrenamiento y, por último, en una tercera parte se clasifican las nuevas imágenes, las denominadas imágenes de test.

Antes de meternos en la explicación del algoritmo, propiamente dicho, nos pararemos en definir una serie de banco de filtros, tanto los usados en [20] como las propiedades del banco de filtros que se usan en el trabajo, a saber: el banco de filtros ICA (*Independent Component Analysis*).

Estos bancos de filtros servirán para filtrar las diferentes imágenes de entrenamiento y, a partir de la respuesta, obtener un diccionario de textones (ver sección 2.3).

2.1 Bancos de filtros a comparar con el banco de filtros ICA.

En [20] el diccionario de textones se obtiene a partir de, previo filtrado de las imágenes, cuatro tipos de bancos de filtros. Como ya adelanté, en la introducción, en este trabajo se filtran las imágenes con otro banco de filtros diferente, el banco de filtros ICA. A continuación, detallo los diferentes bancos de filtros usados por [20],[21] [22]:

1. El banco de filtros de Leung-Malik (LM) [22]. Este banco está formado por 48 filtros de diferentes escalas y orientaciones. Contiene primeras y segunda derivadas de gaussianas en 6 orientaciones y 3 escalas haciendo un total de 36, 8 laplacianas de filtros de gaussianas (LoG) y 4 Gaussianas.

2. El banco de filtros de Schmid(S) que consiste en 13 filtros rotacionalmente invariantes según la fórmula:

$$F(r,\sigma,\tau) = Fo(\sigma,\tau) + \cos\left(\frac{\pi\tau r}{\sigma}\right) e^{-\frac{r^2}{2\sigma^2}} \qquad (2.1)$$

$Fo(\sigma,\tau)$ se añade como componente de continua. El parámetro r representa las coordenadas x e y de un píxel, σ es la escala, τ es la frecuencia entendida como el número de ciclos de la función harmónica dentro de la envolvente gaussiana [1,7]

3. El banco de filtro de máxima respuesta (MR) consistente en 38 filtros de múltiples orientaciones, pero solo 8 respuestas a los filtros. La máxima respuesta en todas las orientaciones consiste en reducir el número de respuestas de 38 filtros (formados por una gaussiana y una laplaciana de la gaussiana(*LoG*) ambas con σ =10, un filtro de borde — *edge filter* — en tres escalas, a saber: (σ_x, σ_y)={(1,3) ,(2,6),(4,12)} y un filtro de barras — bar filter —) a un banco de filtros MR8 formado por las respuestas en tres escalas diferentes para los "edge filters" y "*bar filters*", además de, la respuesta al filtro gaussiano y al *LoG*[1,7].

4. El banco de filtros MR4 consiste en un subconjunto del banco de filtros MR8 donde los filtros orientados "*edge*" y "*bar*" se tienen en una única escala (σ_x=4 y σ_y=12).

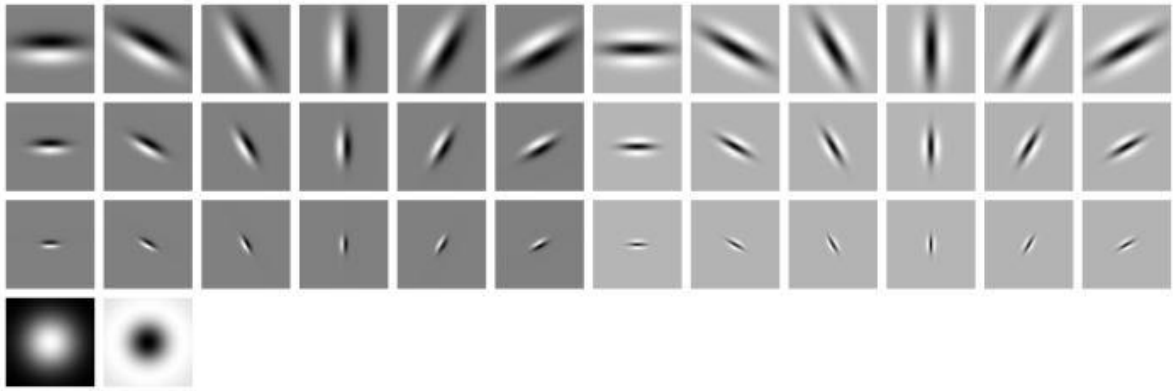

Figura 5: Banco de filtros MR38 del cual se obtienen el banco MR8 y MR4 [20].

2.2 Análisis de componentes independientes.

Llegado a este punto, quizás, el lector haya notado un detalle que comparten en esencia los bancos de filtros expuestos en 2.1. Todos estos bancos de filtros son filtros independientes de la imagen, su formación, no tienen en ningún momento en cuenta la textura con la que se pueda trabajar son, por decirlo de algún modo, filtros independientes.

Sin embargo, los filtros con los que se compara en el presente documento son filtros que tienen por característica, su firme dependencia con las texturas con las que se trabaja. Pues son filtros que se obtienen a partir de estas texturas, en concreto, se obtendrán de las imágenes de entrenamiento. En cierto sentido podemos decir que no usaremos filtros, las propias texturas nos los proporcionarán.

Escrita esta pequeña introducción pasaré a dar una breve pincelada sobre la técnica *ICA* y sobre cómo se obtienen los filtros.

El objetivo que tiene el análisis en componentes independientes, *ICA*, es encontrar una representación de los datos estadísticamente independientes [24] [25]. Es decir, en el presente documento, buscamos descomponer una señal (imagen escala de grises) en una combinación lineal de fuentes independientes. *ICA* está muy relacionada con el método denominado separación ciega de fuentes (Blind Source Separation — *BSS*—).

Desde los experimentos de Hubel y Wiesel (1962,1968) [26], [27], ha habido un gran número de estudios en los que las propiedades de los campos receptivos de las neuronas en el córtex visual primario han intentado medirse y clasificarse ejemplos en [28] y [29]. En [30] Barlow's propone que las neuronas actúan reduciendo

los datos de entradas representándolos con componentes independientes.

De aquí se infiere que, para cualquier imagen, solo unas características particulares son necesarias para su representación. Esto lleva a la idea de dividir una imagen en patches y representarlos como combinaciones de ciertos "patches bases" [31]. A continuación, se reproduce el modelo *ICA*.

$$x = As = \sum_{i=1}^{n} a_i s_i \qquad (2.2)$$

x representa la imagen de entrada y las imágenes bases las a_i, las columnas de A. Las componentes s_i son las llamadas componentes independientes y son variables latentes lo que significa que no pueden ser observadas directamente, son conocidas también como las variables ocultas, "las fuentes" o señales originales. La matriz de mezcla A también se asume como desconocida. En definitiva, a partir de x, obtendremos las componentes a_i y s_i. La matriz A, será la matriz de mezcla. formada por las bases que usaremos en el proceso de filtrado

Figura 6: Modelo de combinación lineal de los "patches base". Fuente [31].

Es posible demostrar que el modelo actual tiene solución si y solo si las componentes s_i son no gaussianas.

2.2.1 Restricciones a la solución aportada por ICA

Para estar seguros si el modelo *ICA* puede ser estimado tenemos que hacer algunas suposiciones y poner ciertas restricciones. Me limitaré a nombrar estas restricciones y dar alguna breve pincelada sobre las mismas. En [24] y [32] se podrá obtener información más exhaustiva.

1. Se asumirá que las componentes independientes, es decir, las componentes s_i son independientes. Este es el principio que sustenta *ICA*. Sorprendentemente, con esta suposición el modelo puede ser estimado. Este es el motivo por el cual *ICA* es un método potente con aplicaciones en muchas áreas diferentes.
   Básicamente, se dice que una variable aleatoria x_i es independiente si la información en la variable x_i no da ninguna información de la variable x_j para $i \neq j$.

2. Las componentes independientes deben tener distribuciones no gaussianas. Eso significa podemos medir la "no gaussianidad" usando, por ejemplo, el estadístico de orden superior denominado *kurtosis*. Este estadístico se mide usando los momentos de tercer y cuarto orden [32]. En el caso de que las variables aleatorias presenten media igual a cero, la kurtosis queda definida tal que así:

   $$kurt(z) = E[z^4] - 3(E[z^2]^2) \qquad (2.3)$$

   Es posible demostrar que si z tiene una distribución gaussiana se obtiene una kurtosis de valor cero. Por tanto, la kurtosis, se puede usar como cierta garantía de la "no gaussianidad". No obstante, hay que indicar que posee algunos inconvenientes como ser muy sensible a muestras extremas que puedan surgir por error, a la hora de obtenerlas, o bien, simple y llanamente presenta mucha sensibilidad a valores que, realmente, son irrelevantes.

Por ello, surge la necesidad de tener otra forma de medir la no gaussianidad de una variable aleatoria. Se define para ello la medida neguentropía, siendo la segunda medida más importante para mediar la no gaussianidad.

La neguentropía está basada en la teoría de la información mediante el concepto de entropía presentando una variable aleatoria un mayor grado de aleatoriedad al ser mayor su entropía. La teoría de la información nos aporta, como uno de sus resultados, que, dado un conjunto de variables de igual varianza, k, aquella que sea gaussiana nos dará la mayor entropía, por tanto, se concluye que la neguentropía es un buen indicador para medir la no gaussianidad siendo cero, solamente, para distribuciones gaussianas. Se define matemáticamente como:

$$J(y) = H(y_{gauss}) - H(y) \qquad (2.4)$$

Siendo H la entropía de una variable aleatoria mientras que y_{gauss} es una variable aleatoria gaussiana de la misma correlación (y covarianza) que la matriz y (variable aleatoria) [32].

3. La matriz de mezcla A es cuadrada y esto implica que el número de componentes s_i a estimar es igual al número de observaciones. A partir de esta suposición podemos obtener la matriz de mezcla, y si se computa la inversa, se puede obtener la matriz W que nos permite obtener las fuentes originales:

$$s = Wx \qquad (2.5)$$

Se asume con ello que la matriz de mezcla es invertible. Si no fuese invertible habría componentes en la matriz de mezcla que serían redundantes, en cuyo caso, la matriz no sería cuadrada y nos encontraríamos en una situación donde el número de componentes en la matriz de mezcla y el número de componentes independientes sería diferente.

2.2.2 Ambigüedades de ICA.

Del modelo de *ICA* se extraen las siguientes indeterminaciones [24] [32]:

1. No podemos determinar las varianzas(energías) de las componentes independientes.
Esto se debe a que al ser A y s desconocidas, cualquier escalar μ_i que multiplique a una de las fuentes s_i podría cancelarse dividiendo la columna correspondiente, a_i, de A por el mismo escalar μ_i:

$$x = \sum_{i=1}^{n} (\frac{1}{\mu_i} a_i)(s_i \mu_i) \qquad (2.6)$$

Lo habitual en la forma de proceder es suponer que las componentes independientes, s_i , tienen varianza igual a la unidad. $E[{s_i}^2] = 1$. La matriz A se calculará teniendo en cuenta esta restricción. Hay que indicar también que existe ambigüedad con el signo porque podríamos multiplicar cualquier componente independiente por -1 sin que ello afectase al modelo. Sin embargo, esta ambigüedad es insignificante en la mayoría de los casos.

2. No podemos determinar el orden de las componentes independientes.

   La razón es, otra vez, que tanto A como s son desconocidas y podemos cambiar fácilmente el orden de los términos del sumatorio en (2.2). Cualquiera de las componentes independientes podría ser vista como la primera. De forma formal, una matriz de permutación P y su inversa P^{-1} pueden ser sustituidas en el modelo (2.2) para dar:

$$x = AP^{-1}Ps \qquad (2.7)$$

Los elementos de Ps son las variables independientes,s_i, originales, pero en un orden diferente. La matriz AP^{-1} sería una nueva matriz de mezcla que se obtendría aplicando la técnica *ICA*.

2.2.3 FastICA

Según lo que vemos en [33] tenemos dos enfoques para la implementación del algoritmo de forma computacional en el entorno *MATLAB*(MathWorks®) que es el usado en el presente trabajo. Por un lado, tenemos los métodos geométricos que se basan en la distribución geométrica de las señales provenientes de las fuentes y, a partir de estas, obtener los parámetros de la matriz de mezcla A. Podemos citar entre otros el método *LatticeIca*, Método Geométrico por Sectores o el Método Separación de Neuronas. Por otro lado, tenemos los métodos estadísticos basados en, como su nombre indica, propiedades estadísticas ya sea la kurtosis u otras diferentes. En este segundo grupo se sitúa el algoritmo usado para estos experimentos *FastICA*. Además, podemos citar otras implementaciones como son el método *Jade* o el método *Infomax*.

En [34] se tiene una explicación de *FastICA*, la implementación numérica de ICA más popular, basada en un algoritmo de punto fijo y, básicamente, consiste en maximizar estadísticos de orden superior como la kurtosis [35].

2.2.3.1 Preprocesado con los datos que recibe FastICA.

FastICA trabaja con datos que, previamente, han sido expuestos a un preprocesamiento que consiste en centrarlos y "blanquearlos", a saber:

1. El proceso de centrado se basa en dada una matriz de datos M obtener su media $E[M]$ y quitársela a M. De tal forma la media de los datos será igual a cero y se tendrá que las componentes independientes s también tendrán media cero.
2. El proceso de blanqueado consiste en obtener un vector cuyas componentes sean no correladas y su varianza unitaria, o bien, desde el punto de vista de la covarianza, el nuevo vector, $\tilde{x}$, tendrá como matriz de covarianza la matriz identidad I. Siendo $\tilde{x}$ el vector de datos blanqueados y x el vector de datos centrados:

$$\tilde{x} = VD^{-\frac{1}{2}}V^T x = VD^{-\frac{1}{2}}V^T As = \tilde{A}s \qquad (2.8)$$

Donde se usa la descomposición en autovalores y autovectores teniendo en cuenta que:

$$E[xx^T] = VDV^T \qquad (2.9)$$

Siendo V una matriz ortogonal de autovectores y D la matriz diagonal de autovalores.

El proceso de blanqueado nos permite reducir el número de parámetros que deben ser estimados.

2.2.3.2 Breve acercamiento al algoritmo FastICA.

En este punto se da una breve pincelada del algoritmo *FastICA* para un único vector w. En [24], [32] y [34] se tiene más información al respecto.

Como se expone en el punto 2.2.1 existe una matriz W que permite obtener las componentes independientes s_i. Basado en el concepto de neguentropía, definido en 2.2.1, se puede definir el concepto de información mutua, I, que es una medida de la dependencia entre variables aleatorias. Expresando I, a partir de la negentropía:

$$I(y_1, y_2, \dots, y_n) = \sum_i^n J(y_i) \quad (2.10)$$

La W se determinará para que la información mutua de las componentes independientes $s_i$sea minimizada [34].

El algoritmo se basa en encontrar una dirección del vector de pesos w tal que la proyección $w^T x$ maximice la no gaussianidad. Se mide aquí la no gaussianidad, a partir de la negentropía $J(w^T x)$, siendo $J(*)$:

$$J(y) \propto \ c[E\{G(y)\} - E\{G(v)\}]^2 \quad (2.11)$$

Donde G es una función no cuadrática, c es una constante irrelevante, y es una variable aleatoria de media cero y varianza unidad y v es una variable aleatoria gaussiana, también, con media cero y varianza unitaria.

Es importante hacer notar que la varianza de $w^T x$ debe de ser unitaria o, lo que es lo mismo, para el preprocesado previo donde sometemos los datos a un blanqueamiento esta condición se traducirá en tener norma unitaria para el vector w. Para medir la no gaussianidad *FastICA* se apoya en una función no lineal no cuadrática $G(u)$, su primera derivada $g(u)$ y su segunda derivada $g'(u)$, tal y como se describe a continuación:

$$G_1(u) = \frac{1}{a_1} logcosh(a_1 u), \qquad g_1(u) = \tanh(a_1 u) \quad (2.12)$$

Son definiciones útiles para propósitos generales, pero si las componentes independientes son supergaussianas o se busca robustez las siguientes definiciones podrían ser una mejor opción [19]:

$$G_2(u) = -\frac{1}{a_2} e^{-\frac{a_2 u^2}{2}}, \ \ g_2(u) = u e^{-\frac{a_2 u^2}{2}} \quad (2.13)$$

Donde $1 \leq a_1 \leq 2$, $a_2 \leq 1$.

Los pasos que se siguen en las iteraciones del algoritmo para estimar una sola componente independiente son:

1. Se escoge un vector de pesos w, por ejemplo, aleatorio.
2. $w^+ = E[x g(w^T x)] - E[g'(w^T x)] w$
3. $w = \frac{w^+}{||w^+||}$

4. Si no converge se vuelve al paso 2. Cabe destacar que la convergencia significa que los viejos y los nuevos valores de $w$apuntan en la misma dirección o, visto de otra forma, su producto escalar es casi igual a la unidad [24].

Para estimar varias componentes independientes necesitamos varios vectores w_i y para evitar que varios vectores acaben convergiendo al mismo máximo se deben "decorrelar" las salidas ${w_1}^T x, \dots, {w_n}^T x$ después de cada iteración. Para conseguir esto existen varios métodos diferentes se recomienda la consulta de [24]. [32],[34].

Por último, conviene señalar algunas de las propiedades que hacen que se use el algoritmo *FastICA*, a saber [34]:

1. Posee convergencia cúbica o, al menos, cuadrática frente a métodos del gradiente donde la convergencia en lineal.
2. No hay parámetros de paso que elegir y si los hay son fáciles de escoger lo que se traduce en un algoritmo fácil de usar.
3. Encuentra componentes independientes de, prácticamente, cualquier distribución no gaussiana usando cualquier función g.
4. El rendimiento del algoritmo se puede optimizar por escoger una función g adecuada.
5. Las componentes independientes se pueden obtener una por una.
6. Los algoritmos de punto fijo, como *FastICA*, tienen la mayoría de las ventajas de los algoritmos neuronales. *FastICA* es distribuido, paralelo y necesita poco espacio de memoria.

2.3 Obtención del banco de filtros ICA.

El banco de filtros *ICA* se obtiene a partir del conjunto de imágenes de entrenamiento que, previamente, se haya estipulado usar. Es decir, si el conjunto de imágenes de entrenamiento está formado por P clases de imágenes con K imágenes por cada clase. El conjunto de $P * K$,imágenes totales, se usarán para la obtención del banco de filtros *ICA*. De aquí se comprueba la dependencia que tienen los filtros *ICA* con las imágenes de entrenamiento. Para nosotros los filtros *ICA* consistirán en las columnas de la matriz de mezcla A ,obtenidas gracias al algoritmo *FastICA*, convenientemente transformados para formar filtros de 2D [36]. Este proceso se hará tanto para las imágenes que se usarán para obtener el diccionario de testones, así como también con las imágenes de entrenamiento. Es importante remarcar que el algoritmo *FastICA*, implementado en *MATLAB®*, permite reducir el número de bases para quedarse con las más relevantes. Esto se lleva a cabo gracias al parámetro "*lastEig"* que permitirá tener respuestas mejor o peor tratables de forma computacional. En la figura 3 podemos ver el proceso de división de una imagen cualquiera en bloques de tamaño $\sqrt{N} * \sqrt{N}$ para después listarlos por columnas de tamaño N quedando T columnas de N filas, cada fila representa una observación, por tanto, formarán la matriz de observaciones X la cual se pasa al algoritmo *FastICA* para obtener la matriz de mezclas A.

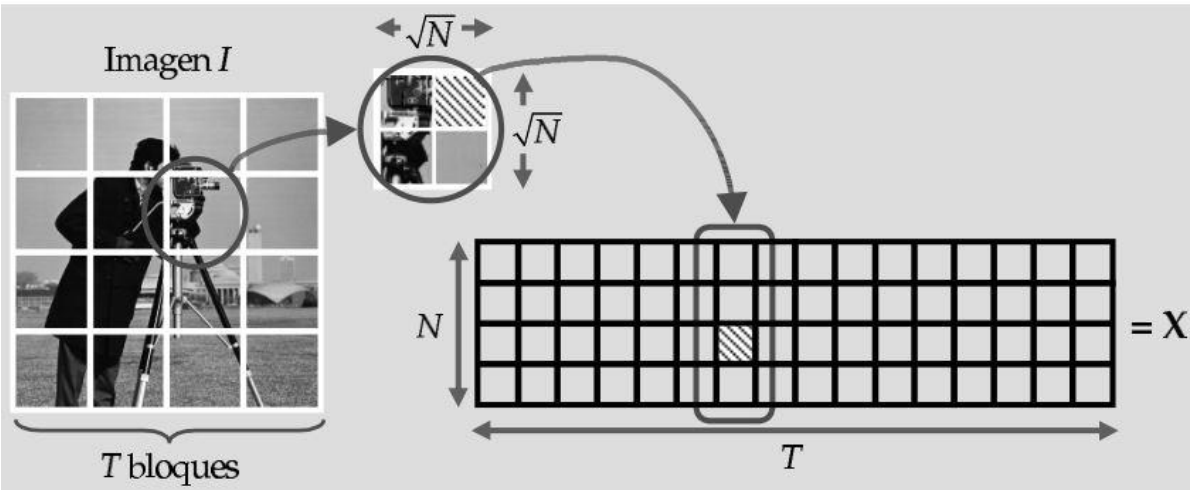

Figura 7: División de una imagen cualquiera en patches y listados de los mismos en columnas [47].

El tamaño del banco de filtros quedará, por tanto, establecido en función del tamaño que tenga el conjunto formado por la matriz de observaciones X, que estará formada por observaciones obtenidas de las clases de texturas con las que trabajemos. En concreto, para una un grupo de imágenes más amplio se tendrá el mismo número de filas, N, y como columnas tendremos T por el número de imágenes que se consideren, por ejemplo 20 texturas formadas por 46 imágenes por clase, ordenadas todas las columnas de forma aleatoria. La matriz de bases A ,que devolverá *FastICA*, tendrá como número de filas el valor de los patches, N,y por número de columnas, aunque se ha visto previamente que la matriz A es cuadrada, tendrá una dimensión reducida habiendo obtenido el número de componentes que se consideran óptimas mediante pruebas computacionales. Estas componentes consideradas óptimas serán las bases dominantes.

2.4 Pasos previos de preprocesamiento.

Antes de ejecutar el algoritmo será necesario estandarizar de alguna manera las imágenes con las que se trabajan. Ya que en [20] se trabaja con imágenes de tamaño 200x200 pixeles sacadas de la base de datos *CUReT*[37] es el enfoque se usa en este trabajo, es decir, todas las imágenes con las que se trabajan,tanto para la parte de entrenamiento, así como, para la parte de testeo serán de tamaño 200x200 pixeles, en escala de grises y normalizadas para presentar media igual a cero y varianza unidad.

Todos los bancos de filtros, en este caso los bancos de filtros *ICA* son normalizados con norma L_1. Cada filtro F_i del banco de filtros se divide entre $||F_i||_1$ para que el filtro tenga norma L_1. Por último, las respuestas al banco de filtros *ICA*, en cada pixel, x, son normalizadas tal que así:

$$F(x) \leftarrow F(x)[\log(1 + \frac{L(x)}{0.03})]/L(x) \qquad (2.14)$$

Donde $L(x) = ||F(x)||_2$.

2.5 ¿Qué se entiende por texton?

Previamente, en el capítulo 1, se da una introducción sobre el concepto de *texton*. En este apartado se desarrolla un poco más el concepto.

Según [17] el concepto de *texton* queda ligado a las microestructuras fundamentales en las imágenes naturales (y en vídeos) y se consideran como "los átomos" en la visión humana, los elementos básicos de la visión. O bien, se podrían definir como las respuestas que mejor representan el resultado de filtrar una imagen con un banco de filtros determinado [38], por ejemplo, cualquier banco de filtros de los expuestos previamente en la memoria.

En [39] se presenta un modelo de generación de imágenes que consta de tres niveles que explica el aprendizaje de *textones* a partir de imágenes de texturas. En ese modelo una imagen, **I,** es una superposición de varias bases de imágenes seleccionadas de un diccionario completo, **Ψ,** que incluye varias funciones de Gabor y laplacianos de Gauss (*LoG*) de diferentes escalas y orientaciones. Se representan las bases de imágenes como puntos y se denotan como un mapa de bases **B**. Este mapa de bases, **B**, se genera a su vez por un número menor de imágenes son, a su vez, generadas por un número menor de *textones*, que se denota por **T.** Este conjunto es seleccionado de un diccionario de *textones* **Π.** En este modelo tanto el mapa de bases, **B,** como el mapa de *textones*, **T,** están ocultos (son variables latentes) y los diccionarios **Ψ** y **Π** son parámetros que deben de aprenderse a través de la adaptación del modelo a las imágenes observadas. En la siguiente figura se esquematiza el proceso comentado:

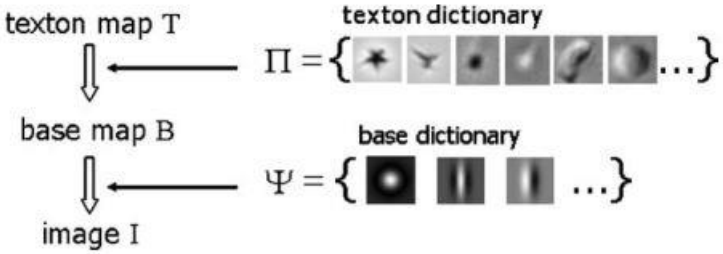

Figura 8: Modelo generativo de tres niveles de una Imagen. Referencia [38].

En [39] también podemos ver que el *texton* se define como objetos significativos que se ven a distancia y que representan de forma diferente según sea la estructura geométrica, dinámica o fotométrica:

1. Estructuras geométricas: Un *texton* consiste en un pequeño número de bases de imágenes con configuraciones espaciales deformables. Las estructuras geométricas se aprenden de una imagen de textura estática con elementos repetidos.

2. Estructuras dinámicas: El movimiento de un *texton* se caracteriza por un modelo de cadena de Markov que puede modificar las configuraciones geométricas a lo largo del tiempo. Los modelos de cadena de Markov se aprenden utilizando las trayectorias de los *textones* y las imágenes que forman las bases se infieren de las secuencias de vídeo.

3. Estructuras fotométricas: Para este tipo de estructuras un *texton* representa un elemento de superficie tridimensional bajo distintas iluminaciones y se le denomina "*lighton"*. Este elemento está formado de tres *textones* en 2D. Para una fuente de luz dada, un "*lighton"* se genera como una suma lineal de los tres *textones*.

En [40] se denomina *textones* a cuatro plantillas de tamaño 2x2. Si los dos píxeles resaltos en colo gris tienen el mismo valor, se formarán un *texton*. En la siguiente figura se puede apreciar la forma de estas plantillas que dentro de una imagen permiten detectar *textones*.

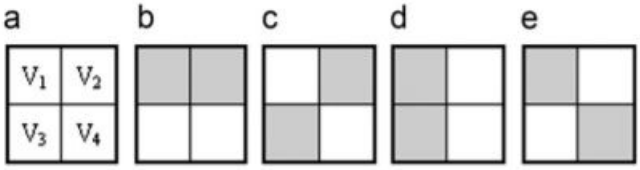

Figura 9: Plantillas para obtener textones [40].

Básicamente, el proceso de detección de *textones* consiste en pasar las plantillas de izquierda a derecha y de arriba abajo a través de la imagen. Si se detecta un *texton* los valores de los píxeles detectados en las plantillas se mantienen sin cambiar, de lo contrario se pondrá a valor cero. Finalmente, se obtiene una nueva imagen. A continuación, se ilustra el proceso:

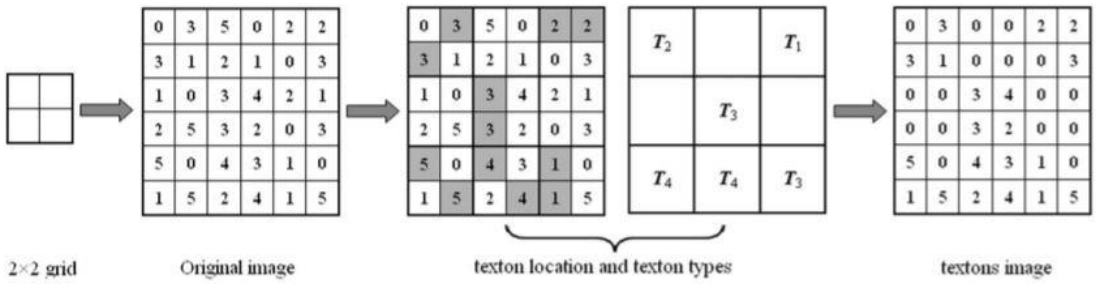

Figura 10: Obtención de imagen de textones [40].

Este enfoque se basa en [41] haciendo hincapié en las distancias entre los elementos de la textura para poder computar los *textones*.

En [42] dan un enfoque nuevo para computar los *textones* siguiendo la definición original [43] y añaden el color como un *texton* más. También aquí se da una buena revisión sobre la teoría de la percepción humana de la textura.

Existen muchos *papers* donde se explora el uso de *textones* en diferentes campos de aplicación. Por ejemplo, en [44] podemos ver un ejemplo del uso de *textones* para la clasificación de comida, en [45] se usan *textones* para clasificar imágenes satelitales, obtenidas de Google Earth, de cara a estimar la densidad de población y en [46] se usan textones para la clasificación de texturas pulmonares.

2.6 Elección de los conjuntos de entrenamiento y test

Antes de entrar a explicar cómo se lleva a cabo el proceso de entrenamiento y testeo es necesario pararse a indicar la distribución de las imágenes y las texturas.

Por cada textura diferente que se usa en el presente trabajo hay un conjunto de 92 imágenes. Este conjunto queda dividido en 46 imágenes para la parte de entrenamiento y 46 imágenes para la parte de testeo. Estas imágenes son elegidas en cada iteración de forma aleatoria, por tanto, no se especifican previamente de forma que cada conjunto entrenamiento/test tienda a ser diferente en cada iteración del algoritmo dando más consistencia los resultados.

Los experimentos son llevados a cabo por dos conjuntos, a saber:

El primero de ellos, que encontramos en [20], lo forman las 20 texturas de la figura 19.a, de la referencia dada, dándonos un total de 20*92= 1840 imágenes diferentes, de las cuales cada par entrenamiento/test estará formado por 920 imágenes.

El segundo conjunto está formado por las 40 texturas que se especifican en la figura 7 de [22]. Este conjunto nos da un total de 40*92=3680 imágenes diferentes, de las cuales para cada par entrenamiento test tenemos, en esta ocasión, 1840 imágenes diferentes.

Es importante destacar que los modelos que se obtendrán de las imágenes de entrenamiento pueden ser reducidos consiguiendo con ello la reducción de carga computacional, de hecho, es objeto de estudio en [20]. Sin embargo, en este trabajo no se estudia la reducción de los modelos. En cierto modo esto podría ser una línea de investigación futura.

2.7 Fase de entrenamiento

2.7.1 Generación del diccionario de textones

Como se comenta en la introducción del presente capítulo la parte de entrenamiento del algoritmo tiene dos partes diferenciadas; primeramente, se obtiene un diccionario de *textones* que será usado para el resto de los experimentos. Seguidamente a ello se generan los modelos.

En la primera parte de aprendizaje, las imágenes de entrenamiento de una determinada textura son filtradas por el banco de filtros ICA para obtener las respuestas a los filtros. Estas respuestas se agregan y agrupan en los *textones*. Para ello se usa un algoritmo de agrupamiento como *K-Means* [21] (en el anexo se da una pequeña introducción al algoritmo). Los *textones* obtenidos de diferentes clases de texturas son unidos para forma un diccionario de *textones*.

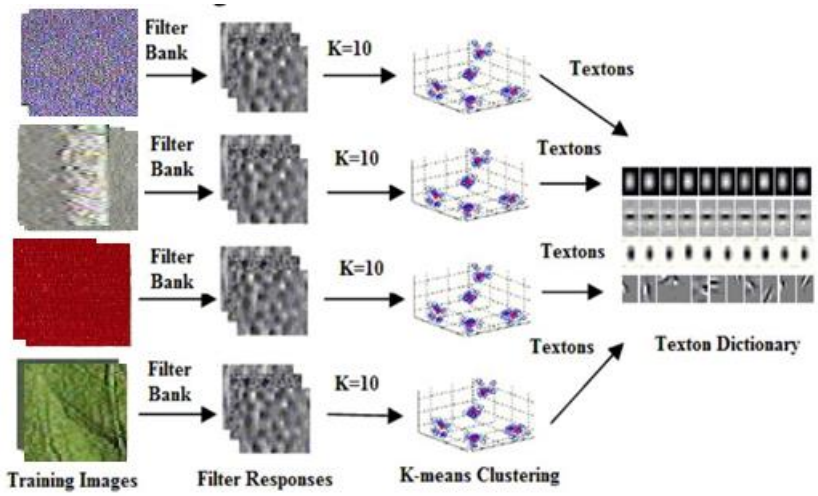

Figura 11: Etapa de aprendizaje I: Generación del diccionario de textones [48].

Para el cálculo del diccionario de *textones* se seleccionan 13 imágenes aleatorias elegidas entre 20 texturas que son las especificadas en la figura 7 de [22]. Tras eso se filtran con el banco de filtros *ICA* y se agregan las respuestas que producen el banco de filtros a cada una de las 13 imágenes de la misma textura. Por último, se calculan 10 *textones*, por cada textura, usando el algoritmo *K-Means* para después juntar todos los *textones* formando un único diccionario. A modo de ejemplo se puede indicar que si tenemos 20 clases diferentes de texturas tendremos un diccionario formado por 200 *textones*.

Cabe destacar que previamente al cálculo del diccionario de *textones* hemos obtenido, como se indica en 2.4, el banco de filtros ICA y este se ha obtenido a partir de las texturas con las que se generan el diccionario de *textones*. Sin embargo, en lugar de usar solamente 13 imágenes aleatorias por clase, se usan 46 imágenes por cada una de las clases que, como veremos, coincide con el número de imágenes de entrenamiento usadas por textura.

2.7.2 Generación de modelos.

Para la generación de modelos se usará un conjunto de 46 imágenes por textura como se indica en 2.6.

Dada una imagen de entrenamiento, se consigue obtener su modelo filtrando la imagen con el banco de filtros ICA y, después, buscando qué *texton* es el más cercado a cada respuesta al filtrado.

Un histograma nos dará un modelo para cada una de las imágenes de entrenamiento tendremos, por tanto, una serie de modelos por clase. Una vez se filtra una imagen por el banco de filtros *ICA*, se obtiene un conjunto de respuestas (una respuesta por cada uno de los filtros que forman el banco de filtros *ICA*). Cada píxel (y,x) de

estas respuestas al banco de filtros tendrá un total de z respuestas diferentes coincidentes con el número filtros que forman el banco de filtros *ICA*, obteniéndose $x * y$ vectores de características con z valores para cada uno de los pixeles. Estas respuestas formarán una matriz de características de la forma que se puede apreciar en la figura 6.

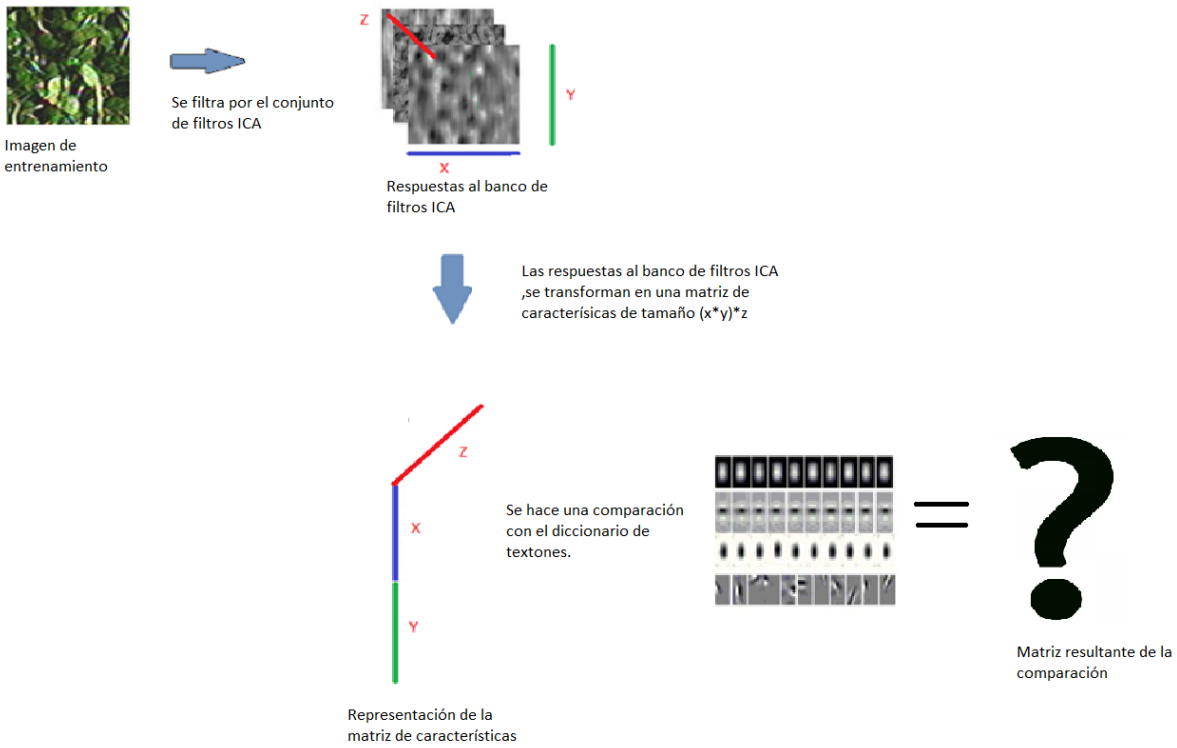

Figura 12: Etapa de aprendizaje. Conversión de respuesta a filtrado en una matriz.

En 2.7.1 se indica que se tendrán 10 *textones* por cada una de las texturas y que el diccionario de *textones* quedará formado por 20 texturas diferentes. En definitiva, se trabajará con un diccionario de *textones* de tamaño 200*número de bases dominantes o, expresado de otra manera, 200 por el número de filtros que forman el banco de filtros *ICA* (cabe recordar que el banco de filtros *ICA* se obtiene a partir de la matriz de mezcla, A, cuyo número de columnas está establecido por el número de bases dominantes).

Partiendo de la generación del diccionario de *textones*, el proceso para la creación del mapa de *textones* (que nos dará el modelo asociado a la imagen de entrenamiento [ver figura 6]); consistirá en la comparación de las respuestas obtenidas al filtrado de una imagen por el banco de filtros *ICA* con el diccionario de *textones*. Para ello se aplica la norma Euclídea que, básicamente, es una comparación entre puntos. Llamando z, al número de filtros que forman el banco de filtros ICA, tendremos un diccionario de *textones* de tamaño 200*z y una matriz de características de tamaño $(x * y) * z$ [ver figura 6]. El mapa de *textones* para una imagen, I, se genera comparando cada una de las filas del diccionario de *textones* con cada fila de la matriz de características y obteniendo el índice de menor valor en cada una de las columnas resultantes de dicha comparación. Cada fila del diccionario de *textones* se compara con cada una de las filas de la matriz de características. A modo de ejemplo si tuviésemos un diccionario de *textones* de tamaño 200x40 y una matriz de características de tamaño 1700x40, la matriz resultante de la comparación se obtendría seleccionando cada una de las filas del diccionario de *textones* (1x40) y comparándolas con cada una de las filas de la matriz características (1x40). En cada iteración se compara una fila del diccionario de *textones*(1x40) con en este caso 1700 filas de (1x40) para generar una fila de 1x1700 (en la fórmula 2.15 se aprecia la distancia que se aplica en la comparación entre dos puntos cualesquiera de tamaño m-dimensional —para mayor eficiencia computacional y sabiendo que el fin de la distancia es solo para fines comparativos; se puede prescindir de aplicar la raíz cuadrada tras la comparación—).

El resultado final será una matriz de tamaño, 200x1700, en definitiva, una comparación de cada píxel, de la imagen filtrada por el banco de filtros *ICA*, a cada uno de los 200 *textones*.

$$distancia\ euclídea = \sqrt[2]{\sum_{i=i}^{n}(p_i - q_i)^2} \qquad (2.15)$$

Tras eso es necesario, como previamente se indica, obtener el índice del menor valor en cada una de las columnas de la matriz resultante del ejemplo anterior (matriz de tamaño 200x1700). Este índice nos indicará que *texton* de los 200 es el más cercano a la respuesta a cada uno de los píxeles de la imagen. Una vez que se obtiene este índice el vector se reordenará para formar una matriz que será el mapa de *textones*. Para ilustrar esta idea vamos a obtener este índice mirando, por ejemplo, la matriz de la figura 7 donde diríamos que en la primera columna el menor valor es el uno y su índice es el cuatro; en la segunda columna el menor valor es el cinco y su índice sería también el cuatro; en la tercera columna el menor valor sería el cero y su índice sería el 3 y en en la cuarta, y última columna, el menor valor sería el uno y su índice el 1. Los índices nos indican a qué *texton* pertenece cada una de las columnas y buscaremos "contar" cuantas veces aparece cada *texton* mediante el uso de un histograma.

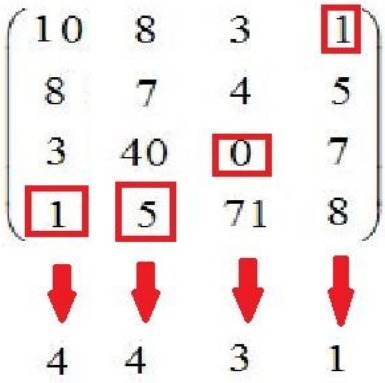

Figura 13: Matriz usada como ejemplo para demostrar cómo obtener el texton más cercano.

Esto creará un modelo para cada imagen y en la parte de test nuestro objetivo será poder encontrar el modelo de entrenamiento que más se parezca a al modelo la imagen de test que queramos clasificar. Para cada clase tendremos varios modelos obtenidos para cada una de las imágenes [ver figura 8].

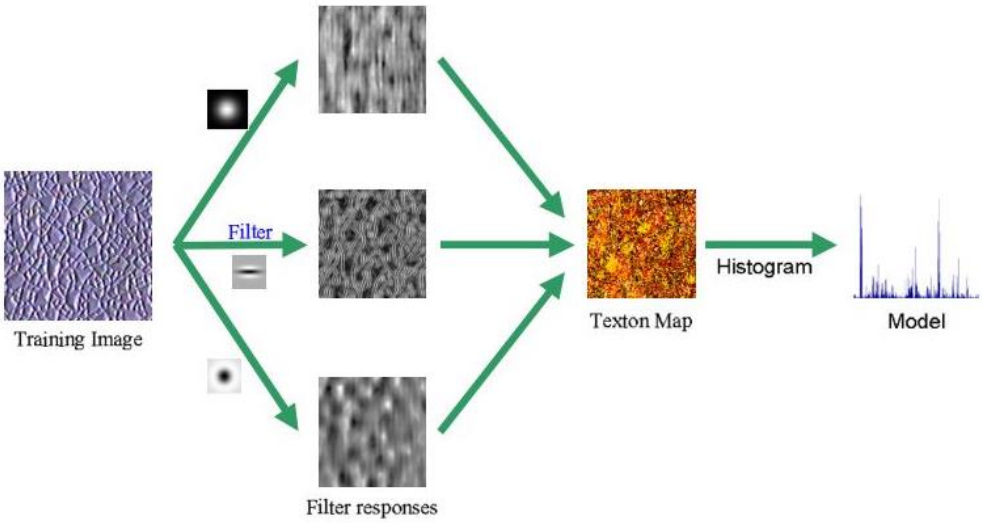

Figura 14: Etapa de aprendizaje II: Generación de modelos [20].

2.8 Fase de clasificación.

Como se comenta en 2.6 el conjunto formado por las imágenes de prueba será igual, en tamaño, al grupo formado por las imágenes de entrenamiento, En concreto por cada una de las texturas independientemente de las texturas usadas para los dos experimentos, se usan 46 imágenes por texturas.

El proceso consiste en volver a ordenar aleatoriamente las imágenes que se disponen de test, para después filtrar cada una de ellas por el banco de filtros ICA que, previamente, fue obtenido en el segundo paso del algoritmo de aprendizaje, es decir, creado a partir de las imágenes de entrenamiento. Tras el filtrado de cada imagen se crea su modelo, tal y como se detalla en 2.7.2, y se compara con los modelos obtenidos en la fase de entrenamiento mediante el algoritmo kNN (k-Nearest-Neighbors), y usando la distancia chi-cuadrado, χ^2. La imagen de test pertenecerá a la clase a la que pertenezca el texton más cercano [ver figura 9]. El lector interesado podrá ver una breve explicación sobre el algoritmo kNN en el anexo.

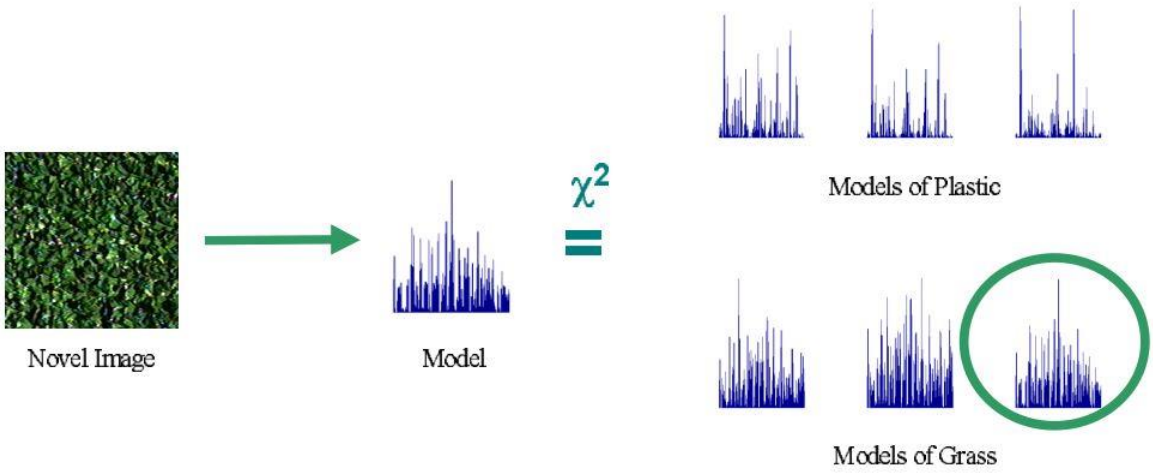

Figura 15: Etapa de aprendizaje II: Generación de modelos [20].

3 EXPERIMENTOS Y RESULTADOS

"In life, unlike chess, the game continues after checkmate"

- Isaac Asimov-

En este capítulo se presentan los resultados que se han ido obteniendo usando el clasificador. Básicamente, una vez que se construye todo el código, el objetivo es jugar con una serie de parámetros que nos darán diferentes bancos de filtros *ICA* y que se traducirán en resultados distintos en la clasificación. Los dos parámetros más importantes son el tamaño del *patch* y el número de bases dominantes.

Con la elección del tamaño del *patch*, cada una de las imágenes de tamaño 200x200 pixeles será dividida en *patches* iguales del tamaño del *patch* que elijamos. Esto condicionará el tamaño de la matriz de observaciones que se le pasará al algoritmo *FastICA*. Esta matriz estará formada por, en el caso del diccionario de *textones* y como previamente se indica, 13 imágenes por clase de 20 clases diferentes. Estas imágenes previamente se someten a una partición en *patches* que se distribuyen por columnas de forma aleatoria [[ver Apartado 2.3].

Tras obtener este diccionario de *textones* es el turno de las imágenes usadas para el entrenamiento y test. Se indica en el punto anterior la formas de obtenerlas, pero aquí subrayo que como característica importante la matriz que se la pasa al algoritmo *FastICA* también dependerá de la elección del tamaño del *patch*. Justamente como el cálculo del diccionario de *textones*. El *patch* nos dará el número de filas e influirá en el número de columnas. Pero no todas las columnas nos aportan información, o quizás y mejor expresado, no siempre son necesarias todas para obtener información. Por ello el otro parámetro importante es el número de bases dominantes que se pasa al algoritmo FastICA. Gracias a este parámetro conseguiremos una matriz de tamaño más reducido pero que presente la mayor parte de la información, sin ser tan amplia y contribuyendo, significativamente, a la mejora del coste computacional.

Por motivos de eficiencia computacional el número de texturas con los cuales se hacen las pruebas corresponden a los detallados en el artículo de Varma y Zisserman [20] para los casos de 20 y 40 texturas.

En la siguiente tabla se comparan los resultados obtenidos con nuestro método y los incluidos en el artículo de Varma y Zisserman:

	Número de texturas	
Banco de filtros	20	40
S	96.30%	95.27%
LM	96.08%	93.75%
MR4	94.13%	92.07%
MR8	97.83%	96.41%
ICA	98,04%	97,14%

Tabla 1. ***Comparativa de banco de filtros ICA con los bancos de filtros S, LM, MR4 y MR8.***

Desarrollando los porcentajes obtenidos para 20 texturas diferentes en función del tamaño del patch y el número de bases dominantes obtenemos:

	Número de bases dominantes	
Tamaño de los patches	16	32
8	96,95%	97,17%
10	90,97%	94,13%
12	97,93%	97,50%
14	95.86%	96,19%
16	97.71%	98,04%

Tabla 2 Distintos resultados para 20 texturas diferentes del banco de filtros ICA variando el número de patches y el número de bases.

Para el caso de 40 texturas diferentes:

	Número de bases dominantes	
Tamaño de los patches	16	32
8	70.43%	74.29%
10	92.17%	94,13%
12	92,41%	92,01%
14	95.86%	96,08%
16	95,94%	97,14%

Tabla 3 Distintos resultados para 40 texturas diferentes del banco de filtros ICA variando el número de patches y el número de bases.

A continuación, se representarán los bancos de filtros *ICA* obtenidos para las diferentes texturas usadas en la obtención del diccionario de *textones*. Como se ha comentado en la parte teórica de este trabajo, los bancos de filtros ICA son dependientes de las imágenes con las que se trabajan y también para obtener el diccionario de *textones*. Por ello, para el caso de las texturas con las que se calcula el diccionario de *textones*, se obtienen una serie de banco de filtros los cuales se representan para diferentes tamaños de los *patches* y para diferente número de bases dominantes. Se recuerda al lector que las texturas usadas para obtener los *textones* son las especificadas en [6]. Hay que añadir que también se hace un análisis en frecuencias de los bancos de filtros. Para ello se hace uso de la transformada FFT-2D (2-D *Fast Fourier transform*). Se puede apreciar cómo se indica en [23] la similitud con los filtros de Gabor. Este hecho se debe al usar como filtros las columnas de la matriz *A* como se indicó previamente.

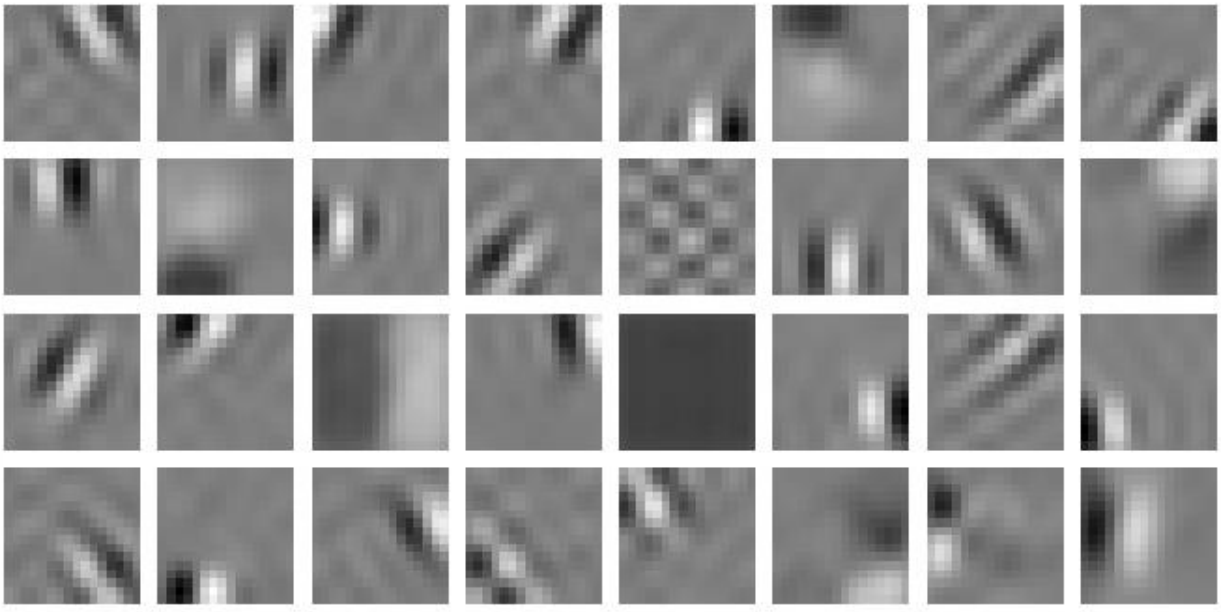

Figura 16. Representación de filtros ICA en el dominio del espacio. Tamaño del patch usado 16x16. Número de bases dominantes 32.

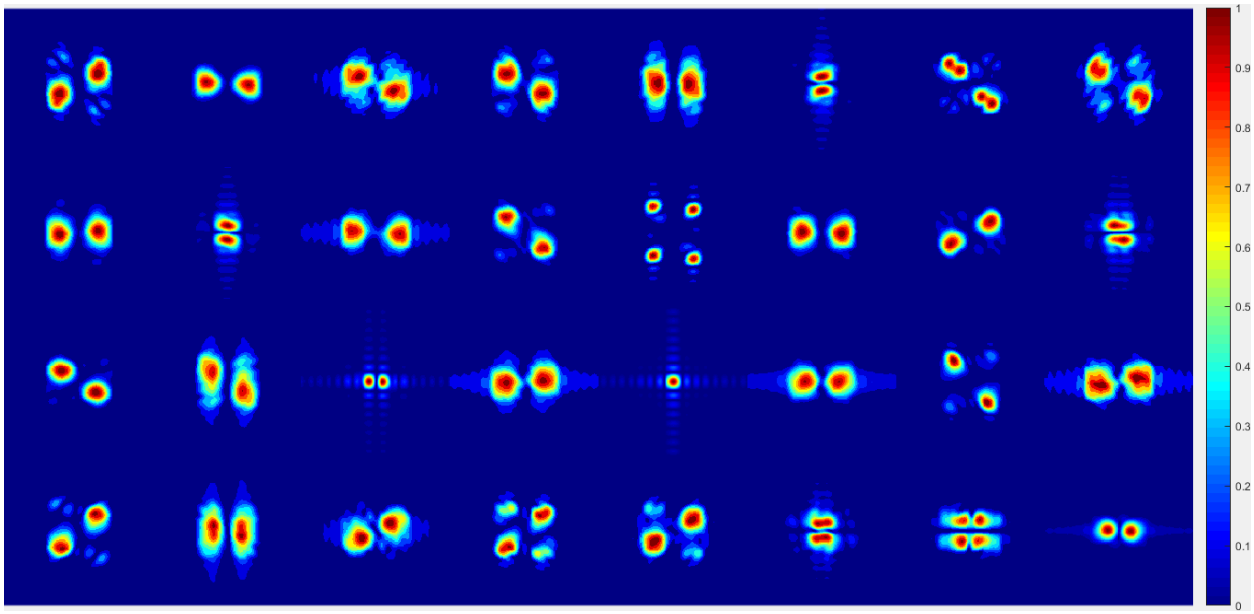

Figura 17. Representación frecuencial de los filtros ICA. Tamaño de los patch usado 16x16. Número de bases dominantes 32.

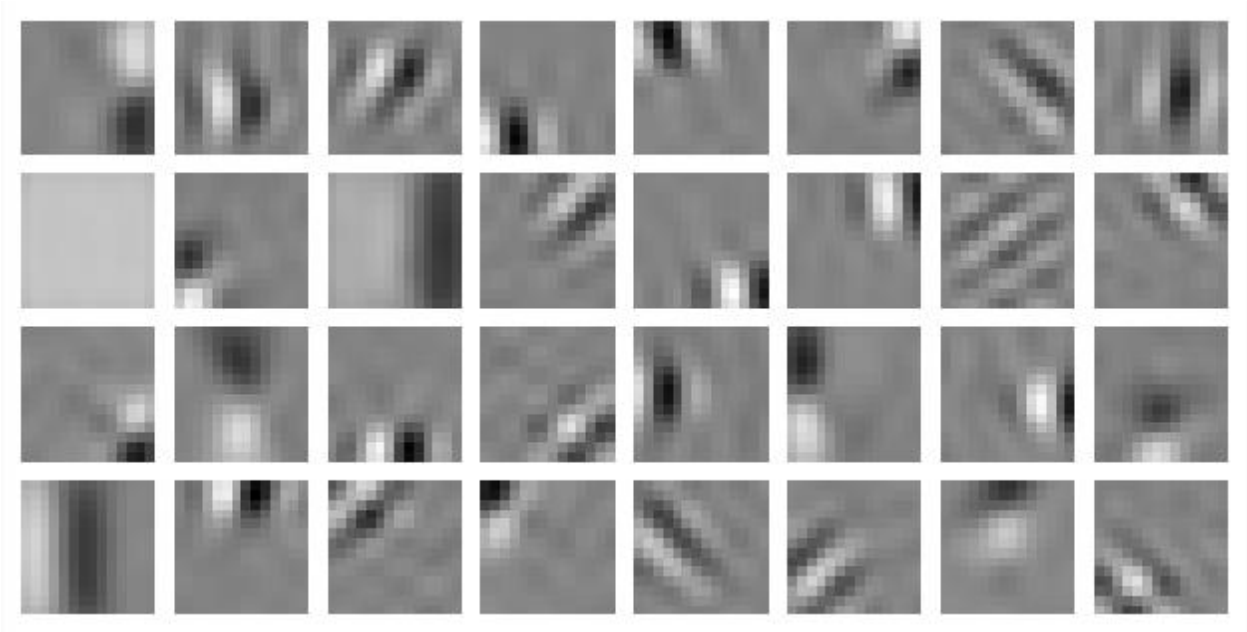

Figura 18. Representación de filtros ICA en el dominio del espacio. Tamaño del patch usado 14x14. Número de bases dominantes 32.

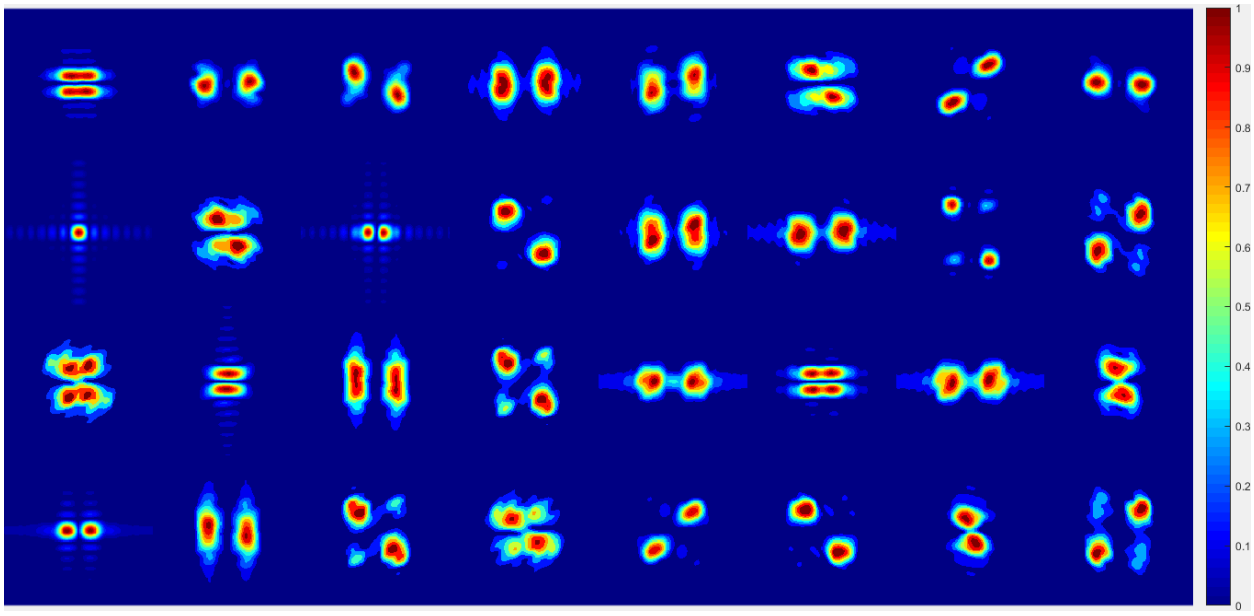

Figura 19. Representación frecuencial de los filtros ICA. Tamaño de los patch usado 14x14. Número de bases dominantes 32.

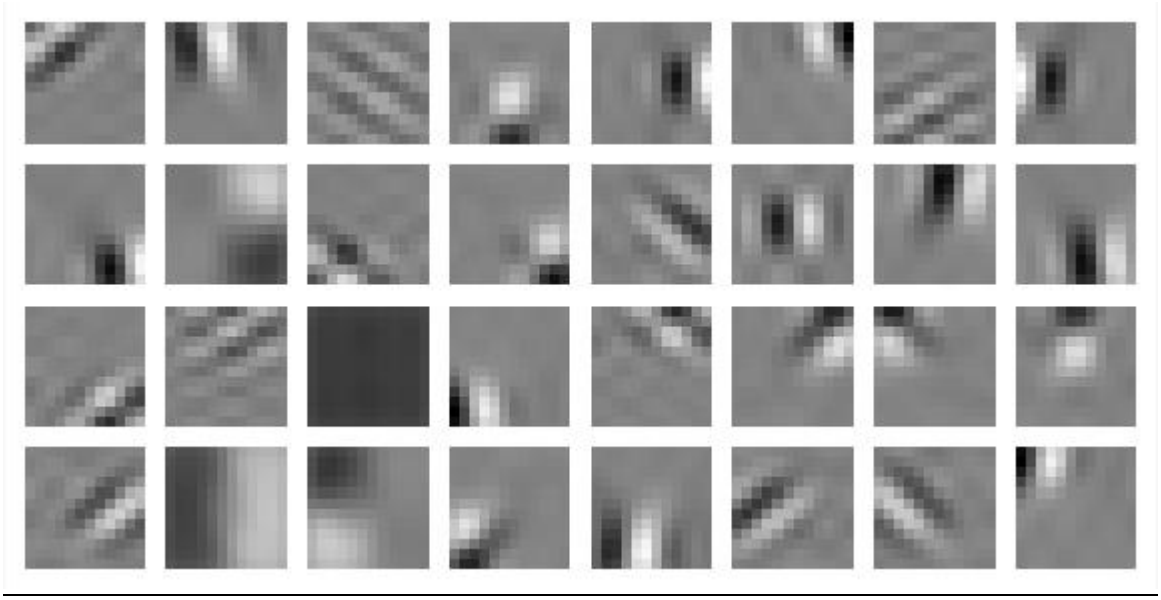

Figura 20. Representación de filtros ICA en el dominio del espacio. Tamaño del patch usado 12x12. Número de bases dominantes 32.

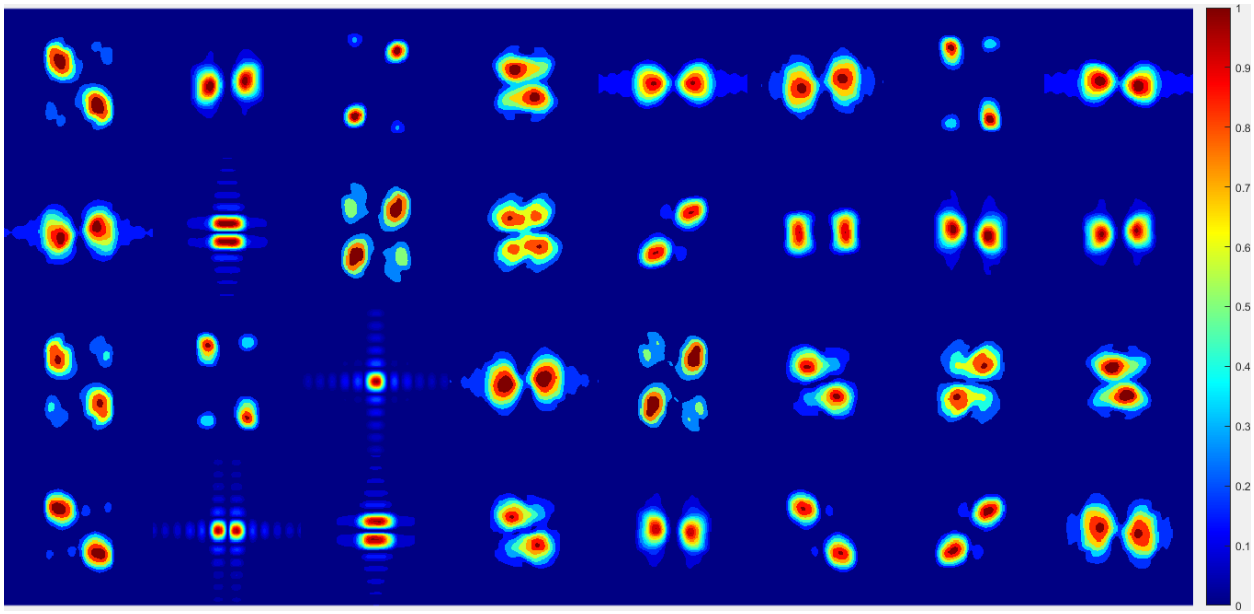

Figura 21. Representación frecuencial de los filtros ICA. Tamaño de los patch usado 12x12. Número de bases dominantes 32.

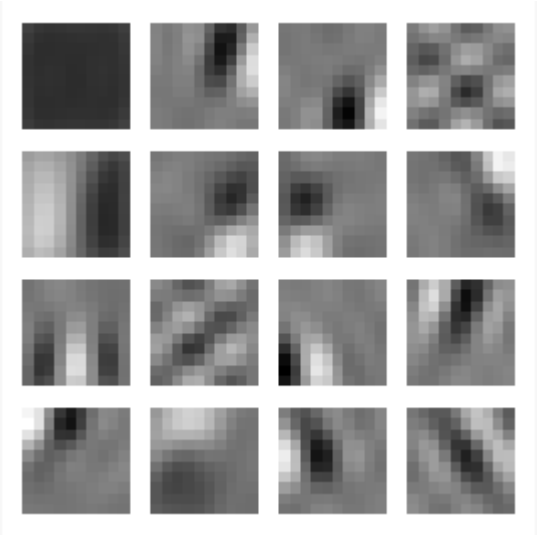

Figura 22. Representación de filtros ICA en el dominio del espacio. Tamaño del patch usado 10x10. Número de bases dominantes 16.

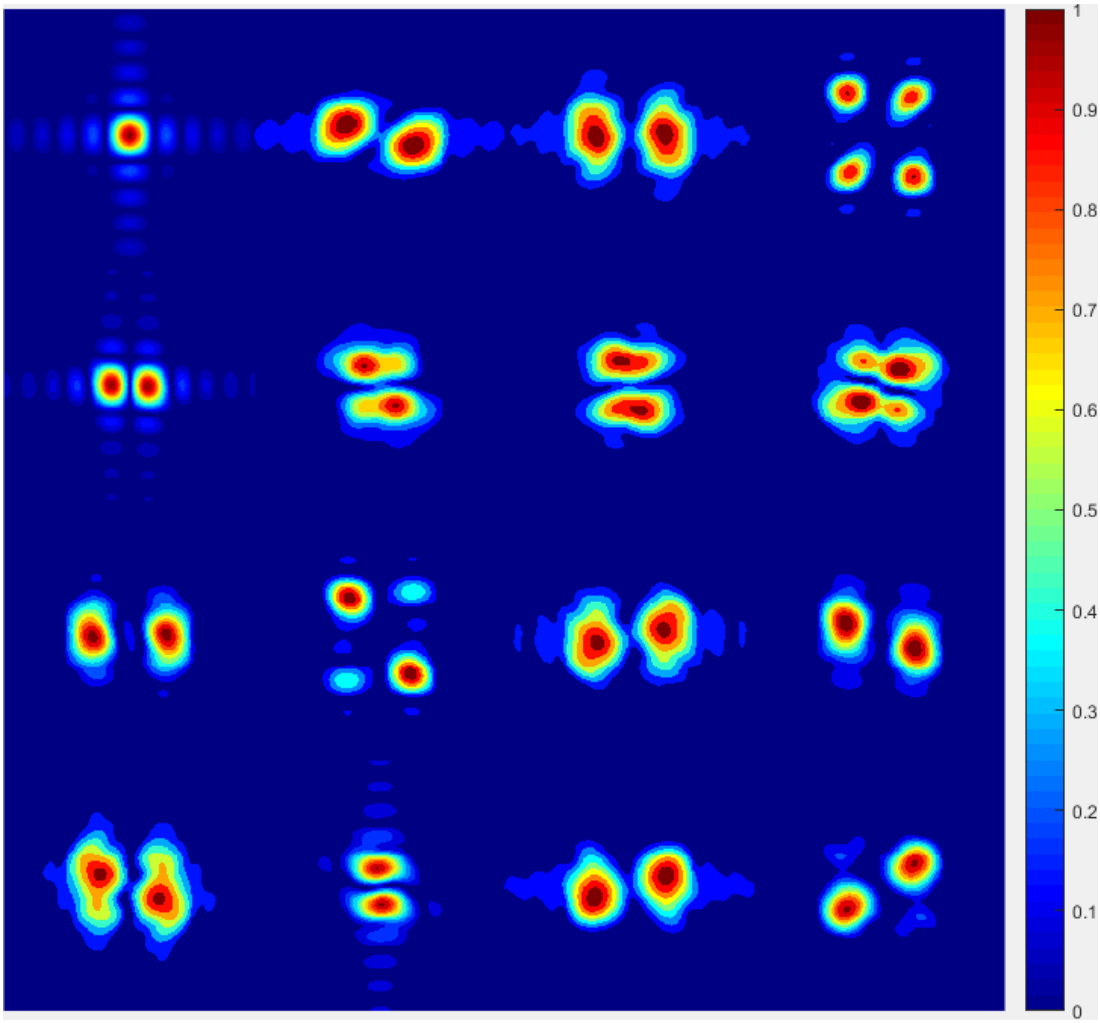

Figura 23. Representación frecuencial de los filtros ICA Tamaño del patch usado 10x10. Número de bases dominantes 16.

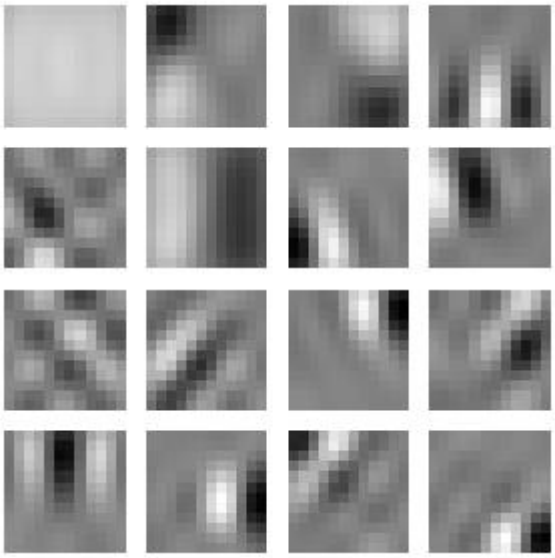

Figura 24. Representación de filtros ICA en el dominio del espacio. Tamaño del patch usado 12x12. Número de bases dominantes 16.

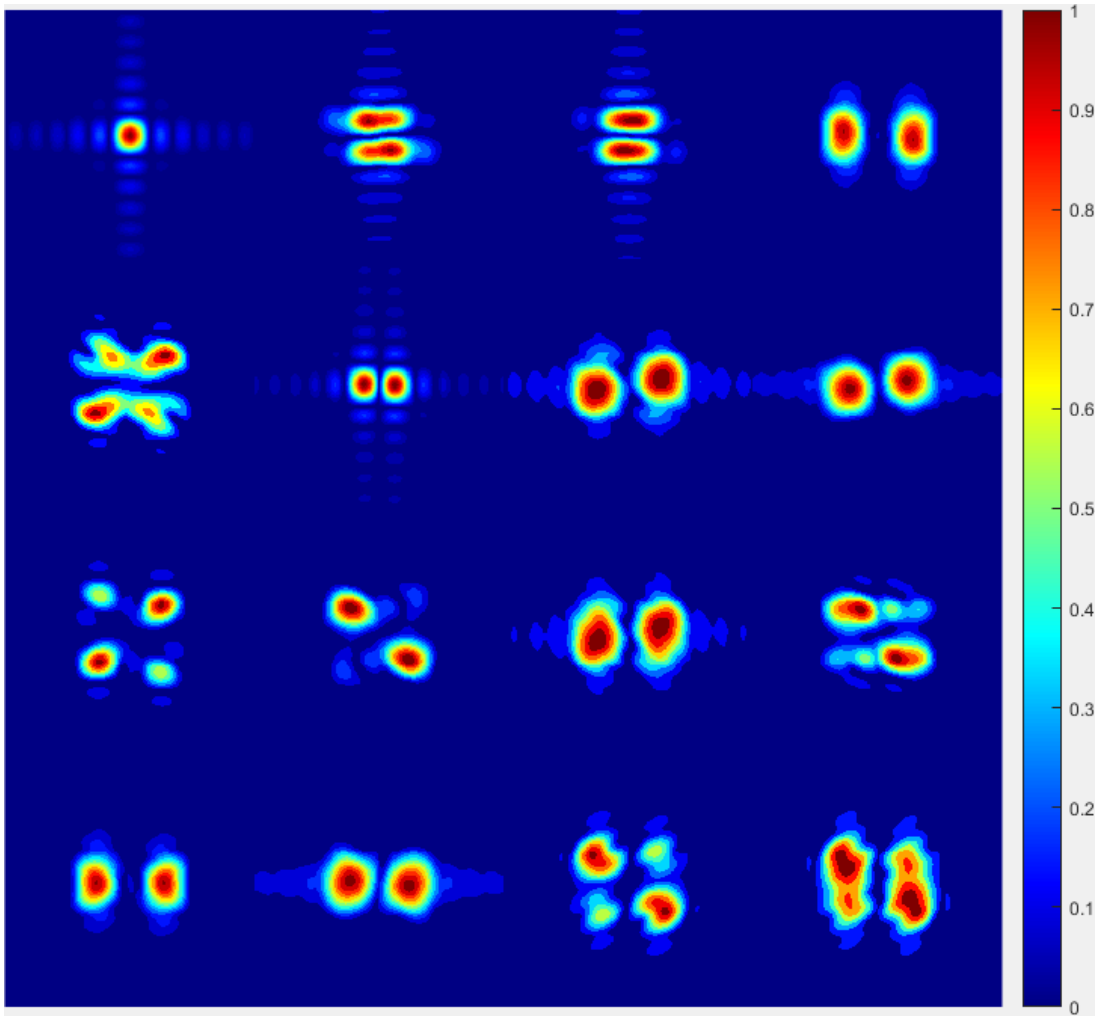

Figura 25. Representación frecuencial de los filtros ICA. Tamaño del patch usado 12x12. Número de bases dominantes 16.

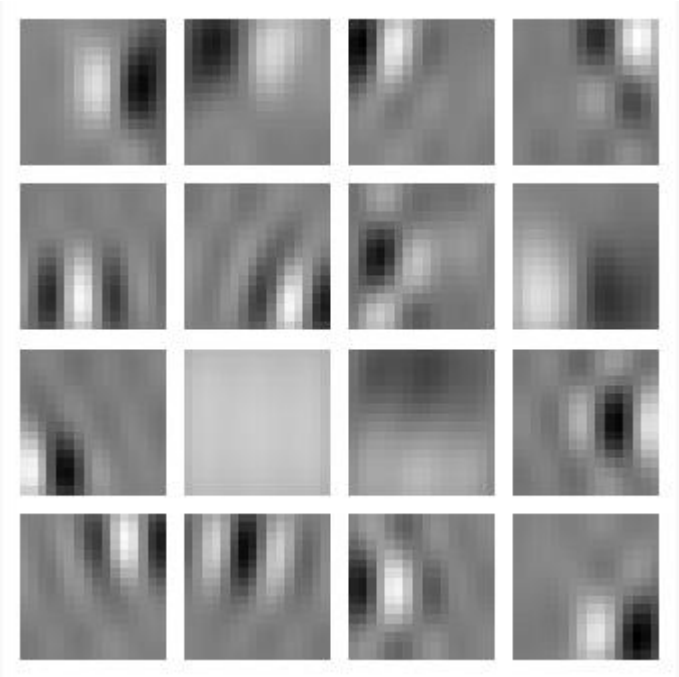

Figura 26. Representación de filtros ICA en el dominio del espacio. Tamaño del patch usado 16x16. Número de bases dominantes 16.

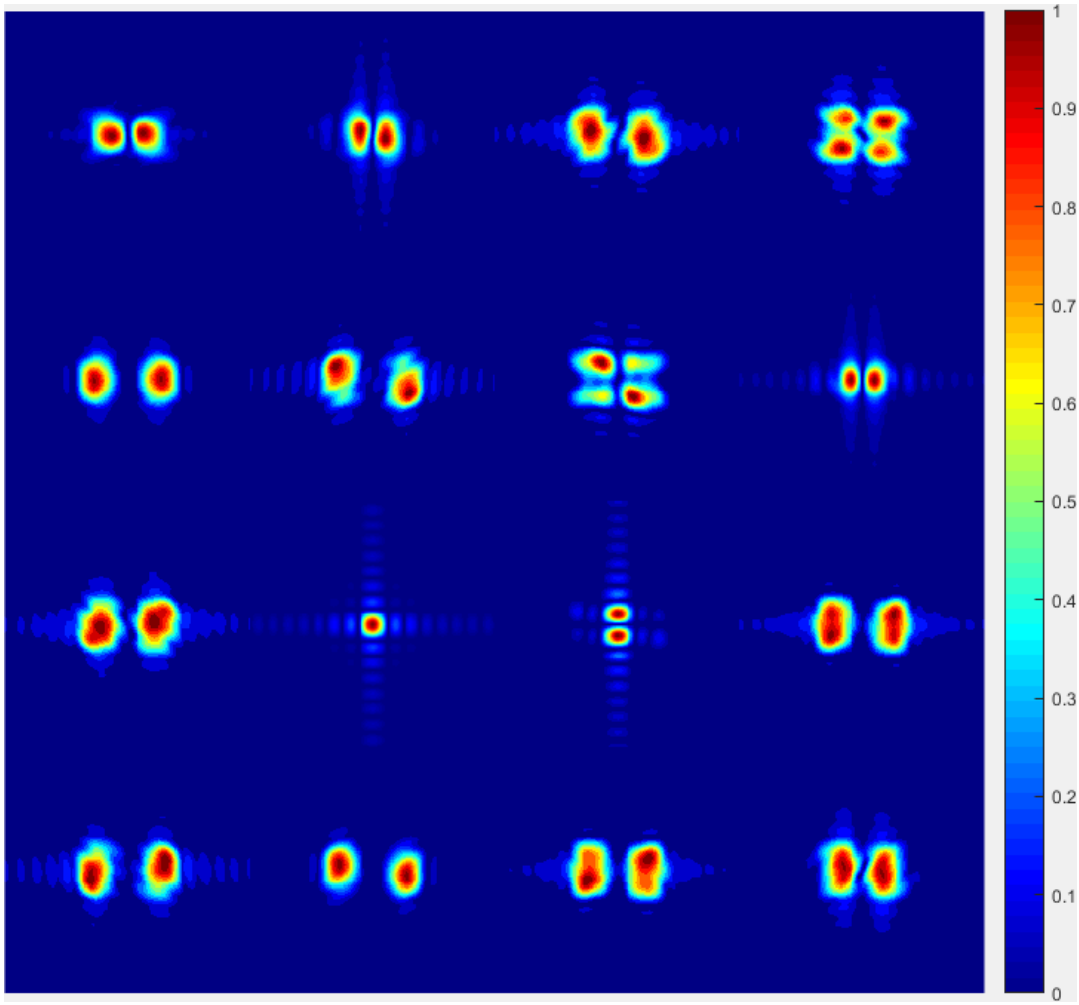

Figura 27. Representación frecuencial de los filtros ICA. Tamaño del patch usado 16x16. Número de bases dominantes 16.

Una vez que se han mostrado varios ejemplos de los filtros ICA, tanto en el dominio espacial como en el dominio frecuencial, se mostrarán diversos ejemplos del filtrado de una serie de imágenes de diferentes clases mediante el banco de filtros *ICA*. Se usarán 32 bases dominantes y un tamaño del *patch* de 14x14 para los filtros.

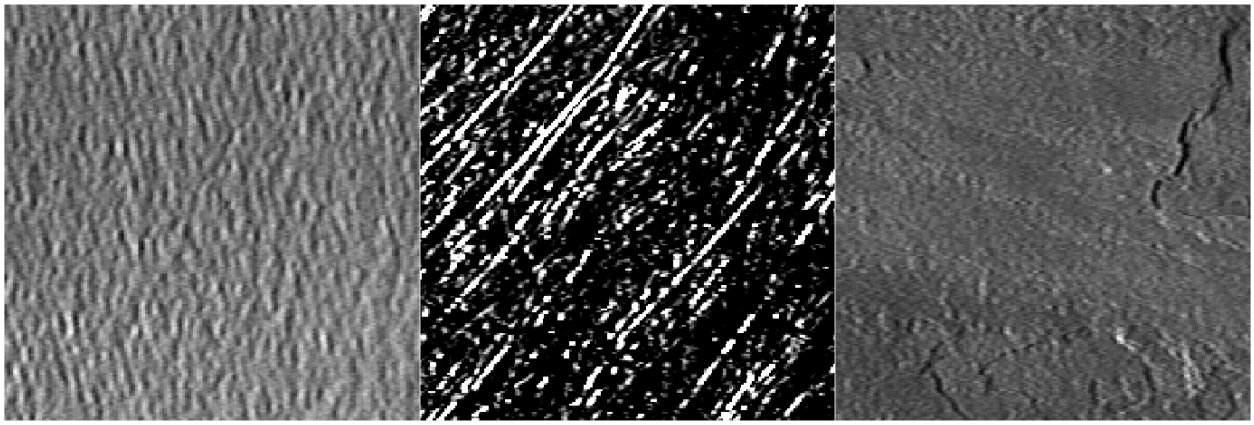

Figura 28: Imágenes de la base de datos CUReT. Las clases de CUReT son la 12, 27 y 34, respectivamente.

En las siguientes imágenes podemos ver cada una de las imágenes de la figura 28 filtradas por los bancos de filtros ICA:

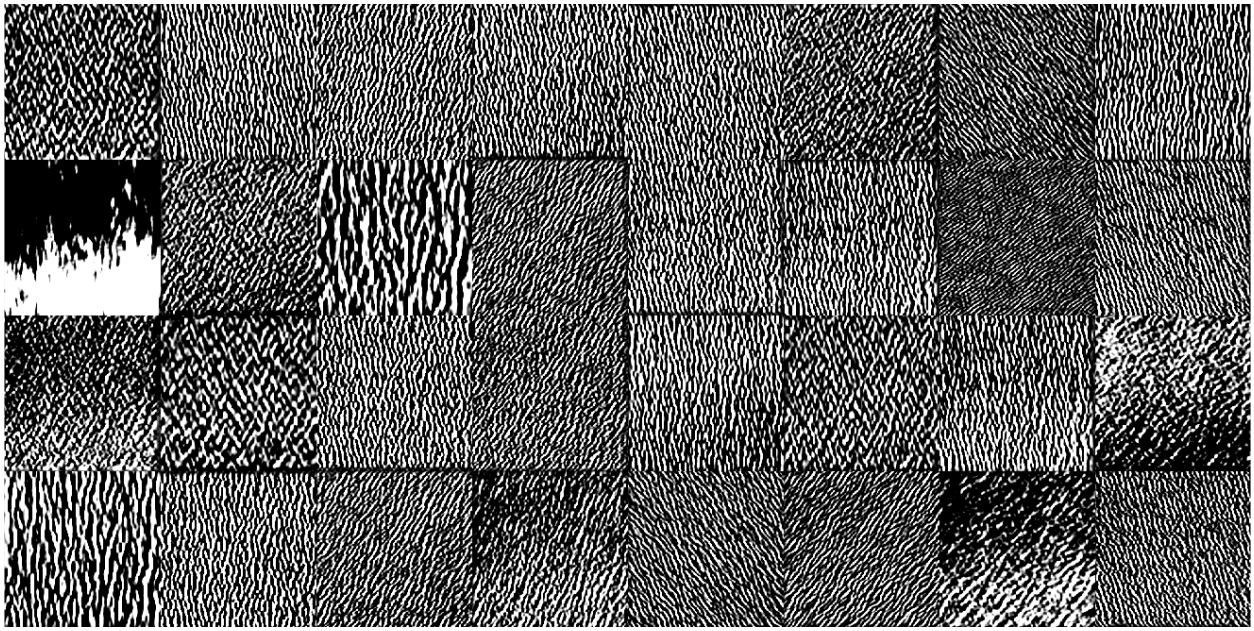

Figura 29: Imagen de la clase 12 de la base de datos CUReT filtrada por un banco de 32 filtros.

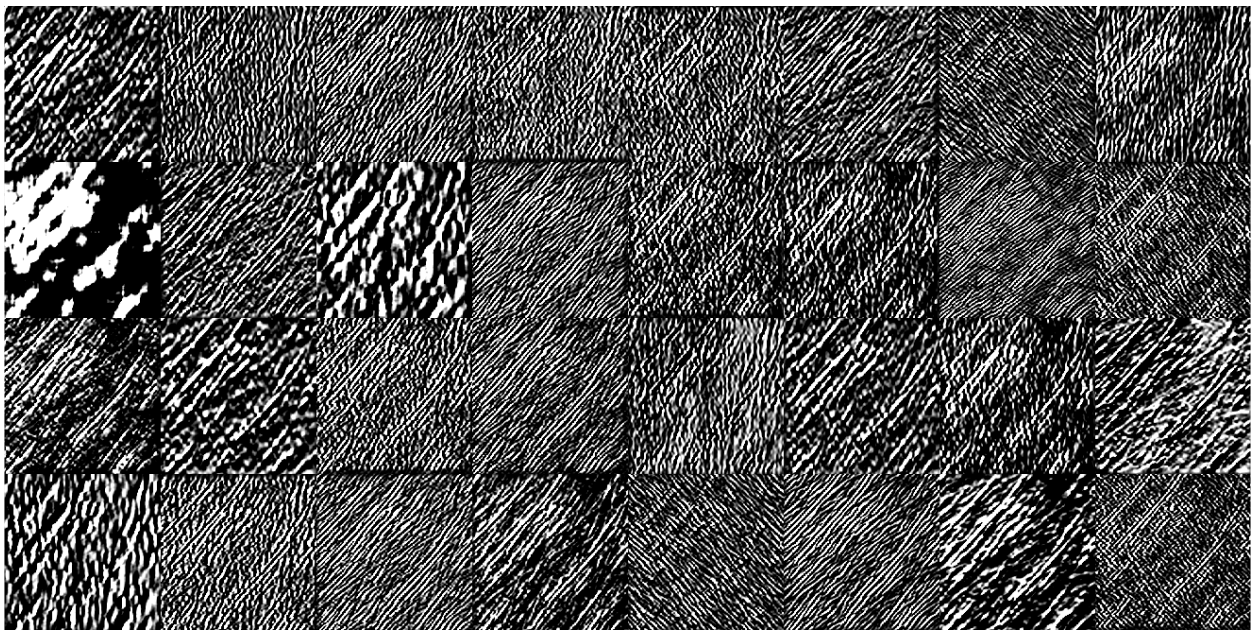

Figura 30: Imagen de la clase 27 de la base de datos CUReT filtrada por un banco de 32 filtros.

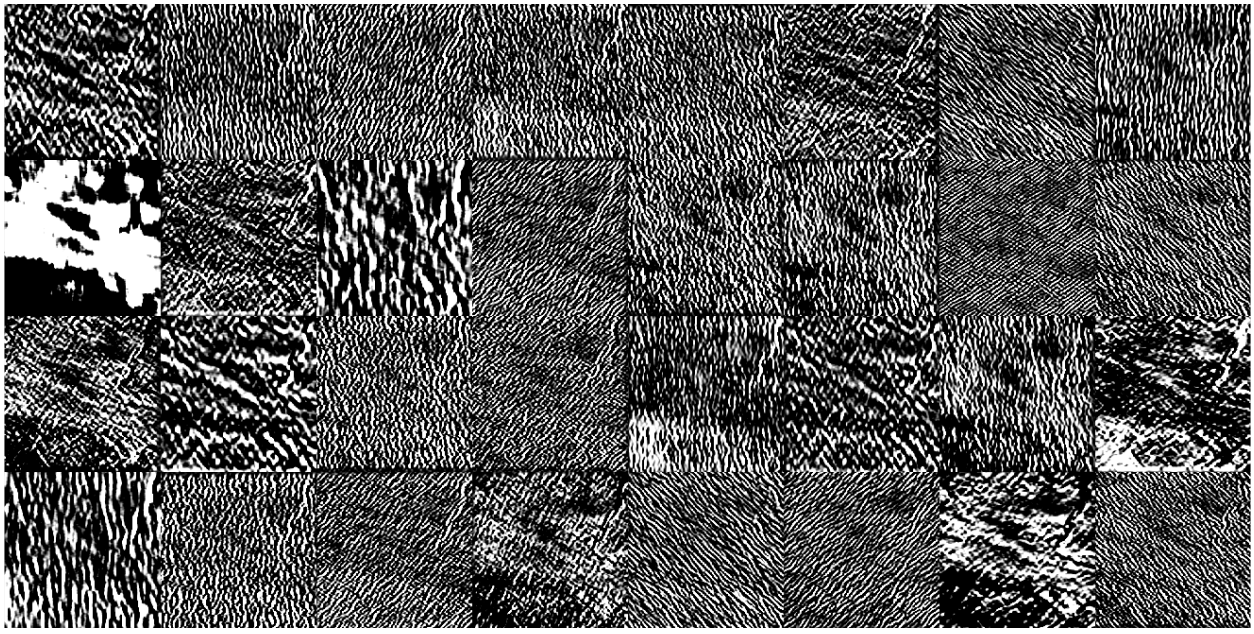

Figura 31: Imagen de la clase 34 de la base de datos CUReT filtrada por un banco de 32 filtros.

Como nota curiosa la imagen 9 parece que da un resultado extremo respecto al resto de imágenes. Esto es debido al filtro empleado (el filtro 9 de los 32 con *patches* de tamaño 14 — en esa iteración del algoritmo da un resultado extemo —). A continuación, se representa este filtro en el dominio frecuencial en 3D. Se puede obsevar como el valor máximo del filtro es bastante alto lo cual también se puede intuir, previamente, en el dominio espacial del mismo.

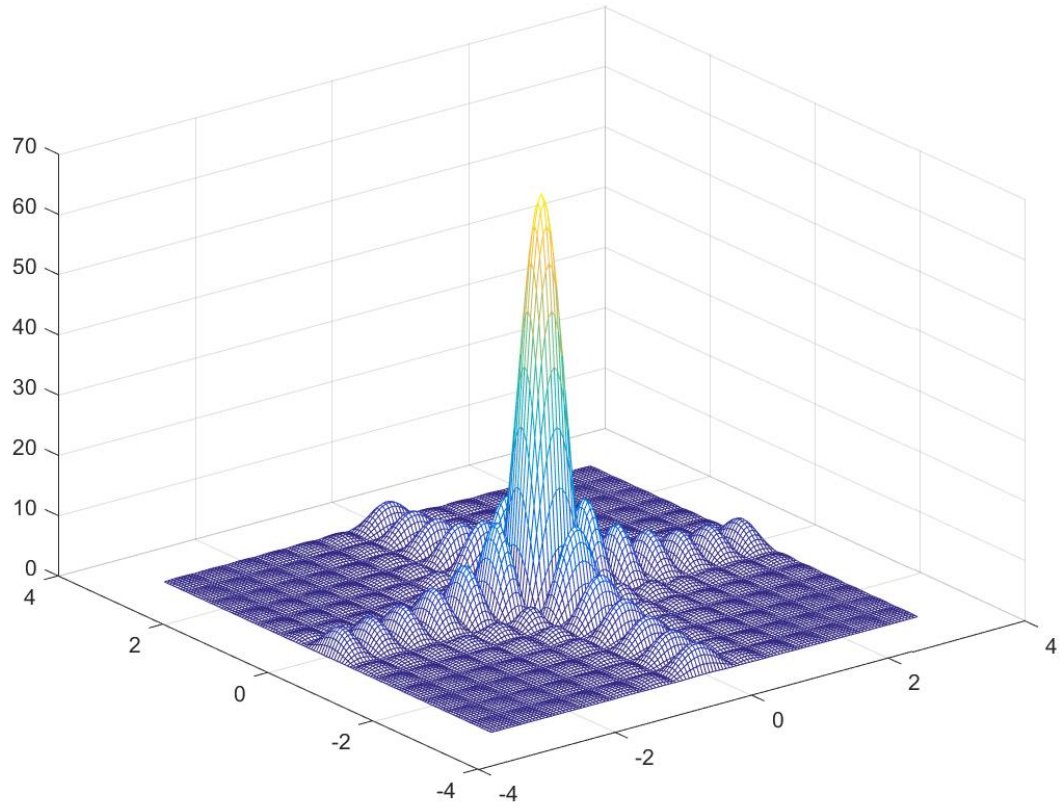

Figura 32: Filtro, representado en 3D, que corresponde al filtro 9 de la figura 19.

Una vez mostradas las imágenes filtradas se reproducen los mapas de *textones* correspondientes a una imagen de entrenamiento de las clases 12, 27 y 34 de la base de datos *CUReT*.

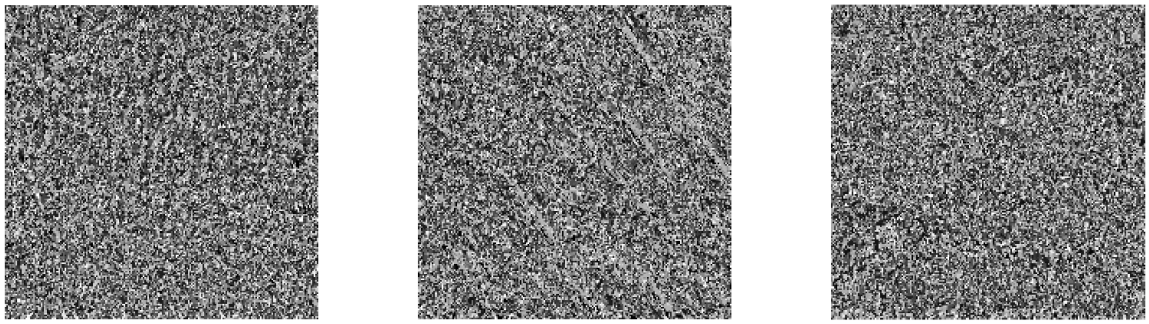

Figura 33: Mapa de Textones obtenidos de las imágenes de la figura 28.

Desde el mapa de textones podemos obtener los correspondientes modelos que usaremos en la comparación para poder clasificar las imágenes de test. Para ello aplicaremos el algoritmo *k-Nearest Neighbors(knn)* que para una imagen de entrenamiento dada y obtenido su correspondiente modelo, como los que se reproducen en la siguiente figura, nos indicará a cuál de todos los modelos que tenemos (46 modelos por clase de textura) se acerca más en parecido y, por tanto, a qué clase pertenece.

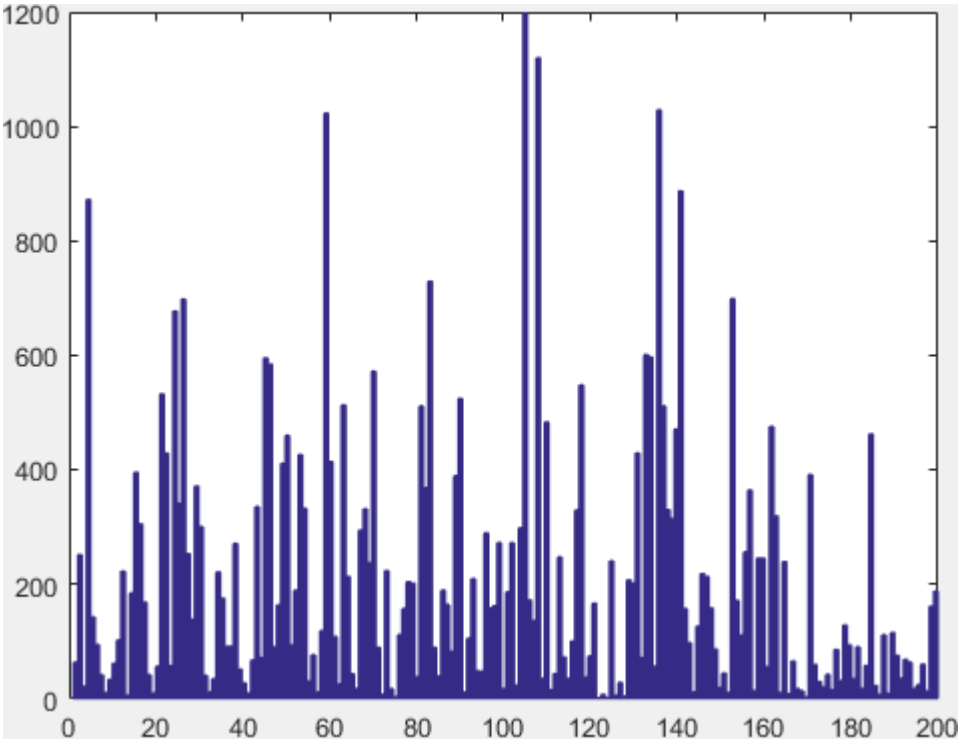

Figura 34: Modelo correspondiente a una imagen de entrenamiento de la clase 12.

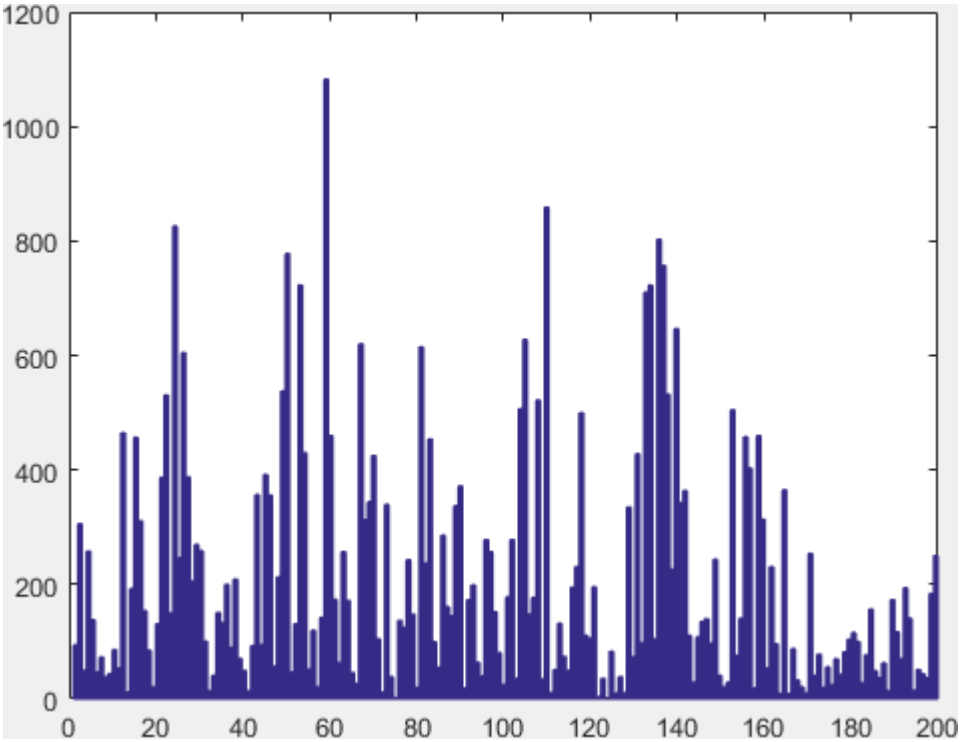

Figura 35: Modelo correspondiente a una imagen de entrenamiento de la clase 27.

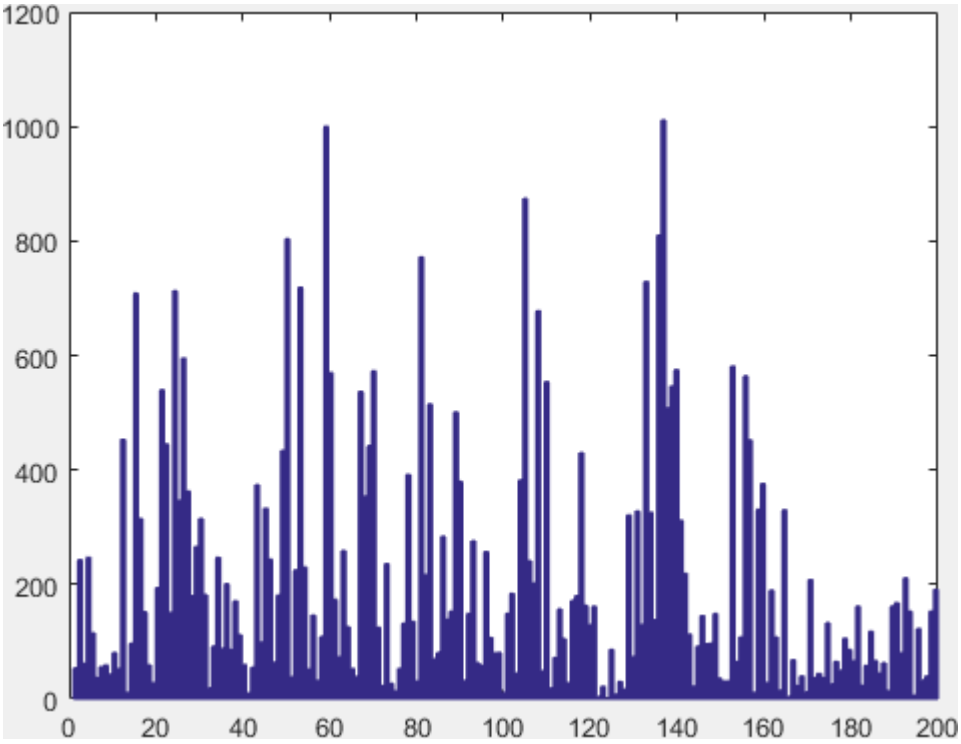

Figura 36: Modelo correspondiente a una imagen de entrenamiento de la clase 34.

4. CONCLUSIONES Y LÍNEAS FUTURAS DE INVESTIGACIÓN.

"Las ideas no duran mucho. Hay que hacer algo con ellas"

- Santiago Ramón y Cajal-

Como se ha podido ver la idea de usar los filtros *ICA* como filtros, que están muy cercanos a las características de las imágenes porque, básicamente, surgen de ellos, parece una buena idea al juzgar los mejores porcentajes conseguidos en el mejor de los casos, y para una base de datos amplia en imágenes. La comparativa que nos propusimos, al inicio del presente trabajo, ha sido satisfactoria con este nuevo enfoque y se puede decir que da mejores resultados que los filtros que indican Manik Varma y Andrew Zisserman. Los filtros *ICA*, surgen de las propias imágenes teniendo una dependencia clara con las texturas.

Pero esto no acaba aquí. Como línea futura de investigación se podría trabajar con todas las clases que aparecen en la base de datos *CUReT*. Para ello sería necesario o, al menos, altamente recomendable explorar la reducción de modelos evitando con ello un coste computacional que seguramente se antoje excesivo. A modo de ilustrar esto que comento. He de indicar que, por ejemplo, para el caso de 40 clases diferentes un tamaño de *patch* de 16x16 pixeles y un número de bases dominantes igual a 32; el algoritmo llega a tardar casi las 10 horas completas para arrojar los resultados finales de la comparación. Si subiésemos el número de clases se prevé un aumento del tiempo computacional. Por las pruebas realizadas este tiempo, intuyo, que está muy ligado al número de modelos utilizados. Conseguir un código más eficiente puede ayudar también a tener unos mejores tiempos a nivel computacional (aquí entran, quizás, un código con un paradigma más correcto, unos algoritmos más adecuados que converjan más rápidamente, otro tipo de estructuras de datos, etc)

En este trabajo hay que resaltar, que antes de usar esta base de datos, trabajé con una base de datos de patrones lunares con un tamaño de 81x81 pixeles. Ya sea por la falta de muestras, entendido ello como un conjunto lo suficientemente amplio de imágenes que fuesen representativas, quizás, por un tamaño insuficiente en el tamaño de las imágenes (81x81 pixeles) y/o incluso imágenes muy parecidas en su textura; la realidad es que los resultados han sido claramente insuficientes en este campo siendo los mismos, básicamente, como si se tirase una moneda al aire con los resultados que se obtuvieron. Quizás solventando los problemas señalados sería una buena idea probar que resultados se podrían obtener con una base de datos amplia de imágenes relacionadas con la medicina.

ANEXO A: ALGORITMOS

En el presente anexo se da una breve introducción de tres algoritmos de importancia usados en el siguiente trabajo. Existen multitud de buenas referencias para el aprendizaje de estos algoritmos, por ejemplo, en [49] donde se explican de mejor forma y más en profundidad. Lo que se pretende en este anexo es, simplemente, dar un pequeño acercamiento a algunos algoritmos que se comentaron en la introducción.

1.1. Supervisado v.s No supervisado

Como se comenta en la introducción se puede dar una somera clasificación de los algoritmos que se tratan en el presente anexo en supervisados y no supervisados.

1. Los algoritmos de carácter supervisado se centran en establecer una relación entre los datos de entradas y salidas. Ejemplo: *k-Nearest Neighbors(kNN*).
2. Los algoritmos de carácter no supervisados buscan encontrar características en los datos sin ningún tipo de información previa. Ejemplo *k-Means* o *k-Medoids*.

1.2. Algoritmo k-Means.

El cálculo del diccionario de textones se hace a partir del algoritmo *k-means* como previamente se ha expuesto. En este punto, simplemente, se busca dar una breve introducción al algoritmo sin ser exhaustivo. Además de ello, se dará en este anexo una breve pincelada al algoritmo *k-medoids* con el cual he hecho pruebas, también, y que está íntimamente ligado al algoritmo *k-means*. En [21],[50]se puede obtener una mejor comprensión del algoritmo.

K-means es uno de los algoritmos de agrupamiento más ampliamente usado. El algoritmo funciona de la siguiente forma:

Dado un conjunto de datos $D = [x_1, x_2, \dots, x_n]$ donde $x_i \in \mathbf{R}$ es un vector tal que cada $x_i = [x_{i,1}, x_{i,2}, \dots, x_{i,m}]$ siendo m un subíndice que hace referencia al número de atributos en los datos. El algoritmo k-means particiona el conjunto de datos en k grupos/clusters donde cada uno de estos clusters tiene un centro denominado centroide. El parámetro k es definido por quien usa el algoritmo. En el caso de la generación del diccionario de textones, en 2.7.1, el valor de $k = 10$.

1. Como primer paso se establecen los k centroides, por ejemplo, en este primer paso puede ser de forma aleatoria.
2. Para cada $x_i \in D$:
3. Se calcula la distancia de cada x_i a cada centroide (el centroide es el vector que representa al punto central en un cluster C_i).
4. Se asigna cada x_i al centroide más cercano (que representa a un cluster).
5. Una vez iterado en todos los x_i se vuelven a calcular los centroides hasta que se converge (la convergencia se entenderá, en este punto, como el mantenimiento de los mismos centroides al volver a calcularlos) y si no converge se vuelve al punto 2.

Como nota es importante destacar que existen varias formas para definir la distancia. Por norma general se usa la distancia Euclídea, pero existen otras medidas de distancia usadas tal y como la distancia *Manhattan o City-Block*, entre otras.

En la siguiente figura se da un ejemplo gráfico del algoritmo k-Means.

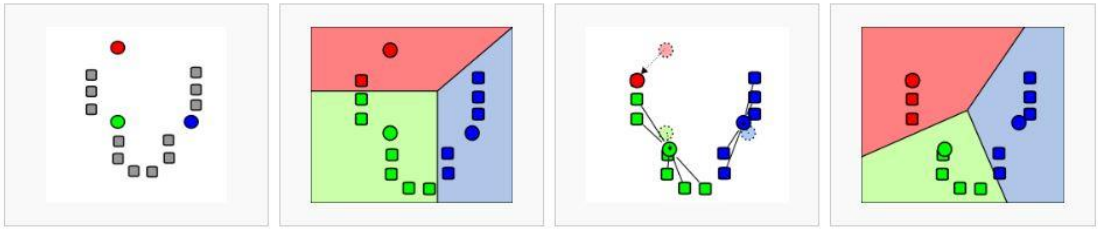

Figura 37: Ejemplo de algoritmo k-Means. Se eligen los centroides de forma aleatoria y van actualizándose los puntos que pertenecen a cada clase y los propios centroides hasta llegar a converger.

1.3 Algoritmo k-Medoids.

Este algoritmo es un algoritmo de agrupamiento similar a *k-means* pues ambos métodos tienen como finalidad encontrar una partición del conjunto de datos en clusters. La principal diferencia entre ambos algoritmos es que *k-medoids* tiene como centro, de cada subconjunto, un punto de dicho subconjunto al cual se denomina medoid (recordar que en *k-means* el centro del conjunto —el centroide— es una media de los puntos del cluster). En este algoritmo se aplica la distancia *Manhattan*, también conocida como distancia L_1. Este algoritmo es más robusto al ruido que el algoritmo *k-means*. Para más información [49], [51].

1.4. Algoritmo k-Nearest Neighbors (KNN)

Es considerado como uno de los algoritmos más simples usados en machine learning; es posible usarlo tanto para problemas de clasificación como de regresión. En el caso del presente trabajo el enfoque que se usa es para la clasificación. Cuando se usa para clasificación la salida que ofrece el algoritmo es la clase a la que pertenecen los datos.

Partiendo del modelo:

$$x_{1,1}, x_{1,2}, x_{1,3} \dots x_{1,m}, y_1$$
$$\vdots$$
$$x_{n,1}, x_{n,2}, x_{n,3} \dots x_{n,m}, y_n$$

Esto nos muestra un modelo donde tenemos n filas y m +1 columnas donde las primeras m columnas son los atributos que usamos para predecir la columna de las y_i (objetivo). También se puede definir una distancia que calcula la distancia, valga la redundancia, entre los datos a clasificar y los datos que como vemos tenemos. Aquella fila que se encuentre más cercana a los datos de entrada será la fila cuya etiqueta y_n identifique a los nuevos datos de entrada. Para calcular los datos se suele usar la distancia Euclídea, pero es posible aplicar otro tipo de distancias como, por ejemplo, la distancia chi-cuadrado, χ^2, tal y como se aplica en el presente trabajo ver 2.8. El valor de k indica el número de puntos más cercanos, siendo estos los "k-vecinos".

En el algoritmo *KNN* la elección del parámetro k tiene importancia. Un valor grande de k puede conseguir que se reduzca la varianza que causen los datos extremos. Sin embargo, hace también que se ignoren patrones pequeños que puedan tener cierta importancia. Por ejemplo, con un valor muy grande la clase más común sería siempre la más votada, pero con un valor muy pequeño una clasificación incorrecta, de los datos en el entrenamiento, podría afectar sobremanera. También es importante decir que si los datos no tienen demasiados datos extremos (no se presenta mucho ruido) el valor de k ,para la clasificación, será de menor importancia puesto que incluso las clases menos representadas tendrá un número suficientemente amplio de ejemplos para votar como vecinos más cercanos. El otro parámetro con especial importancia en el rendimiento es la distancia. Existe multitud de medidas de distancia entre las que se puede nombrar: la distancia Euclídea, la distancia *Hamming*, la distancia de *Manhattan* o la distancia de *Minkowski* (una generalización de las tres distancias precedentes). Sin embargo, tal y como se trabaja en [20] y como se ha comentado previamente, la distancia usada en este trabajo es la distancia chi-cuadrado, χ^2. Aparte de las referencias citadas previamente, se sugiere, por ejemplo, la consulta de [52] para un mejor entendimiento de este algoritmo.

ANEXO B: CÓDIGOS UTILIZADOS.

En este anexo se reproduce parte de los códigos utilizados. Es importante destacar que no es objeto de este trabajo conseguir los códigos de forma que sean lo más eficientes, computacionalmente hablando, posibles. De seguro, se podrían conseguir resultados más satisfactorios en tiempo computacional si se consiguen hacer más eficientes los códigos.

Se reproducirán las partes más notables del código sin entrar en funciones secundarias, en código reiterativo o partes intermedias.

La consecución de la matriz que aglutina los *patches* listados por columnas que después son unidos y ordenados de forma aleatoria para posteriormente conseguir los filtros *ICA*. Es un paso muy similar para la creación del diccionario de *textones* y la parte de clasificación propiamente dicha.

```
% Imágenes y patches de entrenamiento
Ient = struct('imagen', [],'class',[]);

YentImagen = struct('Ye',{},'class',{});
Yent = struct('Ye',{}); % Y para cada clase, combinando todas las imágenes de cada clase
YeImagen = [];

for k = 1:1:secuencia_Entrenamiento(1)
    Ient(k).imagen = Iclase1(k).Icl;
    Ient(k).class = 1;
    YentImagen(k).Ye = generarYsol(Iclase1(k).Icl, tam_patch, sol);%En YentImagen(k) obtenemos, para cada
    %una de las imágenes de entrenamiento, la división en patches. Concretamente, en las columnas de
    %de cada matriz de YentImagen tenemos el tam_patch( Ej: 10*10 -el número de filas
    %será eso-) y el número de columnas será el número de patches que
    %obtenemos de una imagen en concreto(habiendo indicado que tamaño de
    %patch queremos).
    %Cabe destacar que si tenemos todas las imágenes del mismo
    %tamaño el tamaño de YentImagen(k) (para cada imagen)
    %será el mismo.
    YentImagen(k).class = 1;
    YeImagen = [YeImagen YentImagen(k).Ye];%Agrupación de todos los patches de todas las imágenes.
end

Yent(1).Ye = YeImagen(:,randperm(size(YeImagen,2))); %Ordenamos aleatoriamente YeImagen que contiene
%todos los patches de todas las imágenes usadas para entrenamiento. Por
%tanto, tenemos patches ordenados aleatoriamente. Notar que este paso es
%importante, estamos trabajando con una variable aleatoria a la que
%aplicaremos ICA. Para cada Yent(x=1,2,3,4,5).Ye usando ICA reduciremos el
%número de realizaciones de la variable aleatoria, reduciremos el número de
%columnas, ejemplo podría reducirse algo así: 100x832---> 100x64(Aent).Cada fila se ve como una V.A.
```

```
Yent=[Yent(1).Ye,Yent(2).Ye,Yent(3).Ye,Yent(4).Ye,Yent(5).Ye, Yent(6).Ye, Yent(7).Ye,Yent(8).Ye,Yent(9).Ye,Yent(10).Ye, ...
    Yent(11).Ye,Yent(12).Ye,Yent(13).Ye,Yent(14).Ye,Yent(15).Ye, Yent(16).Ye, Yent(17).Ye,Yent(18).Ye,Yent(19).Ye,Yent(20).Ye ];
```

Aplicación del algoritmo *FastICA*:

```
[Sent,Aaux,Baux] = fastica(Yent,'stabilization','on','verbose','off','displayMode','off','approach','symm','lastEig', Nbases,'g','gauss');

    Aent = Aaux;
    Bent = Baux;
```

Se obtiene el banco de filtros:

```
diccionario=Aent;
filter_bank=zeros(tam_patch,tam_patch,size(diccionario,2));

for k = 1:1:size(diccionario,2)

    filter_bank(:,:,k) = defrac(diccionario(:,k),tam_patch,tam_patch); % Esto representa un diccionario de filtros

end



%Normalizo los bancos de filtros para que cada filtro tenga norma L1 igual a '1'.

[a,b,tam_banco_filtros]=size(filter_bank);
filter_bank_L1norm=zeros(a,b,tam_banco_filtros);


for i=1:tam_banco_filtros

    filter_bank_L1norm(:,:,i)= filter_bank(:,:,i)/norm(filter_bank(:,:,i),1);

end
```

Una vez se tienen los filtros ICA toca filtrar las imágenes:

```
for m=1:numel(Ient)

    for  k=1:1:tam_banco_filtros

        if(Ient(m).class==1)

        filterResponsesNormalisedClase1{indice,k}=filter2(filter_bank_L1norm(:,:,k),Ient(m).imagen,'same');

        %Respuesta a los filtros normalizados en cada pixel teniendo en
        %cuenta la ley de Weber.
        norma2_A=abs(filterResponsesNormalisedClase1{indice,k});
        filterResponsesNormalisedClase1{indice,k}=  filterResponsesNormalisedClase1{indice,k}.* ( ( log10( (1+norma2_A)/0.03) )./norma2_A );
```

Las respuestas que se obtienen del filtrado de las imágenes por el banco de filtros ICA se reordenan y, posteriormente, se pasan al algoritmo *k-means* para la obtención del diccionario de *textones*:

```
X18=zeros(altoImg*anchoImg,tam_banco_filtros);
X19=zeros(altoImg*anchoImg,tam_banco_filtros);
X20=zeros(altoImg*anchoImg,tam_banco_filtros);


for m=1:numel(Ient)
    for i=1:tam_banco_filtros

        if(Ient(m).class==1)

           vectorCaracteristicasClase1 = filterResponsesNormalisedClase1{m,i}(:);
            X1(:,i) = vectorCaracteristicasClase1;
        elseif(Ient(m).class==2)

           vectorCaracteristicasClase2 = filterResponsesNormalisedClase2{m-imagenesEntrenamientoParaCrearTextones,i}(:);
            X2(:,i) = vectorCaracteristicasClase2;
```

```
disp('El algoritmo que se va a usar para calcular los textones es kmeans');

[~,centroides1] = kmeans(CaracteristicasClase1,numClusters,'MaxIter',1000);
[~,centroides2] = kmeans(CaracteristicasClase2,numClusters,'MaxIter',1000);
[~,centroides3] = kmeans(CaracteristicasClase3,numClusters,'MaxIter',1000);
[~,centroides4] = kmeans(CaracteristicasClase4,numClusters,'MaxIter',1000);
[~,centroides5] = kmeans(CaracteristicasClase5,numClusters,'MaxIter',1000);
[~,centroides6] = kmeans(CaracteristicasClase6,numClusters,'MaxIter',1000);
[~,centroides7] = kmeans(CaracteristicasClase7,numClusters,'MaxIter',1000);
[~,centroides8] = kmeans(CaracteristicasClase8,numClusters,'MaxIter',1000);
[~,centroides9] = kmeans(CaracteristicasClase9,numClusters,'MaxIter',1000);
[~,centroides10] = kmeans(CaracteristicasClase10,numClusters,'MaxIter',1000);
[~,centroides11] = kmeans(CaracteristicasClase11,numClusters,'MaxIter',1000);
[~,centroides12] = kmeans(CaracteristicasClase12,numClusters,'MaxIter',1000);
[~,centroides13] = kmeans(CaracteristicasClase13,numClusters,'MaxIter',1000);
[~,centroides14] = kmeans(CaracteristicasClase14,numClusters,'MaxIter',1000);
[~,centroides15] = kmeans(CaracteristicasClase15,numClusters,'MaxIter',1000);
[~,centroides16] = kmeans(CaracteristicasClase16,numClusters,'MaxIter',1000);
[~,centroides17] = kmeans(CaracteristicasClase17,numClusters,'MaxIter',1000);
[~,centroides18] = kmeans(CaracteristicasClase18,numClusters,'MaxIter',1000);
[~,centroides19] = kmeans(CaracteristicasClase19,numClusters,'MaxIter',1000);
[~,centroides20] = kmeans(CaracteristicasClase20,numClusters,'MaxIter',1000);
```

```
diccionarioTextones=[centroides1;centroides2;centroides3;centroides4;centroides5;centroides6;centroides7;
 centroides8;centroides9;centroides10;centroides11;centroides12;centroides13;centroides14;centroides15;...
 centroides16;centroides17;centroides18;centroides19;centroides20];
```

Tras obtener las imágenes de entrenamiento y filtrarlas se hace la comparación para conseguir el diccionario de *textones* y los modelos asociados a cada clase:

```
for i=1:numel(Ient)

    for  k=1:1:tam_banco_filtros


          % if(Ient(i).class==1)

                imageModelGeneration{i,k}= filter2(filter_bank_L1norm(:,:,k),Ient(i).imagen,'same');


                %Respuesta a los filtros normalizados en cada pixel teniendo en
                %cuenta la ley de Weber.

                norma2=abs( imageModelGeneration{i,k});
                imageModelGeneration{i,k}= imageModelGeneration{i,k}.* ( ( log10( (1+norma2)/0.03) )./norma2 );
                imageModelGeneration{i,k}=[imageModelGeneration{i,k},tamx*tamy,1)];


     end

[filasImagenFiltrada,~]=size(imageModelGenerationFeatures);
TextonMap=zeros(numTextonesEnDiccionario,filasImagenFiltrada);
for l=1:numTextonesEnDiccionario
      for t=1:filasImagenFiltrada

        TextonMap(l,t)=sqrt(sum( ( diccionarioTextones(l,:)-imageModelGenerationFeatures(t,:) ).^2));

      end

end

 [filaMin,colMin]= min(TextonMap);
```

Se obtienen los modelos:

```
 if(Ient(i).class==1)

       modelosClase1{i}=colMin;
     %  final_TextonMapClass1{i}= reshape(modelosClase1{i},tamx,tamy);
       [votos1,~]=hist(modelosClase1{i},numTextonesEnDiccionario);
       VotosTextonesClase1(i,:)= votos1;



 elseif(Ient(i).class==2)

       modelosClase2{i-numImagenesEntrenamientoPorClase}=colMin;
       [votos2,~]= hist(modelosClase2{i-numImagenesEntrenamientoPorClase},numTextonesEnDiccionario);
       VotosTextonesClase2(i-numImagenesEntrenamientoPorClase,:)= votos2;
```

```
VotosTextones=[VotosTextonesClase1;VotosTextonesClase2;VotosTextonesClase3;VotosTextonesClase4;VotosTextonesClase5;
VotosTextonesClase6; VotosTextonesClase7; VotosTextonesClase8; VotosTextonesClase9; VotosTextonesClase10;VotosTextonesClase11;VotosTextonesClase12;
VotosTextonesClase13; VotosTextonesClase14; VotosTextonesClase15; VotosTextonesClase16; VotosTextonesClase17; VotosTextonesClase18; VotosTextonesClase19;
VotosTextonesClase20; VotosTextonesClase21; VotosTextonesClase22; VotosTextonesClase23; VotosTextonesClase24; VotosTextonesClase25; VotosTextonesClase26;
VotosTextonesClase27; VotosTextonesClase28; VotosTextonesClase29; VotosTextonesClase30; VotosTextonesClase31; VotosTextonesClase32; VotosTextonesClase33;
VotosTextonesClase34; VotosTextonesClase35; VotosTextonesClase36; VotosTextonesClase37; VotosTextonesClase38; VotosTextonesClase39; VotosTextonesClase40];
```

En la parte de test se hace la comparación tras el filtrado de la imagen de test se busca a qué modelo se está más cercano:

```
[filasImagenFiltrada,~]=size(imageModelGeneration_TestFeatures);
TextonMap_Test=zeros(numTextonesEnDiccionario,filasImagenFiltrada);
for l=1:numTextonesEnDiccionario
        for t=1:filasImagenFiltrada

        TextonMap_Test(l,t)=sqrt(sum( ( diccionarioTextones(l,:)-imageModelGeneration_TestFeatures(t,:) ).^2));

        end

end

[~,colMin]= min(TextonMap_Test);


modelosTest{i}=colMin;
final_TextonMapTest{i}= reshape(modelosTest{i},tamx,tamy);


[votosTest,~]= hist(modelosTest{i},numTextonesEnDiccionario);
VotosTextonesTest(i,:)= votosTest;
imageModelGeneration_TestFeatures=[];

textonClaseX= knnsearch(VotosTextones,VotosTextonesTest(i,:),'Distance',@ChiSqrDist);

if (textonClaseX <= numImagenesEntrenamientoPorClase)

    imagenTestClasificadaClase=1;

elseif  (   textonClaseX > numImagenesEntrenamientoPorClase && textonClaseX <= numImagenesEntrenamientoPorClase*2 )

   imagenTestClasificadaClase=2;
```

```
fprintf('La imagen de test pertenece a la clase %d y ha sido clasificada como clase %d\n',Itest(i).class,imagenTestClasificadaClase);


if(Itest(i).class==imagenTestClasificadaClase)

    clasificacion_correcta=clasificacion_correcta+1;
    matrizDeConfusion(Itest(i).class,Itest(i).class)=matrizDeConfusion(Itest(i).class,Itest(i).class)+1;
else

    clasificacion_incorrecta=clasificacion_incorrecta+1;
    matrizDeConfusion(Itest(i).class,imagenTestClasificadaClase)=matrizDeConfusion(Itest(i).class,imagenTestClasificadaClase)+1;
end



end

porcentaje_correctas=(clasificacion_correcta/numImagUsadasParaTest)*100;

fprintf('Imágenes totales %d de las cuales correctamente clasificadas %d que supone un porcentaje de %f%%\n', ...
  numImagUsadasParaTest,clasificacion_correcta,porcentaje_correctas);
```

REFERENCIAS

1. M. Tuceryan and A. K. Jain. Texture analysis. In C. H. Chen, L. F. Pau, and P. S. P. Wang, editors, Handbook of Pattern Recognition and Computer Vision, chapter 2, pages 235276. World Scientic, Singapore, 1993.
2. Samir M. Badawy, Alaa A. Hefnawy, Hassan E. Zidan and Mohammed T. GadAllah, "Breast Cancer Detection with Mammogram Segmentation: A Qualitative Study" International Journal of Advanced Computer Science and Applications(ijacsa), 8(10), 2017
3. Begoña Acha Piñero, Maria Carmen Serrano Gotarredona, R.M. Rangayyan, J.E. Leo Desautels: Detection of Microcalcifications in Mammograms. Recent Advances in Breast Imaging, Mammography and Computer-Aided Diagnosis of Breast Cancer. Bellingham, EE.UU. SPIE- the International Society for Optical Engineering. 2006. Pag.292-308.
4. Gao, Jay. (2009). Digital Analysis of Remotely Sensed Imagery.
5. Pernkopf, Franz & O'Leary, Paul. (2003). Image acquisition techniques for automatic visual inspection of metallic surfaces. NDT & E International. 36. 609-617. 10.1016/S0963-8695(03)00081 1.
6. Chaudhuri, Arindam & Mandaviya, Krupa & Badelia, Pratixa & Ghosh, Soumya. (2017). Optical Character Recognition Systems for English Language.
7. Sclaroff, Stan & Taycher, L & La Cascia, Marco. (1997). ImageRover: a content-based image browser for the World Wide Web. 2 - 9. 10.1109/IVL.1997.629714.
8. Bianconi, Francesco & Fernández, Antonio. (2013). A Unifying Framework for LBP and Related Methods. 10.1007/978-3-642-39289-4_2.
9. Kalle Karu, Anil K. Jain, Ruud M. Bolle,Is there any texture in the image?, Pattern Recognition, Volume 29, Issue 9,1996, Pages 1437-1446, ISSN 0031-3203.
10. Geoff Dougherty. 2013. Pattern Recognition and Classification: An Introduction. Springer Science & Business Media
11. Gurusamy, Vairaprakash & Kannan, Subbu & , G.Nalini. (2014). REVIEW ON IMAGE SEGMENTATION TECHNIQUES.
12. Solomon, C., Breckon, T.: Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab. John Wiley & Sons, Ltd (2010)
13. Dougherty, G. (2009). Digital Image Processing for Medical Applications. Cambridge: Cambridge University Press. doi:10.1017/CBO9780511609657
14. Jain, A. K., Duin, R. P. W., & Mao, J. (2000). Statistical pattern recognition: A review. *IEEE Transactions on pattern analysis and machine intelligence, 22*(1), 4-37.
15. Linda G. Shapiro and George C. Stockman (2001): "Computer Vision", New Jersey, Prentice-Hall, ISBN 0-13-030796-3
16. Gimel'farb G., Zhou D. (2008) Texture Analysis by Accurate Identification of a Generic Markov–Gibbs Model. In: Bunke H., Kandel A., Last M. (eds) Applied Pattern Recognition. Studies in Computational Intelligence, vol 91. Springer, Berlin, Heidelberg
17.]B. Julesz, «Textons, the elements of texture perception and their interaction,» Nature, nº 290, pp. 91-97, 1981
18. Gibson, J.J.: The Perception of the Visual World. Houghton Mifflin, Boston, Massachusetts (1950)
19. J. Hays, M. Leordeanu, A. Efros, and Y. Liu, "Discovering Texture Regularity as a Higher-Order Correspondence Problem," Proc. Ninth European Conf. Computer Vision, vol. 2, pp. 522-535, 2006.

20. M. Varma and A. Zisserman. A statistical approach to texture classification from single images. International Journal of Computer Vision, 62(1-2):61–81, 2005.
21. Duda, R. O., P. E. Hart, and D. G. Stork: 2001, Pattern Classification. John Wileyand Sons, 2nd edition
22. Leung, T. and J. Malik: 2001, 'Representing and Recognizing the Visual Appear-ance of Materials using Three-dimensional Textons'. International Journal ofComputer Vision 43(1), 29–44.
23. Schmid, C.: 2001, 'Constructing models for content-based image retrieval'. In: Proceedings of the IEEEConference on Computer Vision and Pattern Recognition,Vol. 2. pp. 39–45
24. A. Hyvärinen and E. Oja, "Independent Component Analysis: Algorithms and Applications,"Helsinki, 2000.
25. Cao, X., Liu,R.: General approach to blind source separation. IEEE Transactions on Signal Processing, vol.44, no. 3 (1996) 562-571.
26. D. Hubel y T. Wiesel, «Receptive fields, binocular interacition and functional architectura in the cat's visual cortex,» Journal of Physiology, pp. 160: 106-154, 1962
27. D. Hubel y T. Wiesel, «Receptive fields and functional architecture of monkey striate cortex,» Journal of Physiology, pp. 195:2-243, 1968.
28. DeValois, R. L., Lund, E. W., and Hepler, N. (1982).The orientation and direction selectivity of cells in macaque visual cortex. Vision Research, 22:531–544.
29. DeAngelis, G. C., Ohzawa, I., and Freeman, R. D. (1993a). Spatiotemporal organization of simple-cell receptive fi elds in the cat's striate cortex. I. General characteristics and postnatal development. Journal of Neurophysiology, 69:1091–1117.
30. H. B. Barlow, «Unsupervised learning,» Neural Computation, pp. 1:295-311, 1989.
31. P.O. Hoyer and A. Hyvärinen. Independent Component Analysis Applied to Feature Extraction from Colour and Stereo Images. Network: Computation in Neural Systems, 11(3):191-210, 2000.
32. A. Hyvärinen, E. Oja, J. Karhunen. "Independent Component Analysis". Wiley Series on Adaptive and Learning Systems for Signal Processing, Communications and Control, Simon Haykin Series Editor. Editorial John Wiley and Sons , 2001.
33. J. Górriz, C. Puntonet, J. Méndez, M. Salmerón y M. Cazalla, «Implementación en MATLAB de los algoritmos de separación de señales (ICA) basados en el análisis de componentes independientes: ICATOOLBOX2.0.,» Universidad de Cádiz; Universidad de Granada, 2004.
34. A. Hyvärinen, "Fast and Robust Fixed-Point Algorithms for Independent Component Analysis," Helsinki, 1999.
35. Rubén Martín-Clemente y Susana Hornillo-Mellado, "Image processing using ICA: a new perspective " IEEE MELECON 2006, May 16-19, Benalmádena (Málaga), Spain Universidad de Sevilla,Escuela Superior de Ingenieros.
36. Jenssen and Eltoft, "ICA filter bank for segmentation of textured images," 4th International symposiumon ICA and BSS, Nara, Japan, 2003.
37. I. Epifanio (2002): "Descripción de Texturas. Aplicación a su Compresión y Clasificación". Valencia, Universitat de València, 2002. pp. 1-8.
38. J. Malik, S. Belongie, J. Shi and T. Leung, "Textons, contours and regions: cue integration in image segmentation," *Proceedings of the Seventh IEEE International Conference on Computer Vision*, Kerkyra, Greece, 1999, pp. 918 925 vol.2. doi: 10.1109/ICCV.1999.790346
39. S. Zhu, C. Guo, Y. Wang y Z. Xu, «What are Textons?,» International Journal of Computer Vision, nº 62, pp. 121-143, 2005
40. G.-H. Liu, L. Zhang, Y.-K. Hou, Z. yong Li, J.-Y. Yang, Image retrieval based on multi-texton histogram, Pattern Recognition 43 (2010) 2380–2389
41. B. Julesz, Texton gradients: the texton theory revisited, Biological Cybernetics 54 (1986) 245–251
42. Álvarez, Susana & Vanrell, Maria. (2012). Texton theory revisited: A bag-of-words approach to combine textons. Pattern Recognition. 45. 4312-4325. 10.1016/j.patcog.2012.04.032.

43. B. Julesz, J. Bergen, Textons, the fundamental elements in preattentive vision and perception of textures, Bell Systems Technological Journal 62 (1983) 1619–1645.
44. G. M. Farinella, M. Moltisanti, S. Battiato, “Classifying food images represented as Bag of Textons”, 2014 IEEE International Conference on Image Processing (ICIP), pp. 5212-5216, oct. 2014.
45. Y. Javed, M.M. Khan, “Image texture classification using textons”, IEEE 7th International Conference on Emerging Technologies, pp. 1-5., Sept. 2011.
46. J. Yang, X. Feng, E.D. Angelini and A.F. Laine, “Texton and Sparse Representation Based Texture Classification of Lung Parenchyma in CT Images”, 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), pp. 1276-279, Aug. 2016.
47. Susana Hornillo Mellado, "Sobre el análisis en componentes independientes de imágenes naturales", Tesis Doctoral, Universidad de Sevilla, Escuela Superior de Ingenieros, 2005.
48. Yousra Javed, Muhammad Murtaza Khan, 'Image texture classification using textons'. 7th International Conference on Emerging Technologies,20.
49. Christopher M CM Christopher M. Bishop, Pattern recognition and machine learning, vol. 4, 2006.
50. MacQueen, J.B.: Some methods for classification and analysis of multivariate observations. In Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, pp. 281–297. University of California Press, Berkeley (1967).
51. Kaufman, L. and P. J. Rousseeuw: 1990, Finding Groups in Data: An Introduction to Cluster Analysis.John Wiley & Sons, NY, USA.
52. Kramer O. (2013) K-Nearest Neighbors. In: Dimensionality Reduction with Unsupervised Nearest Neighbors. Intelligent Systems Reference Library, vol 51. Springer, Berlin, Heidel.