UNIVERSIDAD CARLOS III DE MADRID

Inteligencia en Redes de Comunicaciones

Ingeniería de telecomunicación

RECONOCIMIENTO DE PATRONES EMPLEANDO REDES NEURONALES BOOLEANAS.

APLICACIÓN AL RECONOCIMIENTO FACIAL

Práctica realizada por:

Alfonso Mateos Andaluz	N.I.A.: 100027597
Pablo Javier Borches Juzgado	N.I.A.: 100021524

Introducción a la práctica

A lo largo de la asignatura hemos analizado el concepto de inteligencia en los sistemas informáticos y de comunicaciones y hemos estudiado algunas de técnicas que permiten incorporar comportamientos "inteligentes" tanto en las redes como en los servicios.

En esta práctica, se orienta la implantación de comportamiento inteligente en sistemas de inteligencia artificial hacia la adquisición de la capacidad humana del aprendizaje, reconocimiento y clasificación de patrones.

Para ello emplearemos como sistema básico las redes neuronales, aunque dada la complejidad de los patrones que pretendemos reconocer implementaremos un tipo particular de redes cuyo funcionamiento se detallará y se explicará con ejemplos más sencillos como vía para comprender su principio de funcionamiento.

En nuestro caso vamos a emplear la capacidad de aprendizaje y clasificación de las redes neuronales para reconocer patrones, en concreto implementando un sistema de reconocimiento facial.

Emplearemos como lenguaje de programación Java, debido a que es multiplataforma (lo que nos permite programar y ejecutar indistintamente en Linux y Windows) y a que presenta clases y métodos ya implementados útiles para algunos de nuestros propósitos como el tratamiento de niveles de gris, etc.

Para verificar el funcionamiento del programa utilizaremos fotografías de compañeros de clase: entrenaremos la red neuronal con unas (pocas) imágenes de entrenamiento para cada persona, y posteriormente la red tendrá que reconocer otras fotografías de esa misma gente, más o menos similares dependiendo de las diferentes expresiones de la cara y otros factores, decidiendo de quién se trata.

Introducción a las redes neuronales

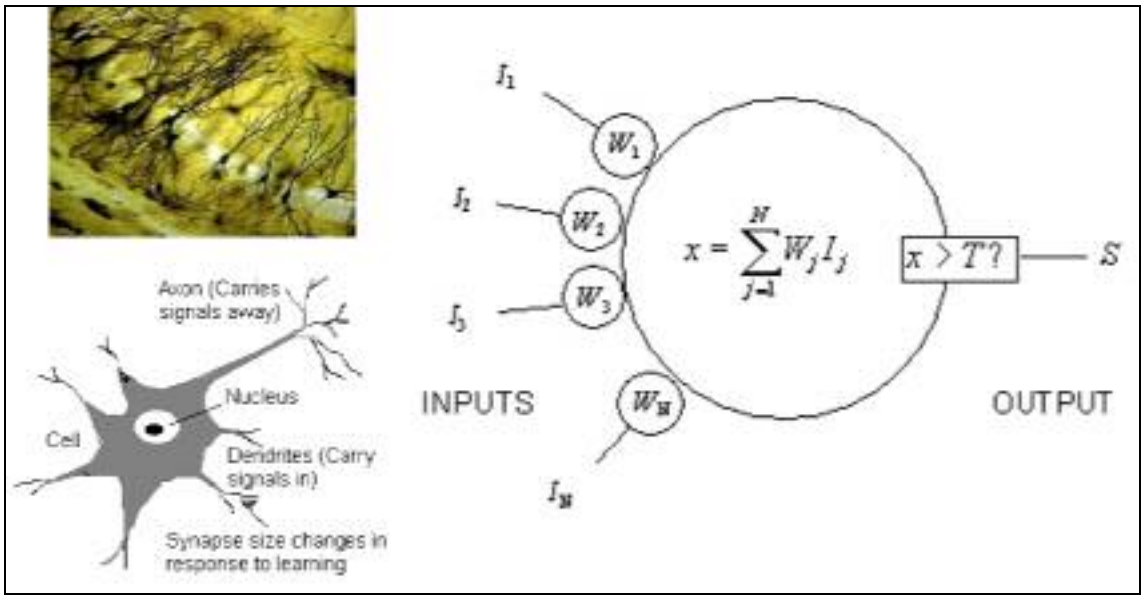
Una red neuronal es una implementación "máquina" de un algoritmo de aprendizaje inspirado en la forma en que el cerebro realiza una tarea de aprendizaje particular.

Las redes neuronales son capaces de "aprender" la relación existente entre sus entradas y salidas, y como su propio nombre indica están inspiradas en el modelo biológico. Al igual que el cerebro está formado por neuronas, las redes neuronales están formadas por pequeñas unidades centrales de proceso del mismo nombre que emulan su funcionamiento, y su comportamiento se basa en una simplificación matemática del esquema estímulo/respuesta de las neuronas.

Las redes neuronales se fundamentan en tres cualidades básicas del cerebro humano:

– El conocimiento está distribuido en un (gran) número de neuronas en el cerebro.
– Las neuronas se comunican (de manera local) unas con otras.
– El cerebro es adaptable (puede aprender).

La analogía y los paralelismos comentados se pueden observar al comparar la estructura de las neuronas con el esquema general de una red neuronal:



La red neuronal toma una decisión teniendo en cuenta las entradas proporcionadas y aplicando sobre ellas las correcciones adecuadas lleva a cabo la clasificación. El conocimiento necesario en la tarea de aprendizaje y clasificación se proporciona en forma de muestras (generalmente etiquetadas) llamadas ejemplos de entrenamiento.

Al diseñar una red neuronal como sistema de clasificación y reconocimiento se persiguen dos características enfrentadas pero ambas fundamentales:

- ha de discriminar entre las distintas clases posibles de la forma más precisa posible, distinguiendo los rasgos que pueden resultar diferenciadores y determinantes.
- ha de generalizar adecuadamente, es decir, ha de comportarse y clasificar correctamente ante nuevas instancias de muestras a reconocer de acuerdo con lo aprendido a través de las muestras de entrenamiento.

Las redes neuronales se usan generalmente como mecanismos clasificadores. Esto significa que ante una determinada entrada desconocida la red neuronal decide, de entre los posibles grupos entre los que puede elegir realizar la clasificación, en cuál de ellos encaja mejor atendiendo a un criterio de semejanza.

Pero además, las redes neuronales pueden actuar como memorias con contenido direccionable, es decir, pueden almacenar patrones complejos para ser luego capaces de reconstruir dichos patrones a partir fragmentos o muestras dañadas de los mismos en base a los ya almacenados siguiendo también un criterio de máxima semejanza.

Estos comportamientos son posibles gracias a que, como ya se ha comentado, las redes neuronales aprenden basándose en la experiencia, pudiendo adaptarse de forma dinámica y actuar en consecuencia ante las nuevas entradas no conocidas.

A este tipo de redes se les puede enseñar a realizar tareas que resultarían difíciles de llevar a cabo mediante los algoritmos de computación tradicionales (como por ejemplo el reconocer patrones –fonemas, formas, dígitos, caras de la gente, etc. –).

Hemos de señalar que aunque las redes neuronales se suelen implementar en código ejecutable (software), hay una creciente tendencia a fabricar chips con la red neuronal diseñada directamente en forma de componentes electrónicos (hardware) con el consiguiente aumento de velocidad de proceso, decisión y clasificación.

Redes neuronales booleanas (BNN)

Las redes neuronales booleanas difieren de las redes neuronales más tradicionales (como el ADALINE, el Perceptron, etc.) en que usan elementos de lógica booleana como componentes básicos y en que no están estructuralmente tan íntimamente ligadas a funcionamientos observados en la naturaleza.

Los elementos de lógica booleana más simples son las puertas lógicas, aunque también se emplean dispositivos más complejos como memorias, ya que éstas se pueden construir en base a puertas lógicas simples.

El hecho de que los componentes estructurales básicos sean elementos de lógica booleana significa que dichos elementos actúan en base a información y reglas lógicas (con ceros y unos), en lugar de hacerlo en base a la suma ponderada de las entradas (números reales) como en las redes neuronales más tradicionales, por lo que las BNN también se suelen conocer como redes neuronales "sin pesos".

En este trabajo implementaremos un tipo de red neuronal booleana llamada WISARD, nombre debido a las iniciales de las personas que lo crearon: "Wilkie, Stonham and Alexander's Recognition Device". Se basa en una red neuronal descrita anteriormente por Bledsoe y Browning en 1959 que emplea conjuntos de memorias direccionables para reconocer patrones, por lo que comenzaremos explicando el principio de funcionamiento de ésta máquina aplicándola a un modelo sencillo.

Este tipo de redes neuronales booleanas comparten una estructura funcional común que se puede resumir brevemente y en primera aproximación como sigue.

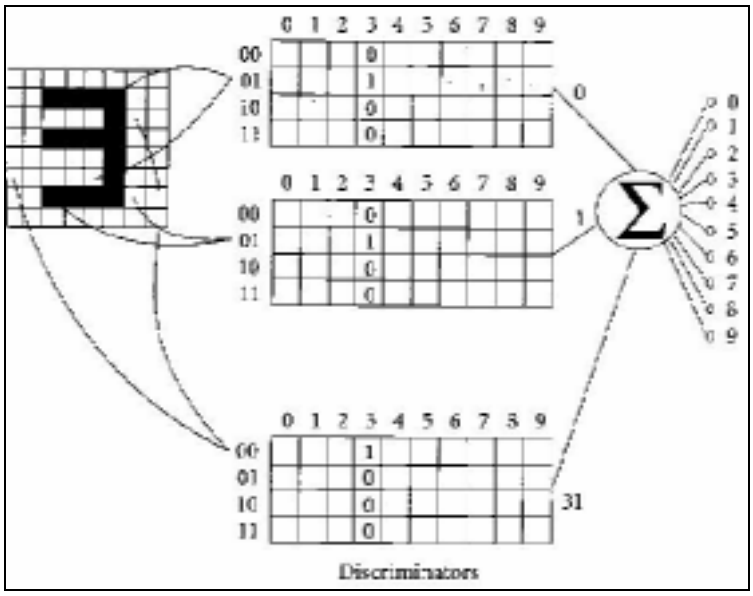
- Reciben como entrada una matriz de píxeles que conforman la imagen del patrón de entrada (tanto durante el entrenamiento como en el posterior proceso de reconocimiento).
- El núcleo de la red está compuesto por memorias RAM, y como tales los bits que almacenan están organizados en direcciones (filas) pudiéndose adicionalmente direccional por posiciones dentro de cada dirección de memoria (columna).
- Los píxeles de la imagen de entrada se agrupan (de forma aleatoria) en grupos, todos ellos con el mismo número de píxeles. Dichos grupos, en el caso general de conjuntos de n píxeles, son conocidos como n-tuplas.
- La combinación binaria formada por la información contenida en los píxeles de cada grupo sirve para escribir la información relativa a esa imagen particular en la columna correspondiente de todas las memorias (cada "columna" corresponde a un grupo de clasificación que es lo que en esencia queremos diferenciar, por lo que también se conocen como "discriminadores").
- Comparando los resultados de las columnas resultado del entrenamiento con la salida proporcionada por la red cuando trabaja sobre una muestra sin clasificar y que ha de reconocer, se clasifica esa muestra en el grupo con el que tenga el mayor número de bits coincidentes.

La máquina de Bledsoe y Browning

Ésta máquina es una red neuronal booleana “sencilla” basada en chips de memoria RAM, cuya misión consiste en reconocer un patrón determinado (o patrones que correspondan a éste con modificaciones “asumibles” que permitan que la semejanza entre ellos sea razonable como para identificar uno mediante el otro) y clasificar dicho patrón de entrada como perteneciente a uno de los conjuntos conocidos, establecidos y aprendidos en el proceso de entrenamiento.

Para explicar su funcionamiento, vamos a ilustrar un ejemplo sencillo y de tamaño manejable, aplicado al reconocimiento de patrones, en el que se aprecia el principio de funcionamiento de este tipo de redes.

El ejemplo ilustra el esquema de reconocimiento de un dígito, ya que nos permite tener un modelo de tamaño muy reducido (muy pocos píxeles) para ver fácilmente el funcionamiento.



Consideramos como entrada una cuadrícula de píxeles correspondiente a la imagen de un dígito (0↔9) en blanco y negro de dimensiones 8x8 píxeles a reconocer.

Tenemos que conectar de forma aleatoria todos los píxeles de nuestra imagen, agrupados de una forma conveniente, a los chips de memorias, de forma que conseguimos un mapeo entre píxeles y direcciones de memoria.

Dado que tenemos 64 píxeles en total, y si agrupamos los píxeles por parejas de forma que no haya solapamientos en las asignaciones, formaremos 32 “agrupaciones” de parejas de píxeles.

Como cada píxel puede tomar los valores 0 o 1 (blanco o negro respectivamente), una pareja de bits codifica 22=4 valores (00, 01, 10, 11). De esta forma, vemos que necesitamos 32 memorias con 4 direcciones (filas) cada una ya que cada pareja de píxeles requerirá 4 direcciones de memoria para poder así representar sus 4 posibles valores a que da lugar la decodificación de los 2 bits. Además, debido a que tenemos 10 grupos de clasificación porque tenemos 10 posibles dígitos (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), necesitamos que cada dirección tenga diez posiciones (tantas como posibles grupos de clasificación).

Durante el entrenamiento se le muestran al sistema varios ejemplos de cada número, de forma que para cada cualquier ejemplo en particular las 32 memorias tienen un 1 en la dirección (fila) determinada por la codificación de los valores de la pareja de píxeles y en la posición (columna) correspondiente al número del que se trate (el que se está mostrando en ese momento).

Tras el entrenamiento un número a reconocer es mostrado a la entrada y las salidas de las 32 memorias se comparan con las resultantes del proceso de entrenamiento, sumándose los aciertos por columnas (hay una columna por cada dígito: "discriminadores"). Por tanto las 10 salidas pueden variar entre 0 y 32, y el dígito seleccionado como el reconocido será el que tenga el mayor valor de esta suma (los empates podrían resolverse aleatoriamente, aunque no sería conveniente llegar a tal situación de duda en el reconocimiento).

La memoria utilizada para este caso particular es sorprendentemente pequeña, ocupándose un total de 1280 bits.

WISARD

Una vez visto un ejemplo sencillo y descubierta la estructura y el funcionamiento de las una red neuronal booleana, podemos extender la misma metodología ampliando el número de bits a agrupar en cada n-tupla, con lo que llegamos a una estructura conocida como WISARD basada en la anterior pero más compleja (si bien una vez comprendido el funcionamiento del sistema anterior, éste es fácilmente extensible al que ahora nos ocupa).

Para el caso general de emplear grupos de n píxeles con imágenes compuestas por P píxeles y D posibles clases en las que clasificar, el tamaño de memoria total necesario será:

$$M = \frac{P}{n} \cdot 2^n \cdot D$$

Como se puede observar, a medida que n aumenta, se incrementan los requerimientos de memoria, lo cual impone un límite físico al tamaño de n.

Los casos extremos en el número de píxeles a agrupar en cada n-tupla serían n=1 y n=P.

- Con n=P la enorme cantidad de memoria necesaria hace que esta opción no sea práctica, aunque el principal inconveniente es la pérdida de capacidad para generalizar. Para valores altos de n (aún sin llegar a n=P) el sistema tiende a perder la capacidad de generalizar.
- El otro extremo es n=1, que tiene capacidad generalizadora pero es de poca utilidad debido a su facilidad para llegar al estado de saturación (todas las posiciones de la memoria a 1 tras varias muestras de entrenamiento). Para valores de n mayores que 1 el problema de la saturación tiende a desaparecer.

Por tanto y como ya se señaló con anterioridad hay un compromiso entre los sistemas con un valor pequeño de n y los sistemas con valores mayores de n.

Se puede comprobar que para un tamaño de imagen concreto, a medida que aumenta el valor de n aumenta también el porcentaje de clasificaciones correctas hasta un valor máximo, pero a partir de ese máximo el porcentaje de clasificaciones correctas decrece hasta llegar a cero. Si el tamaño de la imagen aumenta, el máximo se alcanza con un

valor también mayor de n, pero la forma de la curva de comportamiento de la red neuronal sigue siendo la misma.

De lo comentado podemos concluir que existe un equilibrio que tenemos que mantener entre el tamaño de memoria requerido, el valor de n y la cantidad de datos de entrenamiento (si entrenamos con demasiados patrones, se pierde la capacidad de generalizar ya que la red se especializa demasiado).

Tenemos que señalar sin embargo, que siempre habrá algún patrón que la red basada en el sistema de n-tuplas no será capaz de reconocer y clasificar correctamente, pero esta es una debilidad también propia de cualquier otro sistema de reconocimiento.

Esta red neuronal muestra a su salida el número de coincidencias que tiene la imagen de test con cada uno de los discriminadores, por lo que la salida se suele representar como un gráfico de barras en un monitor, representando cada barra la probabilidad de que el patrón a clasificar pertenezca a una u otra clase. La respuesta con mayor valor indica la correspondencia a la clase indicada por dicho valor máximo.

Dado que es usual que sólo una parte de la imagen sea la que contenga la información relevante, a menudo se suelen "recortar" la imagen inicial fijando una ventana ajustada al área en la que se encuentran los píxeles que realmente son de interés, de forma que disminuyen los requerimientos de memoria.

La gran ventaja de este sistema es que toda la computación se hace en paralelo, de forma que el sistema puede operar en tiempo real (pudiendo llegar con implementaciones hardware hasta 25 imágenes reconocidas por segundo con imágenes pequeñas como las del ejemplo del dígito).

Análisis del WISARD

A continuación trataremos de dar una visión más analítica del principio en el que se basan estos tipos de redes neuronales booleanas.

Como ya se ha visto, cuando se entrena el WISARD se le muestra un patrón cada vez y en cada una de las K memorias RAM, se escriben unos en las posiciones determinadas por las n-tuplas dentro del discriminador correspondiente al patrón con el que se está entrenando.

Por tanto, si el tamaño de la imagen es de P píxeles, el número de memoria que necesitaremos si queremos direccional mediante agrupaciones de n bits es:

$$K = \frac{P}{n}$$

Al presentar otra imagen perteneciente al mismo "objeto" a la entrada pero esta vez con el fin de ser reconocida y clasificada, habrá cierto solapamiento (considerable, ya que se trata del mismo "grupo de clasificación") entre las imágenes anteriores de entrenamiento y la nueva muestra. Con solapamiento nos referimos a píxeles con el mismo valor, bien 0 o 1, y denotaremos como "A" a dicho número de píxeles coincidentes.

La probabilidad de que el discriminador j-ésimo sea escogido como el "más probable" para la imagen de test depende de la probabilidad de que m líneas de las memorias de tengan idénticos valores (0 o 1) para ambos patrones. Esto ocurre cuando las n-tuplas son escogidas de áreas que se solapan, es decir, cuando codifican el mismo valor porque todos sus píxeles tienen la misma "tonalidad". La probabilidad de seleccionar m píxeles correspondientes a áreas que se solapan es:

$$P = \left(\frac{A}{P}\right)^m$$

El valor de salida del discriminador j-ésimo, rj, asumiendo que las coincidencias entre las salidas de los discriminadores en el entrenamiento y la salida de la muestra de test de las RAMs se suman para obtener la puntuación de la clasificación, es:

$$r_j = K\left(\frac{A}{P}\right)^n$$

Claramente, si dos patrones son idénticos tenemos A=P y la salida será r_j=K.
Si normalizamos las salidas a este valor máximo, tenemos que el r_j normalizado es:

$$r_{j\_n} = \left(\frac{A}{P}\right)^n$$

En base a este último parámetro podemos determinar la certeza con que una decisión es tomada, es decir, la cuán seguros o confiados podemos estar de que la clasificación decidida es correcta o no. Su expresión es:

$$C = \frac{r_1 - r_2}{r_1} = 1 - \frac{r_2}{r_1}$$

Donde r_1 es la respuesta (puntuación) del discriminador más probable y r_2 es la valoración del siguiente discriminador por debajo del primero.

La certeza máxima sería C=1 pero esto supondría que sólo el discriminador seleccionado tuviera puntuación no nula, lo cual no es posible, por lo que como es esperable siempre habrá cierto grado de incertidumbre y siempre cabrá la posibilidad de una clasificación incorrecta aunque su probabilidad sea muy pequeña.

Teniendo en cuenta el número de píxeles coincidentes, que en definitiva es el factor realmente determinante, y sustituyendo en la expresión anterior obtenemos:

$$C = 1 - \left(\frac{A_2}{A_1}\right)^n$$

Con lo que se comprueba que la certeza en una decisión de clasificación mejora al aumentar n, al igual que aumenta el poder discriminador. Sin embargo hemos de tener en cuanta que al aumentar el valor de n la red pierde su capacidad de generalización, que es una característica de vital importancia en las redes neuronales empleadas como clasificadores.

Argumento a favor de la agrupación aleatoria de píxeles

Hemos venido comentando que los píxeles que agrupamos para formar las n-tuplas han de ser escogidos de forma aleatoria de entre los píxeles de toda la imagen y en este punto vamos a dar una justificar el por qué.

Si los píxel que conforman una n-tupla están seleccionados de forma “determinista” es probable que nos encontremos con patrones con formas también más o menos “deterministas” que la red no será capaz de clasificar correctamente ya que confundirá los discriminadores.

Por ellos es conveniente conectar de forma aleatoria los píxeles que conforman las n-tuplas. De esta forma no logramos eliminar la posibilidad de errores en la clasificación por confusión entre discriminadores, pero al menos conseguiremos que los patrones que

pueden llevar a este tipo de confusiones sean patrones con una estructura también "aleatoria" que será muy poco común que nos topemos con ellas en nuestros casos prácticos, ya que por definición la "aleatoriedad" no tiene un patrón definido al que responda y que pueda asemejarse a casos reales.

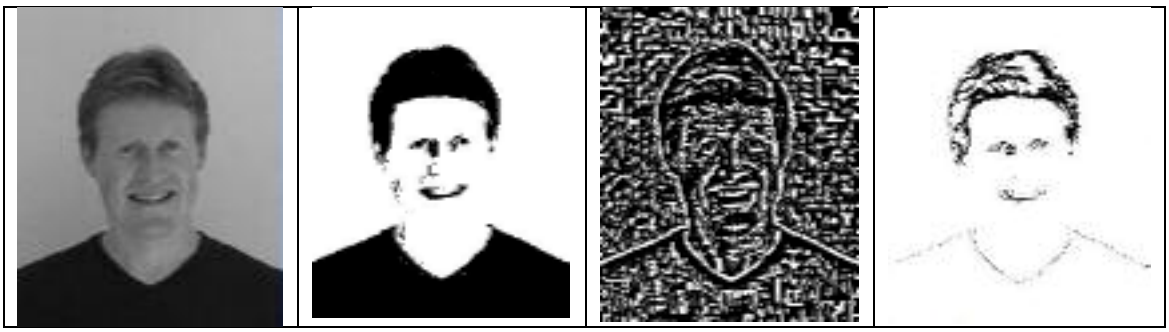
Codificando niveles de gris o tonalidades de color

En el WISARD se pueden emplear imágenes en escala de grises e incluso en color, pero dado que las redes neuronales booleanas trabajan con valores lógicos, 0 y 1, los colores que no son el blanco (0) y el negro (1) se tienen que codificar para expresarlos mediante un código binario. En muchas aplicaciones es suficiente con establecer un umbral en la escala de grises para pasar la imagen a un patrón en blanco y negro codificable directamente en valores lógicos (si tenemos una imagen en color tenemos que convertir dicha imagen en una imagen en es cala de grises para luego aplicar el umbral comentado).

Si quisiéramos emplear cada píxel con su tonalidad original correspondiente tendríamos que codificar cada color mediante un cuya longitud variará dependiendo del número de tonalidades a distinguir, siendo los más empleados los códigos de escala termométrica y la codificación "1-en-n".

Hemos de tener muy en cuenta que este tipo de codificaciones hacen que los requerimientos de memoria se disparen, ya que si tenemos x niveles de color, cada píxel requerirá como mínimo $b = \log(x)/2$ bits para representarlo ($x = 2^b$), y es por este motivo por lo que si no es imprescindible se tiende a emplear una representación en blanco y negro de la imagen original.

Atendiendo a la transformación a blanco y negro (ya que es la que emplearemos en nuestra implementación como posteriormente se verá) hemos de señalar que la elección del umbral en base al cual se realizara la conversión de color/grises a blanco y negro es crítica igual que lo es la elección del método de transformación. Dependiendo del valor de este umbral la imagen se verá de formas muy distintas, y la eficiencia en la clasificación cambiará mucho. A modo de ejemplo veamos las siguientes imágenes:



Como se puede apreciar según el método de transformación y del valor del umbral la imagen sobre la que se realizara el procesamiento varía grandemente, siendo unas transformaciones claramente más convenientes que otras a la hora de lograr la mejor clasificación.

Aún más claro se ve este factor en las siguientes imágenes, en las que se desea reconocer texto en unas fotografías (que al fin y al cabo es un patrón como cualquier otro, y como lo son también las caras de la gente):

Como se aprecia, en la imagen de la derecha es imposible distinguir el texto mientras que en la derecha, se puede leer fácilmente con lo que se podría aplicar un sistema de reconocimiento con éxito.

Mejoras posibles.

A parte de esta configuración básica ya comentada, se pueden implementar versiones de redes neuronales booleanas con funcionalidades mejoradas.

Una mejora interesante consiste en dotar de realimentación a la red neuronal de forma que el proceso de reconocimiento de un patrón se realimente a sí mismo: tomando como entradas la imagen de una cámara que apunte a la imagen a reconocer y otra cámara al gráfico de barras de salida de la red se puede producir un nuevo patrón d entrada compuesto y con mezcla de información de la imagen original y la salida que produce de forma que la red se comporta de forma adaptativa mejorando la clasificación realizada durante varios reconocimientos cada vez más finos.

Implementación del sistema

El código que implementamos sigue los principios ya explicados. En este apartado se describen los criterios de diseño y se justifican los valores de los distintos parámetros del programa.

En primer lugar, comentar que emplearemos fotografías de compañeros de la universidad que amablemente se ofrecieron para el propósito que nos ocupa. Las fotografías se tomaron con una cámara digital en color y se redujeron a tamaño 800x600 pixels, algo más manejable que el tamaño original pero aún así bastante grande (480000 pixels por imagen). Este tamaño (grande) de las fotografías contribuye a que el tiempo de proceso tanto para entrenamiento como para reconocimiento sean mas elevados de lo esperable, pero esto tiene solución si más que reducir el tamaño de las fotografías (no hemos hecho esto para que durante la presentación las imágenes se vean con un tamaño razonable).

Hemos de tener también en cuenta que este tipo de redes neuronales booleanas están pensadas y estructuradas para ser implementadas en hardware y ejecutadas por él directamente sin ayuda de software, por lo que los tiempos reales de una máquina hardware que lleve a cabo esta misma tarea y con estas mismas dimensiones de imagen será notablemente menor al empleado en una ejecución software.

En cualquier caso, y dado que como ya se ha comentado en una fotografía de un retrato no ajustado la cara no ocupa toda la imagen, hemos enventanado la cara dentro de las fotografías para emplear en el procesado sólo los pixels correspondientes a un recuadro más o menos ajustado a las dimensiones medias de las caras. Es decir, de los 800*600 pixels hemos tomado como píxeles que contienen la cara aquellos enmarcados entre las “filas” 80 y 480 y las "columnas” 250 y 550, con lo que el procesado se hará con imágenes de 300*400 (120000 píxeles).

Para la elección del valor de las n-tuplas, hemos tenido que hacer pruebas con distintos valores ya que los resultados varían de forma no lineal. Al final y tras varias pruebas escogimos agrupar los pixels en conjuntos de 4. Con cuatro píxeles podemos direccional 2^4=16 direcciones de memoria, por lo que las RAMs tendrán 16 líneas cada una. Además, con estas agrupaciones tenemos que el número de RAMs a utilizar es de $(300*400)/4 = 30000$. Finalmente la longitud de cada línea de memoria dependerá del número de patrones que deseemos reconocer, de forma que si queremos reconocer D patrones, en nuestra implementación particular requeriremos 30000*4*D bits de memoria, que es una cantidad bastante elevada (120kbits de memoria por cada persona) pero que hoy en día supone un coste perfectamente asumible.

Finalmente comentaremos el valor del umbral elegido. Como ya se ha comentado, este tipo de redes trabajan con valores lógicos por lo que es necesario convertir los colores a bits. En nuestra implementación pasamos las imágenes en color a imágenes en escala de grises (256 niveles), y estas son las imágenes de entrada al sistema. Para ahorrar memoria pasamos cada píxel a blanco o negro según la comparación de la tonalidad original con un umbral. Según nuestras pruebas el sistema reconoce y clasifica adecuadamente tras esta transformación, por lo que nos encontramos ante una aplicación en la que, con el número de píxeles que tenemos, la profundidad de color de no es especialmente determinante (aunque lógicamente los valores de certeza serán mejores cuanto más fiel sea la imagen procesada a la imagen original). Tras varias

pruebas llegamos a la conclusión de que un umbral de valor 20 era el óptimo según las condiciones de nuestro sistema.

Ficheros de entrada: configuración e imágenes

Nuestro programa necesita como es lógico, los ficheros de las imágenes de entrenamiento y de las imágenes de test. Éstas han de estar en formato de mapa de bits (*.bmp), y las dimensiones de la imagen han de ser 800*600 y en escala de grises con 256 niveles como ya se ha comentado. Todas las imágenes de las caras han de estar centradas y su tamaño ha de ser semejante. Los ficheros de entrenamiento para una persona en concreto han de contener imágenes de su cara, con variaciones asumibles como pueden ser gestos faciales que las diferencien (una buena idea es también entrenar con imágenes que sean otra imagen del entrenamiento pero con ruido o pequeñas distorsiones, ya que esto hace más robusto en entrenamiento y no supone un esfuerzo extra en la consecución de nuevas muestras).

Además, como ficheros de configuración el programa necesita los archivos "bmps.txt" y "names.txt".

El fichero "bmps.txt" contiene la información relativa a los ficheros de las imágenes de entrenamiento. En la primera línea se especifica el número total de imágenes de entrenamiento disponibles, y en la línea siguiente se especifica cuantas imágenes (consecutivas) corresponden a cada persona (todas las personas se han de entrenar con el mismo número de imágenes). Las líneas siguientes son los ficheros de mapa de bits, ordenados secuencialmente y con todos los ficheros correspondientes a la misma persona, seguidos.

El fichero "names.txt" contiene los nombres de las personas de la base de datos. Estos nombres han de estar en el mismo orden en que se encuentran en el fichero "bmps.txt"

Todos estos archivos han de encontrarse en el mismo directorio desde el que se ejecuta el programa.

Desarrollo del programa

Para desarrollar el código que ejecutará nuestra aplicación utilizamos el lenguaje orientado a objetos Java. Este lenguaje tiene la ventaja de lo intuitivo que resulta trabajar con objetos así como la facilidad que tiene el ir haciendo futuras ampliaciones de una versión finalizada. Como desventaja hay que destacar que al ser un lenguaje que no es compilable, sino que es interpretado, el tiempo de ejecución es mayor que el de un lenguaje compilable como por ejemplo C. A pesar de ello hemos conseguido un reconocimiento de caras con un tiempo de ejecución no demasiado elevado (menos de un minuto por fotografía a reconocer) y la organización del código en clases ha permitido una buena estructuración del programa.

Nuestro programa está estructurado en tres clases bien diferenciadas:

- Neural: clase principal del programa
- Pattern: clase que representa la imagen de una persona a partir de un patrón de píxeles de 8 bits, y su correspondiente patrón de 1 bit resultado de aplicarle un valor umbral (threshold).
- RAM: clase que representa la matrices de nuestra red neuronal booleana. Formada por un número de columnas que son las que van a representar el número de personas

del grupo que la red es capaz de reconocer; y un número de filas 2^n que son todos los posibles valores de los bits escogido para las tuplas, siendo n pues el valor de la n-tupla. Ésta es la clase más corta en cuanto a líneas de código y métodos que contiene, sin embargo es la más importante, ya que contiene lo que realmente es la estructura de la red neuronal. Además es la clase que resulta menos intuitiva ya que contiene bucles muy complicados que realizan tanto la selección de las muestras en el patrón a partir del número de tuplas escogido y el tamaño de la imagen, como el entrenamiento de la red, que es el proceso más importante.

A continuación se muestra en un árbol las distintas clases con sus diferentes métodos

- Neural
  - public main()
  - public run()
  - public readNames()
  - public readRAMsFromFile
  - public recognizePattern(Pattern p)
  - public saveRAMsIntoFile()
  - public trainNetwork()
  - private maxNumber(int[] num)
  - private namePerson(int n)
  - private selectMemoryPosition(boolean[][] bPixels, RAM ram)

- Pattern
  - public selectPixelsOfTheFace()
  - public testBoolPixels()
  - public testPixels()
  - private loadDataFromFile(File f)
  - private thresholdFilter(int threshold)

- RAM
  - public selectRAMsamplesInPattern(int n, int rows, int columns)
  - public train(Pattern f, int n)

Proceso de ejecución del programa

El programa comienza con el método main() imprimiendo en pantalla la presentación con las siguientes opciones:

1) Reconocer cara usando datos grabados en archivo
2) Reconocer cara entrenando la red
3) Reconocer cara entrenando y almacenar RAMs de entrenamiento en archivo

A continuación se crea una clase Neural y el método main() acaba llamando al método run() es una continuación del anterior, implementado únicamente por eficiencia en la programación.

Suponiendo que hemos escogido la segunda opción, la ejecución continúa llamando al método trainNetwork() de la clase Neural que lee los ficheros de entrenamiento especificados en el archivo “bmps.txt” y les aplica el recorte selectPixelsOfTheFace()

para quedarnos con los bits que más nos interesan (el marco de la cara) y crea los objetos RAM necesarios (unos 30.000 para un reconocimiento con 5 personas). La creación de estos objetos es uno de los procesos que más tiempo llevan al programa para ejecutarse. A continuación trainNetwork() llama a train() de la clase RAM que es el que realmente entrena la red aplicando los unos correspondientes a la matriz RAM mediante el uso de unos complicados bucles. Este último es un método corto pero complicadísimo de elaborar, basta con fijarse en la dificultad que tiene el siguiente bucle:

```
for (int i=0; i<numTuple; i++)
     aux[i] = f.boolPixels[posX[i]][posY[i]];
```

Posteriormente el programa lee el fichero a reconocer y le aplica el marco que selecciona de los píxeles de la cara creando el objeto Pattern de esta imagen y se crean los ficheros **"test1.bmp"** y **"test2.bmp"** que muestran el resultado de ese proceso (contenido de las matrices pixels[][] y boolPixels[][]) y la imagen resultante en tono de grises de 8 bits y píxeles blancos y negros de 1 bit respectivamente. La idea era que se pudiera ver con cualquier programa de dibujo como Paint, pero al ver la dificultad que tenía reproducir la cabecera BMP para que se pudiera visualizar, lo dejamos sin cabecera pudiéndose observar con el programa de texto Notepad con un tamaño de letra pequeño.

Finalmente se ejecuta el método recognizePattern() que realiza el reconocimiento del patrón introducido como parámetro a partir de los datos de todas las matrices RAM almacenados en el array RAMs[]. Para ello utiliza el segundo método más complicado del programa llamado trainselectMemoryPosition(), que selecciona los 4 píxeles (para n=4 en la n-tupla) correspondientes a cada posición de la matriz RAM.

Si se hubiera escogido la opción 3, se ejecutaría posteriormente el método saveRAMsIntoFile() que guardaría en un fichero los datos de entrenamiento, que son los correspondientes a la matriz binaria del objeto RAM. De esta forma el programa sabría reconocer a uno de los 5 individuos a partir de una foto de éste sin necesidad de introducirle los ficheros de entrenamiento. El archivo tiene el nombre "**RAMs.dat"** y ocupa una gran cantidad de espacio: unos 8 megas para nuestra aplicación de reconocimiento de 5 personas con imágenes de 800*600 píxeles.

Código

A continuación se muestra el código Java correspondiente a la implementación de la práctica:

```
   import java.io.*;

// Clase principal que contiene los metodos principales del programa incluyendo el
metodo
// main. Contiene tambien los valores finales que se usan en todo el programa, entre
ellos
// el umbral por defecto calculado experimentalente para que el reconocimiento de el
mejor
// resultado, asi como el numero de tuplas y el tamaÃƒÂ±o de la imagen.
    class Neural {

      final static int numTuple = 4;__// numero de tuplas escogido
      static int numPatterns;  // numero de patrones de entrenamiento
      static int samplesPerPattern; //numero de imagenes de cada persona
      final static int row1 = 80;  final static int col1 = 250; // variables que
especifican
      final static int row2 = 480; final static int col2 = 550; //el marco de la cara
      final static int rowsPictureBMP = 600; // tamaÃƒÂ±o vertical de la foto
      final static int colsPictureBMP = 800; // tamaÃƒÂ±o horizontal de la foto
      static String names_vector[];  // nombre de las personas

   _
      int threshold = 20;  // valor por defecto
      RAM RAMs[]; __ // array de objetos RAM

       public static void main(String args[]) throws Exception {
         System.out.println();

System.out.println("**********************************************************************
***********");
         System.out.println("****************** Redes Neuronales: Reconocimiento de
caras *****************");

System.out.println("**********************************************************************
***********");
         System.out.println();
         System.out.println("1) Reconocer cara usando datos grabados en archivo");
         System.out.println("2) Reconocer cara entrenando la red");
         System.out.println("3) Reconocer cara entrenando y almacenar RAMs de
entrenamiento en archivo");
         System.out.println();
         System.out.print("Escoja su opcion: ");
         char option = (char) System.in.read();
         System.out.print("Introduzca el nombre del fichero (.bmp) que quiere reconocer:
");
         System.in.read();  // leer y rechazar el caracter '\n'
         System.in.read();  // leer y rechazar el caracter '\r'
         BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
         String name = input.readLine();
         if (!(new File(name)).exists()) {
            System.out.println("********* ERROR: Fichero no encontrado ********");
            System.exit(0);
         }

         Neural neural = new Neural();__
         neural.run(option, name);
      }

   _
   // Metodo que es llamado por el main() utilizado simplemente para facilitar la
programacion.
       public void run(int option, String file)
      {
         RAM.numRAMs = (row2-row1)*(col2-col1)/numTuple;
         RAMs = new RAM[RAM.numRAMs];
         switch (option) {
            case '1':
               readNames();
               readRAMsFromFile();
```

```
               break;
            case '2':
               trainNetwork();
               break;
            case '3':
               trainNetwork();
               saveRAMsIntoFile();
               break;
            default:
               System.out.println("ERROR: opcion no reconocida");
         }
         Pattern test = new Pattern(new File(file), threshold);__
         test.selectPixelsOfTheFace(row1,col1,row2,col2);__

         test.testPixels();
         test.testBoolPixels();

         System.out.println('\n' + "La persona reconocida es: " +
namePerson(recognizePattern(test)));
      }

   // Metodo que lee los ficheros de entrenamiento y crea los objetos Pattern
correspondientes
   // aplicando el marco en la cara de la fotografia. A partir de ahi llama al metodo
train()
   // de la clase RAM que es el que entrena la red, es decir, las matrices RAM.
       public void trainNetwork() {
         int rows = row2 - row1;
         int cols = col2 - col1;

         System.out.println("Leyendo patrones");
         try{
            DataInputStream bmps = new DataInputStream(new FileInputStream(new
File("bmps.txt")));
            int tempIndex=0;

            String line="";
            numPatterns=(new Integer(bmps.readLine())).intValue();
            System.out.println("Numero total de archivos: "+numPatterns);
            samplesPerPattern=(new Integer(bmps.readLine())).intValue();
            System.out.println("Numero de archivos por persona: "+samplesPerPattern);
            names_vector=new String[numPatterns/samplesPerPattern];
            Pattern p[] = new Pattern[numPatterns];

            for(int i=0; i<numPatterns; i++){
               line=bmps.readLine();
               System.out.println("Leido archivo: "+line);
               p[i] = new Pattern(new File(line), threshold);
            }


         _
            for (int i=0; i<15; i++)
               p[i].selectPixelsOfTheFace(row1,col1,row2,col2);


         ______
            System.out.println("Entrenando RAMs");
            _________
            for (int i=0; i<RAMs.length; i++) {
               tempIndex=0;
               RAMs[i] = new RAM(numTuple,numPatterns);
               RAMs[i].selectRAMSamplesInPattern(i, rows, cols);
               for (int j=0; j<numPatterns; j++){
                  if((j%samplesPerPattern)==0){
                     tempIndex++;
                  }
                  RAMs[i].train(p[j],tempIndex);
                  _
               }

            }


            bmps.close();
         }
             catch( Exception e){
               System.out.println("Error leyendo bmps.txt");
```

```
            }__
         readNames();

      }
      // Metodo que guarda en un vectos los nombre de las personas con las que se
entrena la red neuronal
       public void readNames(){
         try{
            DataInputStream bmps = new DataInputStream(new FileInputStream(new
File("bmps.txt")));
            numPatterns=(new Integer(bmps.readLine())).intValue();
            System.out.println("Numero total de archivos: "+numPatterns);
            samplesPerPattern=(new Integer(bmps.readLine())).intValue();
            System.out.println("Numero de archivos por persona: "+samplesPerPattern);
            names_vector=new String[numPatterns/samplesPerPattern];
            DataInputStream names = new DataInputStream(new FileInputStream(new
File("names.txt")));
         _
            for(int i=0;i<(numPatterns/samplesPerPattern);i++){
               names_vector[i]=names.readLine();
               System.out.println("Leido nombre:"+names_vector[i]);
            }
            names.close();
         }
             catch(Exception e){
               System.out.println("ERROR leyendo names.txt");
            }
      }

   _
   // Metodo que almacena en un archivo los datos de entrenamiento de la red, es decir,
   // las matrices RAM, para que no sea necesario disponer de las imagens de
entrenamiento
   // para reconocer a una persona de entre un grupo de 5.
       public void saveRAMsIntoFile() {
         try {
            File f = new File("RAMs.dat");__
            DataOutputStream out = new DataOutputStream(new FileOutputStream(f));
            System.out.println("Escribiendo RAMs");
            for (int i=0; i<RAMs.length; i++) {
               for (int j=0; j<RAM.numPatterns; j++)
                  for (int k=0; k<Neural.power(2,numTuple); k++)
                     out.writeBoolean(RAMs[i].bits[k][j]);

            }
            System.out.println("RAMs escritas");
         }
             catch(Exception e) { System.out.print("ERROR en saveRAMsIntoFile "); }
      }

   __
   // Metodo que lee de un archivo los datos de entrenamiento de la red, es decir, las
   // matrices RAM almacenadas. De esta forma no es necesario disponer de las imagenes.
   // El bucle que inicializa los objetos RAM es el que requiere mas tiempo de ejecucion
   // de todo el programa.
       public void readRAMsFromFile() {
         System.out.println("Inicializando RAMs");
         for (int i=0; i<RAMs.length; i++) {__
            RAMs[i] = new RAM(numTuple,numPatterns);__
            RAMs[i].selectRAMSamplesInPattern(i, row2-row1, col2-col1);_
         }
         System.out.println(RAMs.length + " RAMs inicializadas");
         try {
            File f = new File("RAMs.dat");
            DataInputStream in = new DataInputStream(new FileInputStream(f));
            System.out.println("Leyendo RAMs del archivo");
            for (int i=0; i<RAMs.length; i++) {
               for (int j=0; j<RAM.numPatterns; j++)
                  for (int k=0; k<Neural.power(2,numTuple); k++)
                     RAMs[i].bits[k][j] = in.readBoolean();______
            }
         }
             catch(Exception e) { System.out.print("ERROR en readRAMsIntoFile "); }
      }

   // Metodo que realiza el reconocimiento del patron introducido como parametro a
partir
   // de los datos de todas las matrices RAM almacenados en el array RAMs[]. Devuelve el
   // patron que tiene mas lecturas positivas o casillas a 1 de las memorias.
```

```
       public int recognizePattern(Pattern p) {
         int sums[] = new int[RAM.numPatterns];__
         for (int i=0; i<RAM.numRAMs; i++) {___
            for (int j=0; j<RAM.numPatterns; j++) {_________
               int k = selectMemoryPosition(p.boolPixels, RAMs[i]);____
            //System.out.println("Recognizing patterns " + i + " (" + RAM.numRAMs +
")"); //+ " k=" + k);
               if (RAMs[i].bits[k][j])
                  sums[j]++;
            }_    ____________
         }
         return maxNumber(sums);
      }

   // Seleccionar la posicion horizontal correspondientes a los pixeles conectados a la
   // RAM seleccionada.
       private int selectMemoryPosition(boolean bPixels[][], RAM ram)  {
         int aux = 0;
         for (int i=0; i<RAM.numTuple; i++) {
         //System.out.println("POS X: " + ram.posX[i] + " POS Y: " + ram.posY[i]);_
            if (bPixels[ram.posX[i]][ram.posY[i]]) ___ _
               aux = aux + power(2,i);___
         }_
         return aux;
      }

   // Metodo que determina cual de los patrones tiene el numero maximo de lecturas
positivas
   // (con valor'1') de las matrices RAMs. Esto determina cual es la persona que tiene
mas
   // posibilidades de ser la que aparece en la fotografia introducida en el programa.
       private int maxNumber(int num[]) {
         int aux = 0;
         int max = -1;
         for (int i=0; i<num.length; i++) {
            if (num[i]>aux) {
               aux = num[i];
               max = i;
            }
            if (num[i] != 0)
               System.out.println(names_vector[i] + " = " + num[i]);
         }__
         return max + 1;
      }

   // Metodo simple que realiza la potencia de dos numeros: a^b__
       static int power(int a, int b) {
         int aux = 1;
         for (int i=0; i<b; i++)
            aux = aux*a;_______
         return aux;
      }

   // Metodo simple que identifica la persona a partir de su numero identificativo.
       private String namePerson(int n) {
         String out="";
         switch (n) {
            case 0:
               out="Esta persona no es ninguna del grupo";
               break;
            default:
               out=names_vector[n-1];
         }
         return out;
      }
   }

    class RAM {

      static long size;__// tamaÃƒÂ±o de la matriz bits[][]
      static int numTuple;_// numero n de la n-tupla
      static int numPatterns;_
      boolean bits[][];  // valor de los bits de cada posicion de la matriz
      int posX[]; _// Filas. _ Posicion de la conexion: RAM <==> patron
      int posY[];__// Columnas. Posicion de la conexion: RAM <==> patron
      static int numRAMs;
```

```
        public RAM(int nt, int np) {
          numTuple = nt;
          numPatterns = np;
          size = Neural.power(2, nt) * np;__
          posX = new int[nt];
          posY = new int[nt];
          bits = new boolean[Neural.power(2,nt)][np];__
        }

   // Metodo que selecciona la posicion de las distintas conexiones que corresponde a
cada
   // "casilla" de la matriz RAM. Esta casilla es la posicion (i,j) en bits[i][j].  Esta
   // tarea es realizada independientemente del tamaÃƒÂ±o de la matriz, lo que da
flexibilidad
   // a que el tamaÃƒÂ±o del marco de la cara pueda ser distinto.
   // 'n' es el identificador de esta RAM
   // 'rows' y 'columns' son las filas y columnas del patron__
        public void selectRAMSamplesInPattern(int n, int rows, int columns) {
          int aux = 0;
          int cont = 0;

          for (int i=0; i<rows; i++) {
             for (int j=0; j<columns; j++) {
                if ((cont == RAM.numRAMs - n) & (aux < Neural.numTuple)) {
                   posX[aux] = i;
                   posY[aux] = j;
                   aux++;
                   cont = 0;
                }
                cont++;
             }
          }__
        }

   // Metodo mas importante de todos. Entrena la RAM a partir de un patron recibido como
   // parametro. Este metodo se puede y debe ser utilizado varias veces con distintos
patrones
   // de una misma persona para ir llenando de unos '1' distintas posiciones de la
matriz. Este
   // proceso, sin embargo, no se debe ejecutar excesivamente ya que se puede saturar la
matriz
   // impidiendo que la red sea capaz de generalizar.
        public void train(Pattern f, int n) {
          int num = 0;__
          boolean aux[] = new boolean[numTuple]; // binary number depending on the read
value
          for (int i=0; i<numTuple; i++)
             aux[i] = f.boolPixels[posX[i]][posY[i]];

        _
        // Transformar el numero binario en decimal
          for (int i=0; i<numTuple; i++) {
             if (aux[i])
                num = num + Neural.power(2,i);
          }__
        //System.out.println(" length=" + bits.length + " num=" + num + " n=" + n);__
          bits[num][n-1] = true;_// true equivale a '1' binario_
        }

   }

// Clase que representa los patrones de una imagen, tanto en tono de grises con un byte
// por pixel, como su correspondientes binarios al aplicar el valor umbral. Ambos
conjuntos
// de pixeles estan representados por un array bidimensional cada uno.
    class Pattern {

      int rows;____// numero de filas
      int columns;  ___// numero de columnas
      int pixels[][];___// valor de los pixeles de 8 bits
      boolean boolPixels[][]; // valor de los pixeles de 1 bit

   // El metodo constructor lee el archivo a partir del cual se rellena pixels[][]
   // Despues realiza el filtro umbral a partir del cual se rellena boolPixels[][]
        public Pattern(File f, int threshold) {_
          rows = Neural.rowsPictureBMP;_
          columns = Neural.colsPictureBMP;
          pixels = new int[rows][columns];__
```

```
         boolPixels = new boolean[rows][columns];

         loadDataFromFile(f);__
         thresholdFilter(threshold);
      }

   // Metodo que crea un fichero con los datos que contiene array pixels. Ha de verse
con
   // el programa notepad escogiendo un tamaÃƒÂ±o de letra pequeÃƒÂ±o.
       public void testPixels() {__
         File f = new File("test1.BMP");
         int cont = 0;
         try {
            DataOutputStream out = new DataOutputStream(new FileOutputStream(f));
            for (int i=rows-1; i>0; i--) {
               for (int j=0; j<columns; j++) {
                  out.write(pixels[i][j]);
                  cont++;
               }
               out.write('\r');
               out.write('\n');
            }
            System.out.println("Datos almacenados en el fichero 'test1.BMP' " + cont + "
pixels");
         }
             catch (Exception e) { System.out.println("Excepcion en textPixels() " + e);
}
      }

   // Metodo que crea un fichero con los datos que contiene array de pixeles binarios
   // boolPixels. Ha de verse con el programa notepad escogiendo un tamaÃƒÂ±o de letra
   // pequeÃƒÂ±o. El pixel negro esta representado como '1' y el blanco como '0'.
       public void testBoolPixels() {
         File f = new File("test2.BMP");
         int cont = 0;
         try {
            DataOutputStream out = new DataOutputStream(new FileOutputStream(f));
            for (int i=rows-1; i>0; i--) {
               for (int j=0; j<columns; j++) {
                  if (boolPixels[i][j])
                     out.write('1');
                  else out.write('0');
                  cont++;        ___
               }
               out.write('\r');
               out.write('\n');
            }
            System.out.println("Datos almacenados en el fichero 'test2.BMP' " + cont + "
pixels");         __
         }
             catch (Exception e) { System.out.println("Excepcion en textBoolPixels() " +
e); }
      }

   // Metodo que enmarca la parte de la imagen donde se encuentran las caras
aproximadamente.
   // Esos pixeles seran los que se utilicen para entrenar la red, ya que son los
pixeles que
   // realmente queremos reconocer. Ademas no nos interesa reconocer la ropa que lleva
el
   // individuo, ya que la red se confundiria si dos personas llevaran la misma ropa.
       public void selectPixelsOfTheFace(int row1, int col1, int row2, int col2) {
         rows = row2 - row1;
         columns = col2 - col1;
         int newPixels[][] = new int[rows][columns];__
         boolean newBoolPixels[][] = new boolean[rows][columns];__
         for (int i=row1; i<row2; i++)
            for (int j=col1; j<col2; j++) {  ____
               newPixels[i-row1][j-col1] = pixels[i][j];
               newBoolPixels[i-row1][j-col1] = boolPixels[i][j];
            }
         pixels = newPixels;
         boolPixels = newBoolPixels;__
      }

   // Metodo que lee los ficheros de imagenes BMP y lo almacena en la variable pixels de
la clase._
```

```
       private void loadDataFromFile(File f) {_
         try {
            DataInputStream in = new DataInputStream(new FileInputStream(f));
         // Leer e ignorar los bytes de la cabecera del fichero BMP (1078)
            for (int i=0; i<1078; i++)
               in.readByte();
         // Leer y almacenar en el array los bits deseados
            for (int i=0; i<rows; i++)
               for (int j=0; j<columns; j++)
                  pixels[i][j] = in.readByte();        __ __
         }
             catch (Exception e) { System.out.println("Excepcion en loadDataFromFile() "
+ e); }
       }

   _
   // Metodo que transforma el fichero con pixeles de 256 niveles a pixeles de 2 niveles
       private void thresholdFilter(int threshold) {
         for (int i=0; i<rows; i++)
            for (int j=0; j<columns; j++) { ___
               if (pixels[i][j]>threshold)
                  boolPixels[i][j] = true;
               else
                  boolPixels[i][j] = false;
            }
       }
   }
```

Resultados experimentales

A continuación se muestran las salidas resultado de ejecutar el programa con varias personas de entrenamiento.

El sistema se entrena con 5 personas diferentes (elegidas entre Alfonso, Carlos, Chus, Dani, Jesús, Juan Ramón, Pablo, Paco) y con 3 imágenes por persona, y posteriormente se realizan pruebas de reconocimiento con dos imágenes nuevas de las personas con las que se ha llevado a cabo el entrenamiento y con otras que con las que no se ha entrenado y que por tanto no pertenecen a ningún grupo de clasificación.

Las imágenes empleadas son las siguientes, de arriba abajo y Alfonso, Carlos, Chus, Dani, Jesús, Juan Ramón, Pablo, Paco y de izquierda a derecha los archivos son numerados del 1 al 5 (los tres primeros serán para entrenamiento y los otros dos para test, aunque evidentemente puede cambiarse).

Los resultados numéricos son los siguientes.

Resultados con las personas que sí deberían ser reconocidas:

Entrada: alfonso4.bmp Salida: Dani = 16865 Alfonso = 17775 Pablo = 11972 Chus = 17104 Paco = 16783 La persona reconocida es: ALFONSO	Entrada: alfonso5.bmp Salida: Dani = 16833 Alfonso = 20207 Pablo = 12266 Chus = 17091 Paco = 14822 La persona reconocida es: ALFONSO
Entrada: chus4.bmp Salida: Dani = 12181 Alfonso = 16924 Pablo = 17607 Chus = 23419 Paco = 13575 La persona reconocida es: CHUS	Entrada: chus5.bmp Salida: Dani = 12181 Alfonso = 16924 Pablo = 17607 Chus = 23419 Paco = 13575 La persona reconocida es: CHUS
Entrada: paco4.bmp Salida: Dani = 12133 Alfonso = 10824 Pablo = 9174 Chus = 11194 Paco = 19814 La persona reconocida es: PACO	Entrada: paco5.bmp Salida: Dani = 9908 Alfonso = 9487 Pablo = 11388 Chus = 9689 Paco = 7985 La persona reconocida es: PABLO ERROR
Entrada: dani4.bmp Salida: Dani = 20297 Alfonso = 13687 Pablo = 12325 Chus = 10019 Paco = 14673 La persona reconocida es: DANI	Entrada: dani5.bmp Salida: Dani = 23375 Alfonso = 13881 Pablo = 13079 Chus = 11144 Paco = 10284 La persona reconocida es: DANI
Entrada: pablo4.bmp Salida: Dani = 9964 Alfonso = 10071 Pablo = 20013 Chus = 14133 Paco = 8485 La persona reconocida es: PABLO	Entrada: pablo5.bmp Salida: Dani = 9609 Alfonso = 10371 Pablo = 20866 Chus = 14980 Paco = 9524 persona reconocida es: PABLO
Entrada: juanra4.bmp Salida: Dani = 9648 Alfonso = 7908 Pablo = 6676 Chus = 6298 Juanra = 16269 La persona reconocida es: Juanra	Entrada: juanra5.bmp Salida: Dani = 9894 Alfonso = 8553 Pablo = 6546 Chus = 7171 Juanra = 21907 La persona reconocida es: Juanra
Entrada: jesus4.bmp Salida: Dani = 9899 Alfonso = 12262 Pablo = 11499 Chus = 13681 Jesus = 13779 La persona reconocida es: Jesus	Entrada: jesus5.bmp Salida: Dani = 10874 Alfonso = 12326 Pablo = 12458 Chus = 14217 Jesus = 17269 La persona reconocida es: Jesus
Entrada: carlos4.bmp Salida: Dani = 5094 Alfonso = 5755 Pablo = 5880 Chus = 7893 Carlos = 21394 La persona reconocida es: Carlos	Entrada: carlos5.bmp Salida: Dani = 4506 Alfonso = 6065 Pablo = 8768 Chus = 9133 Carlos = 16574 La persona reconocida es: Carlos

Observamos que en la gran mayoría de los casos la certeza es bastante alta constatando que el reconocimiento y la clasificación se realizan de forma bastante fiable.

Para el sujeto “Alfonso” las dos clasificaciones son correctas aunque el fichero “alfonso4.bmp” presenta unos valores que indican que aunque la clasificación es correcta se podría dudar de su validez, siendo mucho mejores los resultados obtenidos con “alfonso5.bmp”.

Analizando los resultados para “Chus” vemos que para ambos ficheros los resultados son claramente favorables, reconociendo con claridad al sujeto correcto ya que los valores numéricos que apoyan la elección del sujeto correcto son claramente mayores que el resto (incluso estando en algunas fotografías con las gafas puestas y en otras con ellas quitadas).

Sólo hay un caso en el que la clasificación es errónea y es con el fichero “paco5.bmp” que reconoce como “Pablo” aun siendo evidente que el parecido entre ambas personas es escaso. Si nos fijamos en los resultados obtenidos para el archivo “paco4.bmp” podemos ver que la clasificación es correcta, pero además se observa que la certeza con la que se realiza dicha decisión. Por esto, podemos atribuir el fallo con el fichero “paco5.bmp” al hecho de que al pasarla a blanco y negro resulte un patrón ambiguo, ya que todos los valores de respuesta a ese fichero son parecidos y ninguno es dominante (lo cual implica un grado de duda significativo).

Los resultados para “Dani” son también favorables, y ofrecen una fiabilidad muy alta ya que los resultados para el sujeto escogido como correcto dominan a los demás con claridad.

Con “Pablo” también obtenemos resultados correctos con una dominación clara de la elección del sujeto correcto sobre las otras posibilidades (certeza alta).

Verificando el funcionamiento con “Jesús” también obtenemos clasificaciones correctas, si bien con “jesus4.bmp” la decisión no es claramente dominante, siendo mucho más clara con “jesus5.bmp” incluso a pesar de que la cara aparece un poco girada (notar que el funcionamiento es correcto a pesar de que en este caso en algunas fotos el sujeto tiene gafas y en otras no).

Finalmente probamos con “Carlos” y nuevamente obtenemos clasificaciones correctas y con grados muy altos de certeza en ambos casos.

A continuación entrenaremos el sistema con imágenes correspondientes a cinco personas determinadas y tres imágenes de cada una de ellas, igual que en el caso anterior, pero ahora las imágenes de test no van a pertenecer a ninguna de las personas con las que se ha efectuado el entrenamiento (con lo que la clasificación no será correcta, evidentemente).

Los resultados con personas que no serán correctamente clasificadas son:

Entrada: alumno0.bmp **Salida:** **Dani = 16699** **Alfonso = 13948** **Pablo = 15475** **Chus = 13592** **Paco = 9700** **La persona reconocida es: DANI**	**Entrada: alumno1.bmp** **Salida:** **Dani = 16867** **Alfonso = 12785** **Pablo = 14590** **Chus = 12180** **Paco = 9999** **La persona reconocida es: DANI**
Entrada: alumno2.bmp **Salida:** **Dani = 9558** **Alfonso = 7893** **Pablo = 4593** **Chus = 6354** **Paco = 7191** **La persona reconocida es: DANI**	**Entrada: alumno3.bmp** **Salida:** **Dani = 7296** **Alfonso = 5002** **Pablo = 4832** **Chus = 5167** **Paco = 6784** **La persona reconocida es: DANI**
Entrada: alumno4.bmp **Salida:** **Dani = 9558** **Alfonso = 7893** **Pablo = 4593** **Chus = 6354** **Paco = 7191** **La persona reconocida es: DANI**	**Entrada: alumno5.bmp** **Salida:** **Dani = 9957** **Alfonso = 7012** **Pablo = 7456** **Chus = 8623** **Paco = 8257** **La persona reconocida es: DANI**

Observamos que curiosamente en todos los casos lo clasifica como si fuera “Dani”, aunque la diferencia en las salidas de todos los posibles candidatos es pequeña lo que demuestra las grandes “dudas” que existen al tomar la decisión (certeza muy baja). Esto nos hace pensar que el patrón en blanco y negro de la cara de “Dani” es el más “estándar” de todos ya que es el más parecido al patrón en blanco y negro de personas diferentes a las del entrenamiento.

Se podría implementar algún método en el código que ante valores de certeza por debajo de cierto límite concluyera que no se puede clasificar con una seguridad razonable, de forma que añadiríamos la posibilidad de “no clasificar” una imagen de test si las dudas son grandes.

Conclusiones

Como se ha podido comprobar, las redes neuronales en general son muy versátiles y permiten resolver problemas muy variados de forma eficiente.

El tipo particular de red neuronal que hemos empleado en nuestra implementación demuestra que se adapta perfectamente a la finalidad a la que la destinamos, el reconocimiento facial, ofreciendo resultados muy satisfactorios.

La estructura WISARD que hemos empleado tiene diversas ventajas sobre otras redes neuronales como por ejemplo:

- Es rápida, sobre todo considerando que está pensada para implementación hardware
- Es bastante sencilla de entender.
- El entrenamiento es rápido, y requiere muy pocos patrones para llevarlo acabo.
- Es más sencilla que otras estructuras de redes neuronales.

Pero como se ha comprobado presenta también ciertos inconvenientes:

- Requiere una gran cantidad de memoria cuando las imágenes son grandes, y el crecimiento de los requerimientos aumenta rápidamente.
- De forma nativa, solo puede tratar con imágenes binarias, de forma que en general es necesario hacer un preproceso de las imágenes, con la consiguiente pérdida de detalles.

A la vista de los resultados experimentales observamos que el número de aciertos es muy grande, ofreciendo resultados más que aceptables para un sistema tan sencillo, reconociendo imágenes pertenecientes a la misma persona incluso con cambios considerables en la expresión de la cara o con otras diferencias como llevar las gafas puestas o quitadas e incluso leves giros de cabeza o cambios de tamaño.

Bibliografía

- Transparencias de la asignatura “Inteligencia en Redes de Comunicaciones”.
- Apuntes de la asignatura “Tratamiento Digital de Señales”
- http://richardbowles.tripod.com/neural/neural.htm
- http://www.aaai.org/AITopics/html/neural.html
- “Neural Networks”, Phil Picton, Grassroots Series.
- “Pattern Recognition with N-Tuple Systems”, Simon Lucas, Computer Science Dept. (Essex University).