UNIVERSIDAD DE ORIENTE
NÚCLEO ANZOÁTEGUI
ESCUELA DE INGENIERÍA Y CIENCIAS APLICADAS
DEPARTAMENTO DE COMPUTACIÓN Y SISTEMAS

"Desarrollo de una aplicación para el reconocimiento facial mediante redes neuronales artificiales"

Realizado por:

Yubraska Karina Montaña Torres

Giuseppe Daniele Sifontes Lopardo

Trabajo de grado presentado ante la Universidad de Oriente como requisito parcial para optar al título de

INGENIERO EN COMPUTACIÓN

Barcelona, Julio de 2009

UNIVERSIDAD DE ORIENTE
NÚCLEO ANZOÁTEGUI
ESCUELA DE INGENIERÍA Y CIENCIAS APLICADAS
DEPARTAMENTO DE COMPUTACIÓN Y SISTEMAS

"Desarrollo de una aplicación para el reconocimiento facial mediante redes neuronales artificiales"

Asesor:

Prof. Msc. Bastardo José Luis
(Asesor Académico)

Barcelona, Julio de 2009

UNIVERSIDAD DE ORIENTE
NÚCLEO ANZOÁTEGUI
ESCUELA DE INGENIERÍA Y CIENCIAS APLICADAS
DEPARTAMENTO DE COMPUTACIÓN Y SISTEMAS

"Desarrollo de una aplicación para el reconocimiento facial mediante redes neuronales artificiales"

Jurado Calificador

_____

Prof. Msc. Bastardo José Luis
(Asesor Académico)

_____ _____

Prof. Ing. Claudio Cortínez
(Jurado)

Prof. Ing. Luis Felipe Rojas
(Jurado)

Barcelona, Julio de 2009

RESOLUCIÓN

De acuerdo en el artículo 44 del Reglamento de Trabajos de Grado:

"Los Trabajos de Grado son de exclusiva propiedad de la Universidad y sólo podrán ser utilizados para otros fines con el consentimiento del Concejo de Núcleo respectivo, quien lo participará al Consejo Universitario."

DEDICATORIA

A mi padre Jehová, a mi señor Jesús y al Espíritu Santo que mora en mí.

A mis padres Yajaira Torres y Leonicio Montaña

A mi hermanito Leo.

A mi amada familia.

Yubraska K. Montaña T.

DEDICATORIA

A mi Padre celestial.

A Jesús mi señor y salvador.

Al Espíritu Santo.

A mis padres, Giuseppina Lopardo y Daniel Sifontes.

Giuseppe D. Sifontes L.

AGRADECIMIENTOS

A Dios, por su misericordia infinita, por su amistad y por darme valor, fuerzas, sabiduría, y apoyo a lo largo de esta etapa de mi vida.

A mis padres por su ayuda incondicional y sus sabios consejos.

A mi hermano por su gran apoyo y por ser mi amigo.

A toda mi linda familia por ser tan atentos conmigo.

A mi compañero de tesis y amigo, Giuseppe, por su gran paciencia durante el desarrollo de este trabajo.

A nuestro asesor Msc. José Luis Bastardo, por su asesoría y dedicación durante la realización del proyecto, así como en la revisión y redacción del documento.

Al profesor Claudio Cortinez, por su disposición y ayuda durante la revisión de este documento.

A mis amigas Giselle, Gilliam y Adriana, y a mis amigos Fernando y Arturo, por brindarme amistad y apoyo desde el primer semestre.

A todos las personas que de alguna u otra manera influyeron durante el camino hacia el cumplimiento de esta meta.

Yubraska K. Montaña T.

AGRADECIMIENTOS

Le doy muchas gracias a Dios por darme las fuerzas suficientes para finalizar esta tesis.

A Jesucristo por guiar mis ideas.

Al Espíritu Santo por estar siempre conmigo.

A mis padres por apoyarme en mis estudios y por ser un ejemplo a seguir.

A mis hermanos por ayudarme en todo cuanto he necesitado.

A mi fiel compañera de tesis y amiga Yubraska.

A nuestro asesor por su gran ayuda y dedicación.

Y por último, y no menos importante, a toda la congregación “Los Valientes de David”.

Giuseppe D. Sifontes L.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ECUACIONES

RESUMEN

En este trabajo se ha desarrollado una aplicación para el reconocimiento de imágenes faciales basadas en niveles de gris mediante el uso de redes neuronales artificiales. A través de esta herramienta el usuario podrá indicar los rostros que la aplicación debe memorizar durante el aprendizaje, también podrá seleccionar algunos parámetros de las redes neuronales, como son: el número de capas y el número de neuronas empleadas por cada capa y el factor de aprendizaje, una vez configurados todos estos parámetros se podrá dar inicio al entrenamiento de las redes neuronales perceptrón multicapa y función de base radial, una vez finalizado el entrenamiento se genera un archivo con todos los pesos, umbrales y demás resultados obtenidos durante el entrenamiento, necesarios para que las redes logren el reconocimiento, este archivo es denominado memoria y se actualiza cada vez que se encuentra una mejor solución a un problema, por lo tanto, se puede entrenar tantas veces como se desee en busca de una mejor solución. Una vez que las redes neuronales artificiales se encuentren entrenadas estas serán capaces de identificar o verificar con un alto grado de acierto los sujetos aprendidos. El análisis y diseño de la aplicación fue realizado utilizando el lenguaje gráfico UML (Unified Modeling Language). Para el desarrollo de esta aplicación se utilizó el lenguaje de cuarta generación Java 1.6 y la herramienta JCreator Pro V3.

CAPÍTULO 1

INTRODUCCIÓN

1.1. Planteamiento del problema

En los últimos años el reconocimiento facial en el campo de las Redes Neuronales Artificiales (RNA) ha sido motivo de estudio para muchos investigadores, algunos buscan mejorar o crear métodos que permitan alcanzar resultados mejores y otros sólo se encargan de implementar sistemas neuronales para realizar tareas específicas. El grado de precisión de un reconocedor facial, está estrechamente ligado a las técnicas utilizadas en su implementación, por lo que se requiere de mucho estudio para la refinación de éstas, así como también para la creación de herramientas o métodos que intervengan en el proceso.

Para el estudio en cuestión se desea desarrollar una aplicación para el reconocimiento facial, que sea capaz de memorizar un conjunto de rostros de individuos, donde cada uno dispondrá de un número determinado de imágenes en diferentes perspectivas. Esta aplicación estará basada en técnicas con un buen rendimiento que permitan escoger entre dos formas de funcionamiento, es decir, que puedan realizar la tarea de identificación o verificación de individuos, como se describe a continuación:

1. Identificación: la aplicación debe ser capaz de identificar si un rostro corresponde a algún otro rostro conocido, es decir, entre el conjunto de rostros conocidos la aplicación debe determinar cual se asemeja más al presentado como entrada.

2. Verificación: la aplicación verifica si un rostro pertenece a un individuo dado, por lo tanto, esta tendría solo dos respuestas válidas, pertenece y no pertenece.

La aplicación estará conformada por dos (2) fases para su funcionamiento total. La fase de aprendizaje y la fase de reconocimiento. La primera se encargará de todo lo concerniente a la formación de las bases necesarias para hacer posible el reconocimiento y la segunda es la parte que hará uso de los conocimientos obtenidos en la primera fase para proporcionar resultados para la identificación o verificación de un individuo.

Fase de aprendizaje

Esta constará de dos (2) etapas, preprocesamiento de la imagen y clasificación de la misma. Cabe destacar que aunque el preprocesamiento no está estrechamente ligado al aprendizaje, este juega un papel importante en la etapa de clasificación, facilitando, agilizando y optimizando el aprendizaje.

Preprocesamiento

Se refiere al tratamiento de la imagen antes de ser procesada por la aplicación, se dispone para ello de las siguientes técnicas:

1. Análisis de componentes principales (ACP): esta técnica se emplea para reducir las dimensiones de la imagen del rostro, obteniendo una transformación lineal con un nuevo sistema de coordenadas, que conserva la información más relevante.
2. Baja Resolución: esta técnica no es más que el cambio de resolución de la imagen a una dimensión menor.

Ambas técnicas tienen en común reducir las dimensiones de las imágenes de entrada, mejorando los tiempos de procesamiento para la clasificación.

Clasificación

Una vez preprocesadas las imágenes de los rostros, estas deberán ser clasificadas de acuerdo a un identificador, lo cual indica que cada imagen estará acompañada del nombre del individuo correspondiente. En este estudio para la clasificación sólo se tomarán en cuenta técnicas basadas en redes neuronales, con el fin de comparar el comportamiento de varias redes neuronales, en cuanto al porcentaje de aciertos, generalización de la solución y minimización del error, en otras palabras, se desea determinar cual de todas las redes estudiadas arrojará un mejor resultado.

El proceso de clasificación se basa en la presentación de cada una de las imágenes preprocesadas, con su respectivo identificador a la RNA en estudio, la red neuronal tendrá como entrada la imagen indicada y como salida un código asociado al identificador de la entrada, por lo tanto, para cada imagen se tendrá un nuevo identificador o código, que corresponderá a la salida de la red neuronal, el cual será utilizado en el proceso de verificación e identificación de los individuos.

Fase de reconocimiento

Esta fase ofrecerá dos (2) formas de reconocimiento facial basadas en redes neuronales, se muestran a continuación:

1. La identificación: este proceso es realizado por una red neuronal previamente entrenada, a la cual se le presenta el rostro que se desea identificar, de manera que ésta determine si existe similitud con algunos de los rostros aprendidos. La salida de la red neuronal será el identificador o código del individuo al cual pertenece el rostro de entrada, un código inválido significaría que el individuo no ha sido reconocido por la aplicación.
2. La verificación: al igual que en la identificación, en la verificación también se le presenta un rostro a la red neuronal y además, el identificador del

> individuo al cual pertenece el rostro, es decir, se le presenta tanto la entrada como la salida esperada a la red neuronal, se realiza el proceso de identificación para la obtención del código correspondiente al individuo presentado, luego se compara esta salida con la esperada, y se verifica su pertenencia.

Las redes neuronales se caracterizan por presentar una gran flexibilidad y receptividad ante los cambios que pueda presentar un problema con un alto grado de complejidad, por lo tanto, para este estudio se desarrollará una aplicación para el reconocimiento facial basada en técnicas dentro del marco de las redes neuronales artificiales, los algoritmos de aprendizaje de estas redes serán sometidos a ciertas modificaciones para ajustarse al ámbito del reconocimiento facial, sin alterar su estructura original, mejorando el proceso de aprendizaje haciendo uso de técnicas existentes, así como la fusión de algunas de ellas o la creación de nuevas técnicas.

Una aplicación para el reconocimiento facial, resultaría de gran ayuda, especialmente para aquellos campos donde la seguridad juega papel fundamental, facilitando el proceso de autenticación al evitar la necesidad de tener consigo algún tipo de documentación que lo identifique, bastando solo con observar una cámara fotográfica, por lo tanto, la aplicación que se desea desarrollar, a diferencia de otras, permitirá tanto la verificación como la identificación de sujetos, a partir de una o varias fotos, aun cuando éstas tengan baja resolución o se encuentren en mal estado. Los algoritmos para el preprocesamiento de las imágenes se emplearán de forma un poco diferente a lo habitual, es decir, de ser necesario se realizarán cambios en la forma de implementar el Análisis de Componentes Principales, para mejorar la extracción de características principales de las imágenes, y así las redes neuronales procesarán información menos redundante de manera más eficiente y en menor tiempo, garantizando de igual manera un alto nivel de reconocimiento.

La aplicación será capaz de identificar y verificar sujetos, para esto será necesario suministrar a la aplicación un conjunto de imágenes correspondientes a los rostros de los sujetos que se desean aprender, para su posterior reconocimiento, por cada sujeto se deberá suministrar una serie de rostros que muestren distintas expresiones y gestos faciales.

Todas las imágenes suministradas a la aplicación tendrán una resolución de mxn píxeles y estarán en escala de grises; el rostro del individuo deberá estar localizado, lo que significa que la mayor parte de la imagen estará ocupada por el rostro, despreciando ciertos aspectos como el cabello.

1.2. Objetivos

1.2.1. Objetivo general

Desarrollar una aplicación para el reconocimiento de imágenes faciales basadas en niveles de gris mediante el uso de redes neuronales artificiales.

1.2.2. Objetivos específicos

1. Identificar las redes neuronales que se utilizarán en el sistema.
2. Establecer la función de activación más adecuada para cada una de las redes neuronales.
3. Diseñar los algoritmos de aprendizaje de las redes neuronales.
4. Codificar los algoritmos de aprendizaje para un entrenamiento automático.
5. Encontrar los valores iniciales que optimicen el funcionamiento de las redes neuronales.
6. Comparar los sistemas neuronales estudiados para la verificación e identificación facial.
7. Medir los niveles de rendimiento del sistema.

CAPÍTULO 2

MARCO TEÓRICO

2.1. Antecedentes

Existen un gran número de problemas en ciencia e ingeniería que implican la extracción de información a partir de datos complejos e inciertos. Para muchos de estos problemas, las aproximaciones tradicionales resultan inadecuadas.

Durante muchas décadas se trató de solucionar estos problemas mediante la utilización de algoritmos. Los desalentadores resultados obtenidos en visión artificial, y otras áreas, hicieron que en la década de los 70 se percatasen de la imposibilidad de manejar todas las combinaciones posibles en situaciones reales, base de la estructura en algoritmo, debido al casi infinito número de perturbaciones que pueden existir.

Surgieron así dos formas de aproximarse al comportamiento biológico:

- **Sistemas expertos**, en los que la aproximación a la experiencia se realiza a través de:
  - Una lista de reglas memorizadas (basadas en diagramas de dependencias entre variables del sistema).
  - Capacidad para inferir conclusiones de esas reglas utilizables como solución a preguntas planteadas al sistema (consejos).

Son especialmente útiles para los conocimientos simbólicos o explícitos.

- **Redes neuronales**, se trata de reproducir el esquema neuronal, tanto en sus unidades de proceso (neuronas), como en sus conexiones.

En general se puede afirmar que los "sistemas expertos" han sido empleados con éxito en problemas que pueden ser estrictamente definidos mediantes reglas. Sin embargo, cuando estas reglas no son fácilmente formulables, el éxito de tales sistemas es muy limitado. Las soluciones a este tipo de situaciones requiere la aplicación de las redes neuronales que presentan un mejor rendimiento cuando hay incertidumbre o datos incompletos en el conocimiento empleado.

El origen de los estudios en redes neuronales artificiales estuvo en la búsqueda de analogías cerebrales a partir de los estudios fisiológicos iniciados a finales del siglo pasado. Por ello puede ser interesante un pequeño análisis biológico.

Los problemas donde el cerebro se nos muestra más eficaz se caracterizan por:

- Estar muy difusos (mal definidos)
- Requerir un laborioso procesado (por ejemplo en el campo de la visión el cerebro es capaz de resolver con tremenda rapidez entre el infinito número de imágenes tridimensionales proyectables sobre una imagen bidimensional mediante pistas, experiencias, etc.).

Esta especialización cerebral demuestra su capacidad de adaptación al medio ya que el mundo es incierto y borroso, y solo modelizable tras innumerables simplificaciones en las que se pierden mucha información.

El cerebro los resuelve utilizando un procesado paralelo masivo haciendo trabajar simultáneamente alrededor de 1 millón de neuronas en la resolución de los mismos, sin embargo, al sistema neuronal biológico le resulta casi imposible competir en velocidad con los computadores en todo aquello describible mediante una secuencia repetitiva.

El cerebro humano esta formado por unos cien mil millones de neuronas. Cada una de ellas consta de un cuerpo celular del que surgen 2 tipos de ramificaciones: el axón (una fibra principal) y las dendritas (varias ramas fibrosas). En total cada neurona presenta aproximadamente unas diez mil conexiones. Las señales de entrada son captadas tanto por las dendritas como por el cuerpo. Este las integra y proporciona la señal emitida por el axón. Entre este terminal y el siguiente conjunto de dendritas se produce la sinapsis (transmisión de la excitación). El axón, inicialmente no ramificado, se subdivide en su tramo final en unas mil conexiones con otras tantas dendritas, formando así complejos mapas de interrelaciones neuronales, y permitiendo el procesado paralelo **[1]**.

Las redes neuronales artificiales surgieron originalmente como una simulación del sistema nervioso, formado por un conjunto de unidades conectadas entre sí, simulando a las neuronas de los sistemas nerviosos biológicos.

Fue 1943 cuando Warren McCulloch y Walter Pitts propusieron un modelo matemático de neurona en el que se basa las redes neuronales actuales. Seis años después, en 1949, en su libro *The Organization of Behavior,* Donald Hebb presentaba su conocida regla de aprendizaje: si un elemento de procesado simple recibe una entrada de otro elemento de procesado y ambos están activos, el peso correspondiente a su conexión debería reforzarse.

En 1957, Frank Rossemblatt presentó el Perceptrón, una red neuronal con aprendizaje supervisado cuya regla de aprendizaje era una modificación de la propuesta por Hebb. La principal aportación del Perceptrón es que la adaptación del funcionamiento de la neurona se realiza teniendo en cuenta el error entre la salida real que se produce y la salida que debería dar.

Se crearon grandes expectativas sobre sus aplicaciones que, posteriormente, se tornaron en gran decepción cuando en 1969 M. Minsky y S. Papert demostraron las grandes limitaciones de esta red.

En los años 60 se proponen otros dos modelos, también supervisados, basados en el Perceptrón de F. Rossemblatt denominados Adaline y Madaline. En éstos, la adaptación de la red se realiza de forma análoga al Perceptrón, pero las reglas de aprendizaje son distintas, definiendo una función de error individual para cada neurona **[3]**.

2.2. Tratamiento de Datos

Para el tratamiento de los datos es necesario hacer uso de técnicas que permitan la manipulación de la información de una forma segura y eficiente, entre las más utilizadas en el área de la estadística se encuentran, el Análisis Factorial y el Análisis de Componentes Principales, estas son herramientas estadísticas que permiten reducir el número de variables. Aunque ambas técnicas se tratan en muchas ocasiones como si se tratase de una sola, conviene aclarar las diferencias entre ambas.

En el análisis de componentes principales, las nuevas variables o componentes principales (independientes entre sí) serán una combinación lineal de las variables originales (relacionadas) y un número relativamente pequeño de componentes explica la mayor parte de la varianza total de todas las variables originales. Los últimos factores o componentes, que explican menos, pueden ser eliminados con pérdidas mínimas de información.

En el análisis de componentes principales no partimos de una hipótesis previa, y es un método descriptivo que, básicamente, permite obtener una representación de nuestros casos en el nuevo espacio dimensional de nuestras nuevas variables o componentes principales.

El análisis factorial es uno de los métodos multivariantes más utilizados. El análisis factorial estudia fenómenos en los que las variables dependen de un factor común (implícito, no medible). Se utiliza entonces un modelo lineal que expresa las variables en función de los factores comunes, mediante coeficientes que se obtienen a partir de los coeficientes de correlación de las variables. El análisis factorial nos aporta unos factores que explican la variabilidad de las variables **[7]**.

2.2.1. Análisis de Componentes Principales (PCA)

El Análisis de Componentes Principales (PCA), desarrollado por Karl Pearson en 1901, es uno de los métodos pioneros del Análisis Multivariante. Sin embargo, debido al cálculo extremadamente complicado necesario cuando se consideran más de dos o tres variables, sólo se ha utilizado ampliamente con la generalización de la presencia de ordenadores en los laboratorios de investigación.

En la sencillez conceptual de este tipo de Análisis Multivariante reside su potencia para el tratamiento de datos compuestos por numerosas variables. Estos bloques de datos son generados por estudios sobre procesos o problemas en los que intervienen multitud de variables, que son registradas experimentalmente. En la mayoría de las ocasiones, la complejidad del problema que se pretende estudiar hace que algunas de las variables no se consideren en el estudio o, simplemente, que sean desconocidas. Ejemplos típicos de casos en los que es útil la aplicación de un PCA son estudios medioambientales (geológicos, biológicos, ecológicos…) en los que existen una gran multitud de factores que intervienen en el parámetro a estudiar. Asimismo, la facilidad con la que el científico es capaz de registrar una cantidad ingente de datos en la observación de cualquier proceso, tiene como consecuencia que la interpretación intuitiva de la información obtenida sea prácticamente imposible. También es importante extraer la mayor cantidad de información relevante contenida en los datos obtenidos, por lo que la aplicación del PCA en estos campos es cada vez más frecuente.

En este sentido, el PCA se puede considerar una técnica de reducción de la información. Conceptualmente, su objetivo es tomar *p* variables correlacionadas (X_1, X_2…, X_p), las cuales describen *n* objetos, y encontrar una combinación de éstas para generar otras variables nuevas (Z_1, Z_2…, Z_p) que no estén correlacionadas. La independencia de las nuevas variables Zi (llamadas *componentes principales*) hace que midan diferentes dimensiones de los datos. Estos componentes principales son ordenados de manera que Z_1 explique la mayor cantidad de variación contenida en los datos originales, Z_2 contiene la segunda mayor cantidad de información original, y así sucesivamente.

Cuanto más dependientes sean las variables, más varianza será explicada por los primeros componentes, y la información contenida en los datos será mejor explicada por un menor número de componentes. De esta manera, las *p* variables X originales se reducen a los primeros Z_i. Si las variables seleccionadas en el estudio no están correlacionadas, no es posible la reducción, ya que la varianza se reparte entre todos los Z_i. En casos en los que las variables presentan altas correlaciones negativas o positivas, los dos o tres primeros componentes explican la mayor parte de la información contenida en la totalidad de las variables consideradas.

Así pues, a partir de una matriz de datos, el PCA permite transformar un conjunto de variables intercorrelacionadas en otro conjunto de variables no correlacionadas denominadas factores o componentes los cuales, como se ha indicado anteriormente, son combinación lineal de las variables originales. El primer componente principal que se extrae es el que resume lo mejor posible la información contenida en la matriz de datos original, es decir, es el que aporta un máximo de la varianza residual resultante, siendo independiente del primero. Se generan tantos componentes como variables originales (*n*) existan en la matriz de datos, siendo todos ellos ortogonales entre si en un espacio R^n.

El proceso de aplicación de una PCA no consiste simplemente en introducir unos datos determinados y apretar la tecla del ordenador. Como se ha explicado anteriormente, un análisis estadístico y, concretamente, un Análisis Multivariante basa su validez en la correcta elección de variables y de la interpretación de los resultados obtenidos. Ambas cosas dependen del conocimiento del científico de su propio campo de investigación, ya que en este caso la investigación no deja de ser una herramienta en manos del científico experimental.

En la aplicación de un PCA, el primer paso consiste en la tipificación de las variables, ya que en la mayoría de los casos serán de magnitudes diferentes. La tipificación asegura que las diferentes magnitudes de las medidas obtenidas experimentalmente no influyan sobre el peso que posteriormente tendrán en el análisis. Si no se tipificaran, las variables con magnitudes más elevadas dominarían la tendencia del PCA.

Antes de llevar a cabo el PCA propiamente dicho es recomendable la exploración de las diferentes variables y el cálculo de la matriz de correlación o de covarianza, con el objeto de conocer las interrelaciones existentes entre las variables.

Los objetos de la matriz de datos (n) se pueden representar en un espacio R^n según el valor calculado para los objetos respecto a cada componente, que se utilizan como coordenadas. Estos valores de los objetos o individuos para cada componente se calculan a partir de los valores propios (eigenvalues) y los vectores propios (eigenvectors) que nos dan los coeficientes de cada variable en la combinación lineal generada, calculados a partir de la matriz de correlaciones o de covarianzas.

El valor del primer objeto con respecto al primer componente (Z_i) sería:

$$Z_1 = a_{11} \cdot X_1 + a_{12} \cdot X_2 + \ldots + a_{1n} \cdot X_n \qquad (2.1)$$

Donde los coeficientes *a* se corresponden con los autovectores y $X_1 \ldots X_n$ son las variables tipificadas para el primer objeto.

El cálculo de los valores $Z_1 \ldots Z_n$ permite representar los objetos en un espacio R^n definido por los *n* primeros componentes. Esto permite visualizar los objetos respecto a los primeros componentes. Puesto que la mayor parte de la varianza se debe explicar en los primeros componentes **[4]**.

Los pasos que se deben seguir en un PCA son los siguientes:

1. Normalizar la matriz de entrada X, haciendo que su media sea igual a cero.
2. Calcular la matriz de covarianza, a partir de la matriz normalizada.
3. Calcular los autovalores de la matriz de covarianza.
4. Calcular los autovectores asociados a cada autovalor.
5. Calcular la matriz de transformación W.
6. Calcular la matriz resultante reducida,

$$Y = W^T \cdot X \qquad (2.2)$$

Donde Y es la matriz resultante, W^T es la transpuesta de la matriz de transformación y X es la matriz de entrada.

La matriz resultante reducida es el resultado de aplicar PCA a un conjunto de datos, la dimensión de esta matriz varía de acuerdo al número de autovectores seleccionados, cabe destacar que los primeros autovectores representan gran parte de la varianza concentrada en los datos, por lo tanto, unos cuantos autovectores son suficientes para explicar la matriz inicial.

2.1.1.1. Matriz normalizada.

Esta matriz representa el conjunto de datos del cual se desea obtener las componentes principales.

Los datos de entradas deben estar organizados y en un formato adecuado, por ejemplo, para una muestra con *n* individuos, donde a cada uno se le han medido *m* variables, se tendría una matriz donde las filas representarían las variables y las columnas serían los individuos, por lo tanto, se tendría un vector columna por cada individuo y el conjunto de individuos formaría la matriz de entrada de datos, se recomienda que esta matriz se normalice respecto a la media, es decir, se calcula un vector columna que sea la media de todas las columnas y este se le resta a cada columna, obteniendo de esta manera una matriz normalizada.

2.1.1.2. Matriz de covarianza

En estadística la matriz de covarianza es una matriz que contiene la covarianza entre los elementos de un vector, esta es una herramienta muy útil en varios campos. A partir de ella se puede derivar una transformación lineal que puede encontrar una base óptima para representar los datos.

Hay varias características dignas de mención acerca de las matrices de varianza-covarianza.

- Estas matrices son cuadradas; el número de columnas es igual al de filas, para que el total de celdas para N valores sea igual a N^2.

- Las varianzas de los valores aparecen en la diagonal de la matriz, que son las celdas que están en la línea que va de la esquina superior a la esquina inferior derecha de la matriz.

- La matriz es simétrica; el número que aparece en la fila i de la columna j también aparece en la fila j de la columna i. Es decir, los elementos de las celdas que están sobre la diagonal también aparece en las celdas correspondientes debajo de la diagonal.

Para obtener esta matriz se multiplica la transpuesta de la matriz normalizada por la matriz normalizada, obteniendo una matriz de menor dimensión **[6]**.

2.1.1.3. Autovalores y autovectores

Los autovalores y autovectores de matrices constituyen herramientas de suma importancia para abordar el estudio de algunas técnicas de la estadística Multivariante, tales como el análisis de componentes principales, el análisis de correspondencias binarias, el análisis de correspondencias múltiples, el análisis discriminante, el análisis de correlación canónica y el análisis de factores.

La determinación de los autovectores y autovalores de una matriz cuadrada de componentes reales, consiste en encontrar escalares λ perteneciente al campo de los números reales o de los complejos, para los cuales el sistema de ecuaciones $AX = \lambda X$ tenga soluciones no triviales para el vector X.

Sea A una matriz cuadrada de orden *n* y de componentes reales. Se dice que el escalar λ perteneciente a un campo K es un autovalor, valor propio, o valor característico de la matriz, si existe un vector $X \neq K^n$, diferente del vector nulo, que satisfaga la ecuación $AX = \lambda X$.

Al vector X se le denomina autovector, vector propio, o vector característico de la matriz A, asociado con el autovalor λ.

Algunas implicaciones derivadas de las definiciones de autovalor y autovector son las siguientes:

- En primer lugar, la ecuación matricial $AX = \lambda X$ es no lineal, ya que involucra el producto de dos valores desconocidos λ y X. Sin embargo, si λ fuese conocido, estaríamos en presencia de un sistema de ecuaciones lineales.

- La ecuación que define a los autovalores y autovectores de una matriz A escrita en la forma:

$$(A - \lambda I)\, X = 0 \qquad (2.3)$$

indica que el vector X es el vector de incógnitas en un sistema de ecuaciones homogéneo.

- Todo autovector está asociado a uno y solamente un autovalor.
- Los autovectores son vectores que al ser pre multiplicados por la matriz asociada dan origen a los mismos vectores, salvo por un factor λ que lo contrae, dilata o invierte su dirección, de acuerdo con si $0 < \lambda < 1$, $\lambda > 1$ o $\lambda < 0$, respectivamente.
- Una matriz de orden *nxn* tiene *n* autovalores.

Los autovalores y autovectores se deben obtener de la matriz de covarianza, por lo tanto, para una matriz de dimensión *nxn* se tendrían *n* autovalores y sus correspondientes autovectores, obteniendo como resultado otra matriz de dimensión nxn, donde cada columna representa un autovector **[9]**.

2.1.1.3.1. Método de la Potencia

Este método es empleado para la obtención de los autovalores y autovectores de una matriz, para esto se multiplica la matriz por un vector inicial para obtener, de esta manera, una sucesión de vectores convergentes a un vector propio asociado al valor propio más grande. La sucesión es la siguiente:

$$t_k = At_{k-1}, es\ decir, t_k = A^k t_0 \qquad (2.4)$$

Tal como está escrita, el modelo de los elementos de esta sucesión tienden a cero o a infinito según como sea $\|A\|$. Entonces, es mejor normalizar el vector a cada iteración:

$$z_{k+1} = At_k \qquad (2.5)$$

$$t_{k+1} = \frac{z_{k+1}}{\|z_{k+1}\|} \qquad (2.6)$$

Para estudiar la convergencia de este método supongamos que la matriz A es diagonalizable y sus valores propios son:

$$|\lambda_1| = |\lambda_2| = \cdots = |\lambda_r| > |\lambda_{r+1}| \geq |\lambda_n| \qquad (2.7)$$

Con $\lambda_1 = \lambda_2 = \cdots = \lambda_r$, es decir: hay un valor propio de módulo máximo y de multiplicidad r. Denotaremos por x_j los vectores propios linealmente independientes asociados a los correspondientes valores propios. Entonces, expresando t_0 en función de estos vectores,

$$t_0 = \sum_{j=1}^{n} \alpha_k x_j \qquad (2.8)$$

Podemos escribir

$$t_k = \frac{A^k t_0}{\|A^k t_0\|} = \frac{1}{\|A^k t_0\|} \sum_{j=1}^{n} \alpha_j \lambda_j^k x_j = \frac{\lambda_1^k}{\|A^k t_0\|} \left[\sum_{j=1}^{r} \alpha_j x_j + \sum_{j=r+1}^{n} \alpha_j \left(\frac{\lambda_j}{\lambda_1}\right)^k x_j \right] \qquad (2.9)$$

Tomando límites es claro que:

$$\lim_{k \to \infty} t_k = \pm \frac{\sum_{j=1}^{r} \alpha_j x_j}{\left\| \sum_{j=1}^{r} \alpha_j x_j \right\|} \qquad (2.10)$$

Es decir, el vector t_k converge a una cierta combinación lineal de los vectores propios asociados a los valores propios dominantes **[2]**.

2.1.1.3.2. Métodos de Deflación

Un método de deflación consiste en, dada una matriz A, un valor propio λ y su vector propio asociado *v*, hallar una nueva matriz A' de dimensión inferior, tal que sus valores propios sean los mismos que los de la matriz inicial A, excepto, como es natural, el ya conocido λ, o bien se puedan calcular de manera sencilla.

Utilizando conjuntamente el método de la potencia y deflaciones sucesivas, podemos ir hallando todos los valores propios y vectores propios de una matriz **[2]**.

Supongamos que hemos usado el método de potenciación para determinar un valor propio λ_1 y un vector propio ϵ_1 de una matriz simétrica A, ahora consideramos la matriz:

$$\mathbf{A_1 = A - \lambda \frac{\epsilon_1 \epsilon_1^t}{\epsilon_1^t \epsilon_1}} \qquad (2.11)$$

Para cualquier vector propio ϵ_1 de *A*, A_i satisface:

$$A_i e_i = Ae_i - \lambda_i \frac{\epsilon_1 \epsilon_1^t}{\epsilon_1^t \epsilon_1} \qquad (2.12)$$

Cuando i = 1 tenemos por lo tanto:

$$A_i e_i = Ae_1 - \lambda_1 e_1 = 0 \qquad (2.13)$$

Para i > 1, puesto que los vectores propios de una matriz simétrica son ortogonales $(i.e. e_1^T e_i = 0)$

$$A_1 e_1 = A e_i = \lambda_i e_i \qquad (2.14)$$

Así los valores propios de A_1 son $\lambda_1, \ldots, \lambda_n, 0$, y el método de potenciación seleccionará el segundo más grande (en valor absoluto) valor propio λ_2.

Claramente este método se puede continuar haciendo:

$$A_2 = A_1 - \lambda_2 \frac{\epsilon_2 \epsilon_2^T}{\epsilon_2^T \epsilon_2}, \ldots, A_i = A_{i-1} - \lambda_i \frac{\epsilon_1 \epsilon_1^T}{\epsilon_1^T \epsilon_1} \qquad (2.15)$$

En cada caso cambiamos el más grande valor propio λ_i descubierto hasta el momento a cero, y por lo tanto el método de potenciación aplicado a A_i producirá el valor propio restante más grande λ_{i+1}. Así podremos generar los valores propios de A, uno a la vez en orden inverso según su valor absoluto.

2.1.1.4. Matriz de transformación

Esta matriz es la empleada para transformar tanto la dimensión como el valor de las entradas, para obtenerla se hace uso de los autovectores y de la matriz normalizada, para esto se multiplica la matriz normalizada por la matriz de autovectores. Es recomendable normalizar esta matriz, debido a que la multiplicación entre diferentes columnas debe ser igual a cero, ya que estas columnas forman vectores ortogonales entre sí. Para esto se calcula la norma de cada columna y se divide entre cada valor de las columnas.

2.1.1.5. Matriz resultante reducida

Para reducir las entradas se hace uso de la matriz Transformada, para esto se multiplica la transpuesta de la matriz transformada por la matriz de entrada y se obtiene una nueva matriz que representa las componentes principales de la matriz de datos inicial. Como se puede apreciar en la ecuación 2.2.

2.3. Redes neuronales Artificiales (RNA)

Las redes neuronales han constituido un centro focal de intensa actividad durante los últimos años, dando lugar a una maduración de las primitivas ideas desarrolladas en los años sesenta. El procesamiento de la información realizado por estos sistemas es un tipo alternativo al que utilizan los actuales sistemas informáticos, los cuales procesan secuencialmente instrucciones almacenadas en una memoria y manipulan datos de la misma memoria. En las redes neuronales, en cambio, el paradigma emula el modelo biológico de aprendizaje y computación, con una fuerte base de procesamiento en paralelo, y estando involucrados en cada unidad de cálculo el proceso y la memoria.

La principal cualidad de estos sistemas la constituye su adaptabilidad dinámica, esto es, su capacidad para variar de comportamiento en situaciones cambiantes. Para llegar a esto, utilizan técnicas como el aprendizaje, generalización o auto-organización. Están inspirados en el modelo de neurona biológica, usando unidades elementales de procesamiento que mimetizan algunas de las características de las neuronas biológicas.

El comportamiento global de una red determina su capacidad para ensayar hipótesis, detectar patrones estadísticos y regularidades o ajustar dinámicamente un modelo implícito implementado en la misma arquitectura. Este comportamiento va más allá de la suma de las potencialidades de las neuronas que la componen. El resultado es la emergencia de propiedades nuevas que pertenecen al sistema como un todo **[1]**.

2.3.1. Definiciones básicas de RNA

Aunque ya se ha realizado una descripción de las RNA, existen diversas definiciones que pueden ser aclaratorias:

- "Una red neuronal es un procesador distribuido y con estructura paralela que tiene una tendencia natural a almacenar conocimiento experimental, haciéndolo apto para su uso. Se parece al cerebro en dos cosas: el conocimiento es adquirido por la red a través de un proceso de aprendizaje y este conocimiento se almacena en los pesos sinápticos o conexiones entre neuronas".
- "Una red neuronal es un modelo computacional con un conjunto de propiedades específicas, como son la habilidad de adaptarse o aprender, generalizar u organizar la información, todo ello basado en un procesamiento eminentemente paralelo".
- "Los sistemas neuronales artificiales o redes neuronales son sistemas celulares físicos capaces de adquirir, almacenar y utilizar conocimiento extraído de la experiencia".

2.3.2. Breve perspectiva Histórica

Los años 40 supusieron los primeros inicios de las RNA. Esta época supuso el comienzo de la búsqueda de modelos alternativos a los computadores tradicionales basados en máquinas de Von Neumann, debido a la incapacidad de éstos de tolerar fallos, funcionar de manera adaptativa, procesar en paralelo, etc.

Los primeros intentos destacables fueron el de Mc Cullocch y Pitts (1943), que crearon el primer modelo simple de neurona en base a funciones lógicas y además introdujeron la idea de umbral de activación, y el de Hebb (1949) que propuso una ley simple de entrenamiento (regla de Hebb).

Potenciado por estas primeras investigaciones, se promovieron gran cantidad de estudios sobre el tema y surgieron los primeros grandes avances entrando en un período emergente y prometedor. Por un lado Rossemblatt (1959) propuso el Perceptrón junto con su correspondiente regla de entrenamiento, por otro lado Widrow y Hoff (1959) propusieron el *Adaline* con la regla de aprendizaje precursora del *Backpropagation* (*delta rule*) y que posteriormente se comercializó para eliminar los ecos de las líneas telefónicas. El gran interés por las RNA durante la década de los 60, llevó asociado un estudio más en profundidad del tema y el descubrimiento de las limitaciones formales de las redes. La aparición del estudio *Perceptrons* de Minsky y Paper en 1969 puso en evidencia la imposibilidad de las RNA de generalizar un algoritmo de entrenamiento para las redes multicapa. Este artículo supuso el comienzo de una época de abandono de esta aplicación.

El resurgimiento de las RNA llegó en la década de los 80, debido básicamente a la solución de los problemas planteados anteriormente, al desarrollo de nuevos algoritmos de entrenamiento y topologías de redes, a los avances técnicos que permitieron la posibilidad de implementar las RNA en hardware y sobre todo una importante inversión económica en este tipo de investigación.

Algunas contribuciones interesantes realizadas a partir de la década de los 80 en adelante son: la creación en 1982 de las redes de Hopfield, el desarrollo del algoritmo *Backpropagation* en 1986 por Rumelhart, Hinton y William que permite el entrenamiento de redes multicapas, la aportación de Kohonen en 1984 de los mapas auto-organizativos (que se usaron para el reconocimiento de voz, composición musical, etc.), los estudios desde el punto de vista matemático y biológico de Grossberg en un gran número de artículos y también la creación de la *International Neural Network Society INNS* que ha sido uno de los grandes promotores de las investigaciones en este campo.

2.3.3. Elementos básicos de procesado

Estos elementos básicos de procesado son los encargados de realizar las operaciones de la red neuronal. Se presentarán a continuación de manera ordenada dichos elementos constitutivos de las redes.

2.3.3.1. Sistema de patrones entrada-salida

Se llamará patrón de entrada de una RNA al conjunto de variables independientes de entrada que se han elegido para representar el sistema que se quiere modelar. En el caso de tratarse de datos sin evolución temporal, pertenecen a un instante concreto y se representan por un vector de dimensión R, siendo R el número de variables de entradas independientes. Un sistema de patrones de entradas será el conjunto de todos los patrones de entradas que formarán el grupo de datos disponibles para el entrenamiento, y que será representado como una matriz de dimensión Rxm, siendo m el número de datos.

Del mismo modo, un patrón de salida es el conjunto de variables de respuesta del sistema a modelar, correspondiente a un patrón de entrada. Un patrón de salida

tiene dimensión $Sx1$ (siendo S el número de variables de salida independientes que se quieren obtener) y se define el sistema de patrones de salida (de dimensión Sxm, donde m es el mismo número que para las entradas) al conjunto de todos los patrones de salida disponibles.

Este sistema de patrones entrada-salida se utilizará para el entrenamiento y la posterior validación de la red. El objetivo de estas dos fases será ajustar las funciones de las RNA de manera que al aplicarlas sobre los patrones de entrada se obtengan resultados lo más parecido posible a los patrones de salida Sxm.

2.3.3.2. Conexiones y pesos sinápticos

Las conexiones sinápticas son las vías o caminos de comunicación entre los diferentes elementos de procesado (neuronas) y entre éstos y las entradas/salidas del sistema. Las conexiones se establecen únicamente entre dos elementos y solo pueden transmitir información en un sentido. La mayoría de las veces estas conexiones también procesan la información que transmiten de un modo sencillo. Cada conexión lleva asociado un peso o ponderación, que en adelante denotaremos con w_{ji}, donde el índice i corresponde al número de neuronas de la que parte la conexión y j a la que llega.

Además de los pesos propiamente dichos, específicos para las conexiones que unen las neuronas, existen unos pesos adicionales a considerar llamados *sesgos, umbrales* o *bias* y que representaremos con b_j. Estos sesgos están asociados a unas conexiones con origen en una unidad ficticia de entrada y la neurona de llegada. La inclusión de este grado de libertad mejora la convergencia del ajuste de los pesos

durante el proceso de entrenamiento, ya que evita la restricción de que la solución pase por el origen.

2.3.3.3. Funciones de activación

Las neuronas biológicas pueden estar excitadas o no excitadas, con lo que poseen cierto "estado de activación". Las neuronas artificiales también tienen grados de activación, que pueden limitarse únicamente a dos, como en el caso de las biológicas, o variar entre un rango de valores (normalmente definidos entre [0, 1] o entre [-1, 1]). Este valor indica el estado de la neurona: inactiva (0 o -1), activa (1), o bien un estado intermedio entre estos límites que indica su grado de activación.

Para obtener la salida final de una neurona se debe aplicar a la entrada ponderada total una función de activación que calcula el valor de salida o estado de actividad de la neurona.

Las neuronas de la capa de salida suelen utilizar funciones de activación identidad (función lineal), mientras que para los demás elementos de procesado se suelen utilizar funciones no lineales. Estas no linealidades pueden ser modeladas con funciones de tipo escalón, cuya respuesta es más fiel a la respuesta biológica, o con sigmoides, cuyo comportamiento matemático es menos hostil.

Aunque existen un gran número de funciones de activación diferentes, las de uso más común son las funciones sigmoidales, ya sea en su forma binaria (o logarítmicas) o en su expresión bipolar (o tangente hiperbólica). Sus ecuaciones se muestran a continuación y sus representaciones gráficas se presentan en la figura 2.1.

$$\textit{Sigmoide Binaria: } f_1(x)\frac{1}{1+e^{-\gamma x}} \qquad (2.16)$$

$$\textit{Sigmoide Bipolar: } f_2(x)\frac{1-e^{-\gamma x}}{1+e^{-\gamma x}} \qquad (2.17)$$

El parámetro γ en las ecuaciones 2.16 y 2.17 es siempre positivo y controla la pendiente de la función sigmoidal. Los cambios en esta pendiente influyen de forma directa en la brusquedad de los cambios a la salida de la neurona: mientras que para valores pequeños de γ la variación es casi lineal, para valores elevados casi se obtienen respuestas tipo escalón, con lo que no hay casi valores intermedios entre las asíntotas de saturación. La diferencia entre las dos funciones sigmoidales es el rango de variación de cada una de sus salidas: mientras que la sigmoide binaria está comprendida entre [0, 1], la bipolar lo está entre [-1, 1].

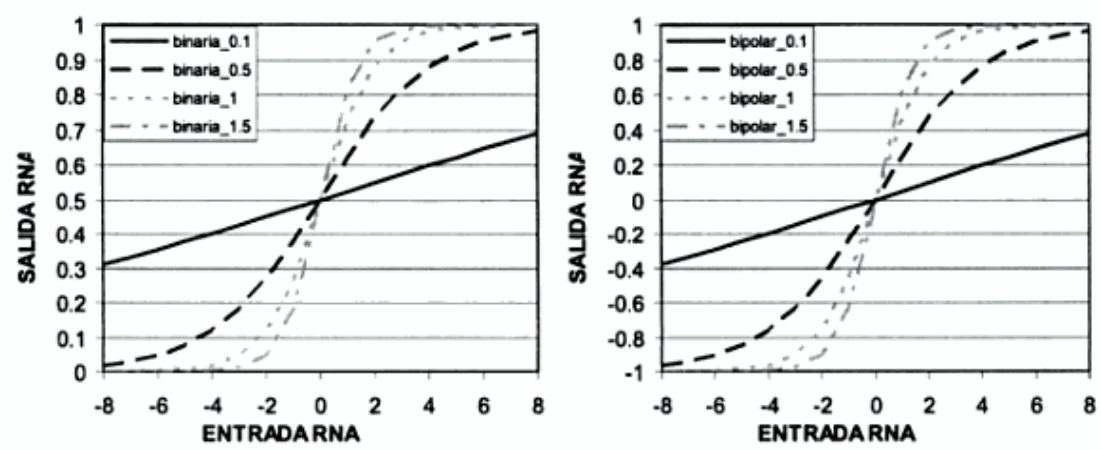

Figura 2.1. *Funciones de activación sigmoidales binarias (izquierda) y bipolar (derecha). Fuente:* **[8]**.

Un requisito que han de cumplir las funciones de activación para poder aplicar los algoritmos de entrenamiento tipo *Backpropagation*, es que sean derivables, condición que cumplen todas las funciones de activación descritas.

2.3.3.4. Elemento de procesado: neurona

La neurona es el elemento que procesa la información en la RNA y juega un papel fundamental en su funcionamiento. El modo de operación de las neuronas es el siguiente:

1. El conjunto de entradas que llegan a la neurona j (p_i), será multiplicado por cada uno de los pesos (w_{ji}), asociados a las conexiones.
2. Se realizará una suma de esta ponderación de entradas con los pesos, junto con el valor del sesgo de la neurona (b_j).
3. Una vez obtenida esta suma ponderada, se obtiene la salida final al aplicar una función de activación propia de la neurona.

La ecuación que describiría el funcionamiento de los pesos para una neurona sería:

$$a_j = f(w_{j1} \cdot p_1 + w_{j2} \cdot p_2 + \cdots + w_{jR} p_R + b_j) \qquad (2.18)$$

$$a_j = f(\sum_{i=1}^{R} (w_{ji} \cdot p_i) + b_j) \qquad (2.19)$$

Siendo p_i la entrada procedente de la neurona i, w_{ji} el peso que modifica la entrada procedente de la neurona i y que va hasta la neurona j, b_j el sesgo correspondiente a la neurona actual j y a_j la salida de la neurona j actual.

El modo de operación de una neurona artificial se describe de forma detallada en la figura 2.2:

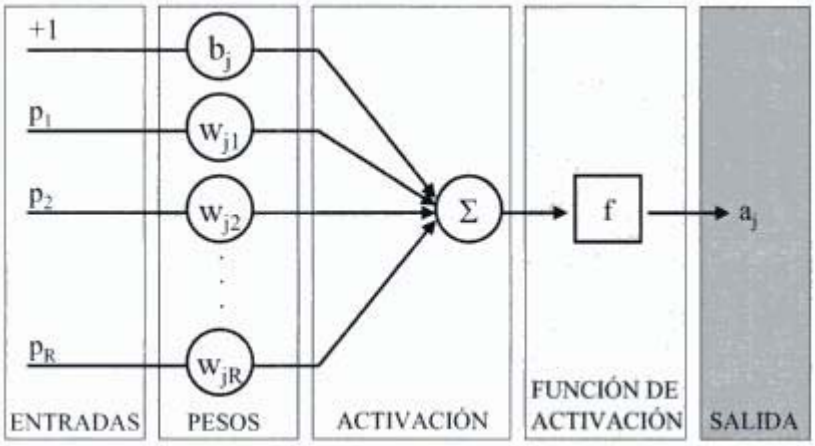

Figura 2.2. *Estructura y funcionamiento de una neurona artificial. Fuente:* **[8]**.

2.3.3.5. Asociación de neuronas

Normalmente las neuronas se organizan en distintos niveles de la red neuronal, que suelen denominarse capas. Estas capas se clasifican según su situación dentro de la red siguiendo el siguiente criterio:

De entrada: es siempre la primera capa de una RNA, y tiene tantas neuronas como entradas a la red. Cada neurona tiene una sola entrada que proviene directamente del exterior y varias salidas a neuronas de capas posteriores. A menudo, en el recuento de capas de una RNA no se incluye la de entrada, ya que sus neuronas se limitan a distribuir las entradas externas a las capas posteriores, pero no realizan ninguna operación "típica" de estos elementos de procesado.

Ocultas: Estas neuronas son las que realizan el trabajo y no tienen contacto directo con las señales externas a la red. Las neuronas de estas capas son las que mejor se ajustan a la definición de neuronas que se ha hecho en apartados anteriores. Puede haber varias capas de neuronas ocultas, desde cero hasta un número elevado. La primera de las capas ocultas estará conectada con la capa de entrada y la última de estas capas comunicará generalmente con la capa de salida. La forma de

interconexión entre las distintas capas ocultas junto con el número de neuronas en cada capa determinará la topología de la red.

De salida: es la capa de neuronas que transmite la información generada por la red al exterior. El número de neuronas de esta capa será igual al número de salidas de la RNA.

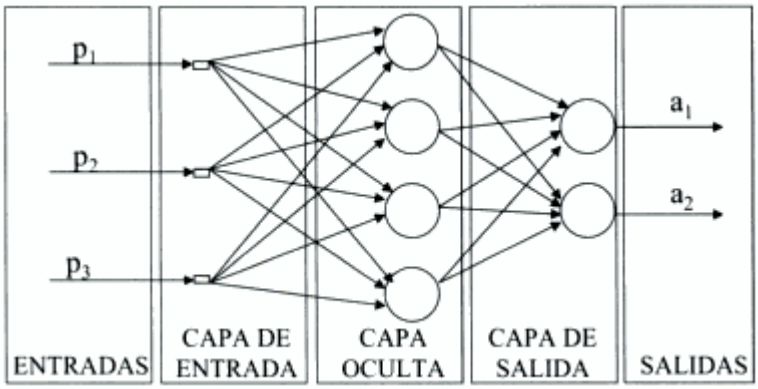

Figura 2.3. *Asociación de neuronas por capas. Fuente:* **[8]**.

La figura 2.3 muestra un ejemplo de RNA multicapa con 3 entradas, 2 salidas y una capa oculta de 4 neuronas. En este caso específico, cada capa está totalmente interconectada con la siguiente, es decir, la salida de una neurona de una capa anterior llega a la entrada de todas las neuronas de la capa siguiente.

La clasificación de neuronas en capas en un primer paso para introducir las diferentes topologías o arquitecturas que se pueden encontrar en las RNA, y que se describirán a continuación.

2.3.4. Arquitecturas de RNA

La organización de las neuronas y el tipo de conexiones que existen entre ellas es lo que se denomina arquitectura de las RNA. La elección de la arquitectura óptima para

cada aplicación específica a la que se quiera aplicar las RNA es un tema clave para encontrar una solución satisfactoria al problema a tratar.

Un posible esquema de la clasificación de las RNA en función de dichos parámetros aparece en la figura 2.4. Como se puede observar en la figura, los parámetros fundamentales que determinan la arquitectura de la RNA son el número de capas y la conectividad que se establece entre las neuronas. Tanto la nomenclatura de cada una de las RNA, como las diferentes subdivisiones dentro de estos dos criterios generales, se trata a continuación.

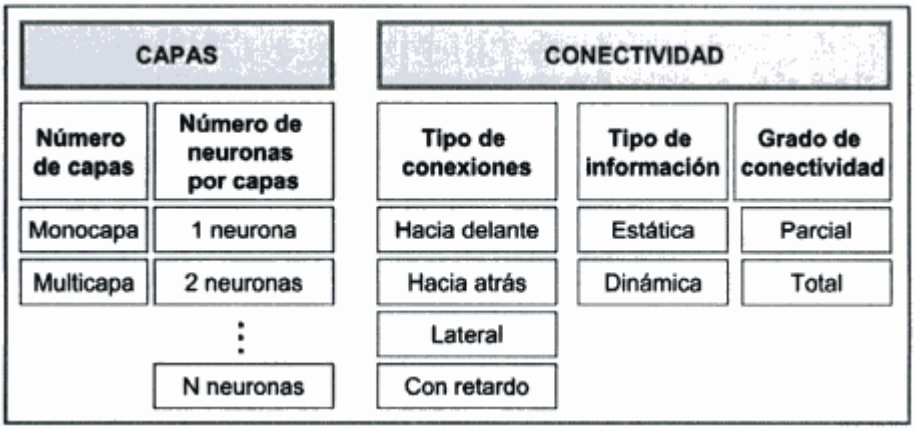

Figura 2.4. *Clasificación de las RNA. Fuente:* **[8]**.

2.3.4.1. Clasificación según número de capas

Existen dos tipos de clasificaciones de las RNA según su número de capas:

Redes monocapa. Las redes monocapa solo cuentan con una capa de neuronas de entrada y una de salida, que intercambian señales con el exterior y que constituyen a un tiempo capa oculta y capa de salida del sistema. En las arquitecturas típicas de este tipo de red, las neuronas de entrada están totalmente interconectadas con las de salida, pero también existen casos donde las conexiones entre las neuronas son más complicadas.

Las redes monocapa se utilizan típicamente en tareas relacionadas con lo que se conoce como auto-asociación; por ejemplo, para regenerar informaciones de entradas incompletas o distorsionadas que se presentan a la red, o para el reconocimiento de patrones. Por otra parte, los problemas que pueden ser resueltos por una red monocapa son bastante limitados, teniendo que recurrir en la mayoría de los casos a las redes multicapa. Ejemplos de redes monocapa son las redes de Hopfield o la máquina de Boltzmann.

Redes multicapa. Las redes multicapa están compuestas por al menos una capa oculta entre la capa de entrada y la de salida de la RNA.

Las redes multicapa pueden resolver problemas más complicados que las redes monocapas, a cambio eso sí, de tratar con superficies más complejas y un entrenamiento mucho más costoso y con más dificultadas para evitar mínimos locales. Normalmente la dificultad del problema a resolver va asociado con el tamaño de las redes multicapa, que está definido por el número de capas ocultas y de neuronas en cada una de ellas. Como ya se ha comentado, para obtener una mejor generalización se tiene que determinar apropiadamente el tamaño de las redes a usar en cada caso.

2.3.4.2. Clasificación según tipo de conexiones

La conectividad entre las neuronas canaliza la información de salida de una neurona como entrada de otra. Se presentan a continuación los diferentes tipos de conexiones entre las neuronas en una red neuronal:

- **Conexiones hacia delante:** el sentido de transmisión de la información en este tipo de conexiones se realiza desde neuronas de capas inferiores (más cercanas a la capa de entrada) hacia neuronas de capas superiores (más cercanas a la salida). También se denominan *feedforward.*
- **Conexiones hacia atrás:** estas conexiones llevan los datos de las neuronas de una capa superior, a otras de una inferior. Son también llamadas *feedbackwared*.
- **Conexiones laterales:** son las conexiones que se hacen entre neuronas de una misma capa. Estas conexiones se diseñan como excitadoras, permitiendo la cooperación (con peso positivo) o la inhibición (con peso negativo) entre neuronas, lo que establece una competición entre neuronas de la misma capa.
- **Conexiones con retardo:** los elementos de retardo se incorporan en las conexiones para implementar modelos dinámicos temporales (redes dinámicas). Se almacenan los datos antiguos para después procesarlos junto con la información actual.
- **Conexiones auto-recurrentes:** la salida de una neurona se conecta con su propia entrada.

Dependiendo del tipo de conexiones entre las neuronas, se establece la siguiente clasificación de arquitecturas:

Redes con conexiones hacia delante (*feedforward*). En este tipo de redes las señales neuronales se propagan siempre hacia delante a través de las capas de la red. No existen conexiones hacia atrás y normalmente tampoco auto-recurrentes ni

laterales. Ejemplos de redes *feedforward* son: Perceptrón multicapa o red de funciones de base radial.

Este tipo de redes tiene la ventaja de que la respuesta de la red es rápida y que además admiten algoritmos de entrenamiento del tipo *Backpropagation*.

Redes recurrentes. En este tipo de redes, la información circula tanto hacia delante (*feedforward*) como hacia atrás (*feedbackward*), por medio de conexiones que comunican con neuronas de capas anteriores. Las redes recurrentes están especialmente indicadas para los casos en los que la dinámica del proceso es importante, es decir, cuando los patrones de entrada-salidas son secuencias temporales de datos. A diferencia de las anteriores, la obtención de las respuestas es más lenta y su entrenamiento es más complicado y largo.

Algunos ejemplos de redes neuronales recurrentes son: Perceptrón multicapa recurrente, red de retardos temporales, red de Hopfield, mapa auto-organizativo de Kohonem, redes Elman, etc.

2.3.4.3. Clasificación según tipo de información

La última clasificación que se ha considerado es la del tipo de información que reciben las redes y se presenta a continuación:

Redes estáticas. El tipo de información que manejan las redes estáticas tiene carácter estacionario, es decir, el valor de la salida solo depende del valor de las entradas en el mismo momento en el que ésta se produce.

Estas redes se pueden caracterizar estructuralmente por la inexistencia de bucles de realimentación y de elementos de retardo entre los distintos elementos de proceso que las forman. Normalmente van asociadas a las redes con conexiones hacia delante (*feedforward*).

Redes dinámicas. La información que se maneja en este tipo de redes incluye la variable tiempo, es decir, la salida depende de las variables de entrada en el momento actual y de los valores de entrada y/o salida en momentos anteriores.

Estas redes suelen asociarse al tipo de redes recurrentes explicadas con anterioridad. Para incluir la variable tiempo, se pude incorporar en la red retardos temporales o conexiones de tipo *feedbackward* entre las capas de neuronas. El entrenamiento para las redes dinámicas puede plantear problemas de convergencia y estabilidad, y los resultados que generan son en general de difícil análisis.

2.3.5. Entrenamiento de RNA

Como se ha ido explicando en los apartados anteriores, los datos de entrada de la RNA se procesan mediante un conjunto de operaciones internas, que proporcionan una salida de la red. Las secciones anteriores se han centrado en caracterizar las redes desde el punto de vista de estructura y funcionamiento matemático interno. Desde el punto de vista de la aplicación de las RNA a un problema específico, existen dos fases: la fase de entrenamiento o aprendizaje y la fase de testeo o de prueba. En la fase de entrenamiento, se usa un conjunto de datos o patrones de entrenamiento para determinar las características que definen el modelo neuronal. Una vez entrenado este modelo, se pasa a la llamada fase de testeo, en la que se procesan nuevos patrones no presentados a la RNA hasta el momento y se analizan la capacidad de generalización de la misma.

Puesto que la topología de la red y las funciones de activación son características fijas de la RNA, los cambios que se producen durante el proceso de entrenamiento se reducen a la modificación de los pesos asociados a las conexiones entre las neuronas. Existen distintos tipos de criterios para modificar los pesos de las RNA, que se pueden agrupar en dos estrategias generales de aprendizaje: entrenamiento supervisado y no supervisado, y que se detallan a continuación:

2.3.5.1. Entrenamiento no supervisado

Tal y como se puede ver en la figura 2.5, en el entrenamiento no supervisado el conjunto de datos de entrenamiento solo está formado por los datos de entrada y no se dispone de los datos de salida de estos patrones. Se suele decir que la red es entrenada sin maestro, ya que ésta no recibe ninguna información del exterior que le indique si la respuesta generada es o no correcta. Las redes que usan este tipo de entrenamiento también se denominan auto-organizativas.

La red aprende a adaptarse basándose en las experiencias recogidas de los patrones de entrenamiento vistos con anterioridad, encontrando regularidades, correlaciones o categorías entre estos. En definitiva, la red modifica los pesos de forma que los vectores de entrada más similares sean asignados a la misma unidad de salida (o *cluster*), en lo que se conoce como mapeo de características. De ello se deduce que la aplicación fundamental de este tipo de entrenamiento es la clasificación o identificación de patrones de entrada. Ejemplos de redes con este tipo de entrenamiento son los mapas auto-organizativos de Kohonen.

Las principales clases de entrenamientos no supervisados son el aprendizaje de Hebb y el aprendizaje competitivo y cooperativo.

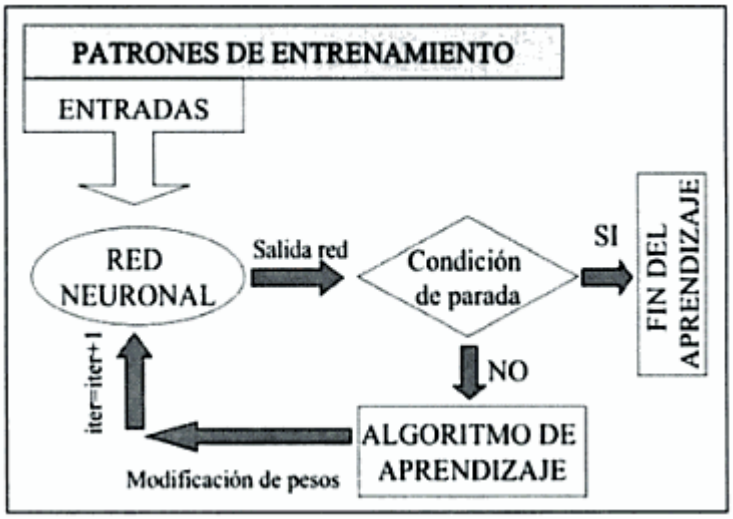

Figura 2.5. *Diagrama de flujo de entrenamiento no supervisado. Fuente:* **[8]**.

2.3.5.2. Entrenamiento supervisado

A diferencia del caso anterior, y tal y como se muestra en la figura 2.6, para este tipo de aprendizaje se dispone de unos patrones de entrenamiento compuestos por un conjunto de pares de elementos entrada-salida. En este caso, el proceso de aprendizaje se realiza mediante un entrenamiento controlado por un agente externo (supervisor, maestro) que determina la respuesta que debería generar la red a partir de una entrada determinada. En cada iteración, el supervisor comprueba la salida de la red y en el caso de que ésta no coincida con la deseada, se procede a modificar los pesos de las conexiones con el fin de conseguir que la salida obtenida se aproxime más a la deseada.

La red realizará varias iteraciones modificando los pesos de sus conexiones hasta alcanzar la condición de paro del entrenamiento, dejando a partir de estos momentos los pesos fijos. Estos pesos formarán el modelo de RNA que se usará en la fase de testeo que se ha comentado anteriormente.

Dentro de esta clasificación también existen diferentes formas de implementación:

Entrenamiento por refuerzo. Se aplica en casos en los que no se conoce o no se desea indicar exactamente las salidas deseadas para los patrones de entrada. Se basa en técnicas de premio-castigo, ya que se generan señales de refuerzo o inhibición mediante mecanismos de probabilidades que valorarán la respuesta obtenida con valores positivos (+1, premio) o con valores negativos (-1, castigo).

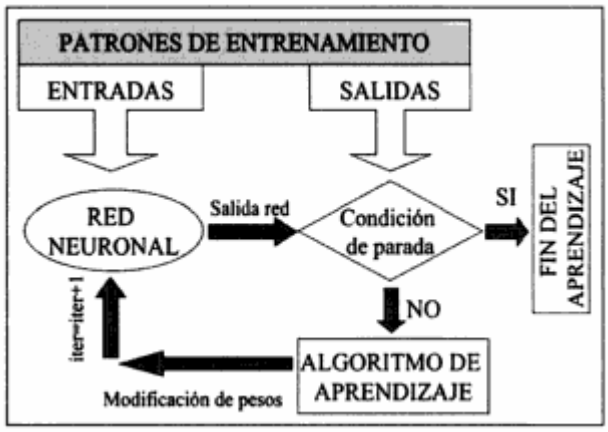

Figura 2.6. *Diagrama de flujo de entrenamiento supervisado. Fuente:* **[8]**.

Entrenamiento estocástico. Se aplican cambios aleatorios en los valores de los pesos de las conexiones de la red y se evalúa su efecto en función de la salida deseada y de unas funciones de probabilidad. Para ello se define la función de energía, que representa el grado de estabilidad de la red, de maneta que el estado de mínima energía corresponda con el funcionamiento que más se ajusta al deseado. Si los cambios aleatorios realizados disminuyen la energía, el cambio se acepta y en caso contrario se aplican los cambios según una distribución de probabilidades. Un ejemplo de RNA que utiliza este tipo de entrenamiento es la máquina de Boltzmann.

Entrenamiento por corrección de error. Consiste en ajustar los pesos de las conexiones de la red en función de la diferencia entre los valores deseados y los obtenidos en la salida de la red, con el objetivo de minimizar esta diferencia.

Dentro del tipo de entrenamiento por corrección del error existe una gran variedad de algoritmos utilizados para las RNA, normalmente asociados a la tipología concreta de red utilizada. Por ejemplo, la regla de entrenamiento del perceptrón utiliza el ajuste de pesos presentados en la ecuación 2.20. Esta regla, formulada por Rossemblatt en 1948, fue una de las primeras reglas implementadas en las redes, pero como se puede observar en 2.20, sólo es aplicable a redes de una sola capa.

$$\Delta w_{ji} = \rho \cdot a_i(y_j - a_j) \qquad (2.20)$$

Donde Δw_{ji} es la variación del peso que va de la neurona i hasta la j, ρ es el factor de aprendizaje que es mayor que cero, a_i es la salida de la neurona $ii, y_j y_j$ es el valor deseado de la salida jj y a_j $a_j$es el valor que obtiene la red para la salida jj.

Posteriormente surgieron algoritmos más evolucionados, que proporcionaban un entrenamiento más rápido y un campo de aplicación más amplio. Un ejemplo es la regla de aprendizaje Delta o regla del mínimo cuadrado, que a diferencia de la regla anterior considera todas las neuronas anteriores a la salida, con lo que se dispone del error de la red global, que se distribuye después entre las conexiones predecesoras.

A raíz de la regla Delta, surgió la regla Delta generalizada, más conocida como algoritmo de *Backpropagation,* que permite aplicar la regla delta a redes multicapa ya que actualiza los pesos de las neuronas de las capas ocultas. Este algoritmo ha dotado a las RNA de un extenso campo de aplicación y es ampliamente utilizado actualmente en las aplicaciones con RNA **[8]**.

2.3.6. Perceptrón multicapa

El perceptrón multicapa es una generalización del perceptrón simple, surgió como consecuencia de las limitaciones de dicha arquitectura en lo referente al problema de la separabilidad lineal. Como se ha discutido, Minsky y Papert mostraron en 1969 que la combinación de varios perceptrones simples –inclusión de neuronas ocultas- podía resultar una solución adecuada para tratar ciertos problemas no lineales. Sin embargo, los autores no presentaron una solución al problema de cómo adaptar los pesos de la capa de entrada a la capa oculta, pues la regla de aprendizaje del perceptrón simple no puede aplicarse en este escenario. No obstante, la idea de combinar varios perceptrones sirvió de base para estudios posteriores realizados por Rumelhart, Hinton y Williams en 1986. Estos autores presentaron una manera de retropropagar los errores medidos en la salida de la red hacia neuronas ocultas, dando lugar a la llamada regla delta generalizada, que no es más que una generalización de la regla delta.

Se ha demostrado que un perceptrón multicapa es capaz de aproximar cualquier función continua sobre un compacto R^n, con al menos una capa oculta de neuronas. La habilidad del perceptrón multicapa para aprender a partir de un conjunto de ejemplos, aproximar relaciones no lineales, filtrar ruido en los datos, etc. Hace que sea un modelo adecuado para abordar problemas reales, sin que esto indique que sean los mejores aproximadores universales. Cada una de las clases de aproximadores tiene sus propias características; se conocen ciertas condiciones bajo las cuales un método es preferible a otro, pero en ningún caso se puede decir que un método sea absolutamente el mejor. Serán las condiciones prácticas de cada problema las que determinen la elección de un aproximador u otro.

Dentro del marco de las redes neuronales, el perceptrón multicapa es en la actualidad una de las arquitectura más utilizadas en la resolución de problemas. Esto es debido, fundamentalmente, a su capacidad como aproximador universal, como se comentó anteriormente, así como a su fácil uso y aplicabilidad. Estas redes han sido aplicadas con éxito para la resolución de problemas en una gran variedad de áreas diferentes, como reconocimiento de habla, reconocimiento de caracteres ópticos, reconocimiento de caracteres escritos, modelización de sistemas dinámicos, conducción de vehículos, diagnósticos médicos, predicción de series temporales, etc.

Es necesario señalar, sin embargo, que aunque sea una de las redes más conocidas y utilizadas, esto no implica que sea una de las más potentes y con mejores resultados en las diferentes áreas de aplicación. De hecho, el perceptrón multicapa posee también una serie de limitaciones, como el largo de proceso de aprendizaje para problemas complejos dependientes de un gran número de variables; la dificultad en ocasiones de codificar problemas reales mediante valores numéricos; la dificultad para realizar un análisis teórico de la red debido a la presencia de componentes no lineales y a la alta conectividad. Por otra parte, es necesario señalar que el proceso de aprendizaje de la red busca en un espacio amplio de funciones, una posible función que relacione las variables de entrada y salida al problema, lo cual puede complicar su aprendizaje y reducir su efectividad en determinadas aplicaciones.

2.3.6.1. Arquitectura del perceptrón multicapa

La arquitectura del perceptrón multicapa se caracteriza porque tiene sus neuronas agrupadas en capas de diferentes niveles. Cada una de las capas está formada por un conjunto de neuronas y se distinguen tres tipos de capas diferentes: la capa de entrada, las capas ocultas y la capa de salida, como se observa en la figura 2.7.

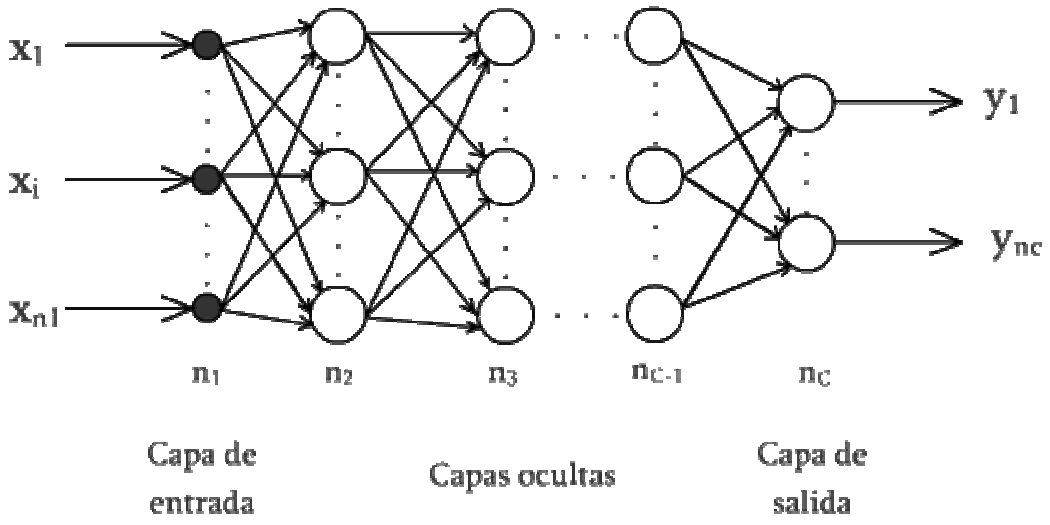

Figura 2.7. *Arquitectura del perceptrón multicapa. Fuente: Propia*.

Las neuronas de la capa de entrada no actúan como neuronas propiamente dichas, sino que se encargan únicamente de recibir las señales o patrones procedentes del exterior y propagar dichas señales a todas las neuronas de la siguiente capa. La última capa actúa como salida de la red, proporcionando al exterior la respuesta de la red para cada uno de los patrones de entrada. Las neuronas de las capas ocultas realizan un procesamiento no lineal de los patrones recibidos.

Como se observa en la figura 2.7, las conexiones del perceptrón multicapa siempre están dirigidas hacia delante, es decir, las neuronas de una capa se conectan con las neuronas de la siguiente capa, de ahí que reciban también el nombre de redes alimentadas hacia delante o redes "*feedforward*". Las conexiones entre las neuronas llevan asociado un número real, llamado peso de la conexión. Todas las neuronas de la red llevan también asociado un umbral, que en el caso del perceptrón multicapa suele tratarse como una conexión más a la neurona, cuya entrada es constante e igual a 1.

2.3.6.2. Propagación de los patrones de entrada

El perceptrón multicapa define una relación entre las variables de entrada y las variables de salida de la red. Esta relación se obtiene propagando hacia delante los valores de las variables de entrada. Para ello, cada neurona de la red procesa la información recibida por sus entradas y produce una respuesta o activación que se propaga, a través de las conexiones correspondientes, hacia las neuronas de la siguiente capa.

Sea un perceptrón multicapa con C capas -C – 2 capas ocultas- y n_c neuronas en la capa c, para c = 1,2,…, C. Sea $W^c = (w_{ij}^c)$ la matriz de pesos asociada a las conexiones de la capa c a la capa c + 1 para c = 1,2,…, C – 1, donde w_{ij}^c representa el peso de la conexión i de la capa c a la neurona j de la capa c + 1; y sea $U^c = (w_{ij}^c)$ el vector de umbrales de las neuronas de la capa c para c = 2, …, C. se denota a_i^c a la activación de la neurona i de la capa c; estas activaciones se calculan del modo:

- Activación de las neuronas de la capa de entrada (a_i^1). Las neuronas de la capa de entrada se encargan de transmitir hacia la red las señales recibidas del exterior.

$$a_i^1 = x_i \; para \; i = 1,2, \dots, n_1 \qquad (2.21)$$

Donde $X = (x_1, x_2, \dots, x_{n1})$ representa el vector o patrón de entrada a la red.

- Activación de las neuronas de la capa oculta $c(a_i^c)$. Las neuronas ocultas de la red procesan la información recibida aplicando la función de activación f a la suma de los productos de las activaciones que recibe por sus correspondientes pesos, es decir:

$$a_i^c = f\left(\sum_{j=1}^{n_{c-1}} w_{ji}^{c-1} a_j^{c-1} + u_i^c\right) para \; i = 1,2, \dots, n_c \; y \; c = 2,3 \dots, C-1 (2.22)$$

Donde a_j^{c-1} son las activaciones de las neuronas de la capa $c - 1$.

- Activación de las neuronas de capa de salida (a_i^c). Al igual que en el caso anterior, la activación de estas neuronas viene dada por la función de activación f aplicada a la suma de los productos de las entradas que recibe por sus correspondientes pesos:

$$y_i = a_i^c = f\left(\sum_{j=1}^{n_{c-1}} w_{ji}^{c-1} a_j^{c-1} + u_i^c\right) para \; i = 1,2, \dots, n_c \qquad (2.23)$$

Donde $Y = (y_1, y_2, \dots, y_{nc})$ es el vector de salida de la red.

La función *f* es la llamada función de activación. Para el perceptrón multicapa, las funciones de activación más utilizadas son la función sigmoidal y la función hiperbólica. Dichas funciones poseen como imagen un rango continuo de valores dentro de los intervalos [0, 1] y [-1, 1], respectivamente, y vienen dadas por las siguientes expresiones:

$$\textit{Función Sigmoidal:}\quad f_1(x) = \frac{1}{1 + e^{-x}} \qquad (2.24)$$

$$\textit{Función tangente hiperbólica:}\quad f_2(x) = \frac{1 - e^{-x}}{1 + e^{-x}} \qquad (2.25)$$

Ambas son funciones crecientes con dos niveles de saturación: el máximo, que proporciona salida 1, y el mínimo, salida 0 para la función sigmoidal, y salida -1 para la tangente hiperbólica **[5].**

2.3.6.3. Algoritmo Backpropagation (Retropropagación)

El algoritmo *Backpropagation* es uno de los más avanzados dentro de la clasificación de aprendizaje por corrección del error de los algoritmos supervisados, y su implementación supuso un gran impulso dentro del campo de las redes neuronales. Fue introducido simultáneamente por Rumelhart, Hinton y Williams en 1986 y su implementación estándar es una generalización de la regla de aprendizaje Delta.

El nombre Backpropagation o de retropropagación del error proviene de la forma que tiene de propagar hacia atrás el error, desde la capa de salida a la de entrada, modificando los pesos de las capas intermedias y permitiendo así su aplicación en modelos de redes neuronales de dos o más capas ocultas.

El nombre se utiliza normalmente para el tipo de entrenamiento, pero por extensión muchos autores denominan Backpropagation no solo al método de aprendizaje, sino también a la arquitectura de RNA que utiliza ese tipo de entrenamiento.

2.3.6.4. Desarrollo matemático del algoritmo

Tanto para desarrollo como para la demostración matemática del algoritmo partiremos del esquema presentado en la figura 2.8. La nomenclatura que aparece en la figura corresponde a las siguientes definiciones:

p_i: patrón de entrada para la neurona de la capa de entrada

R: número de entradas de la red neuronal

w_{ji}^k: peso de la conexión que va de la neurona j a la neurona i de la capa k

Q: número de neuronas de la capa oculta k

N_j^k: neurona de la capa k

a_j^k: valor de la salida obtenido para la neurona j de la capa k

S: número de salidas de la red neuronal

y_h: patrón de salida de la neurona h de la capa de salida

δ_j^k: parámetro de retropropagación de la neurona j de la capa k

Además, aunque no aparecen en la figura, se empleará durante la demostración posterior la siguiente nomenclatura:

m: número de patrones de entrada-salida disponibles

f_j: función de activación de la neurona j

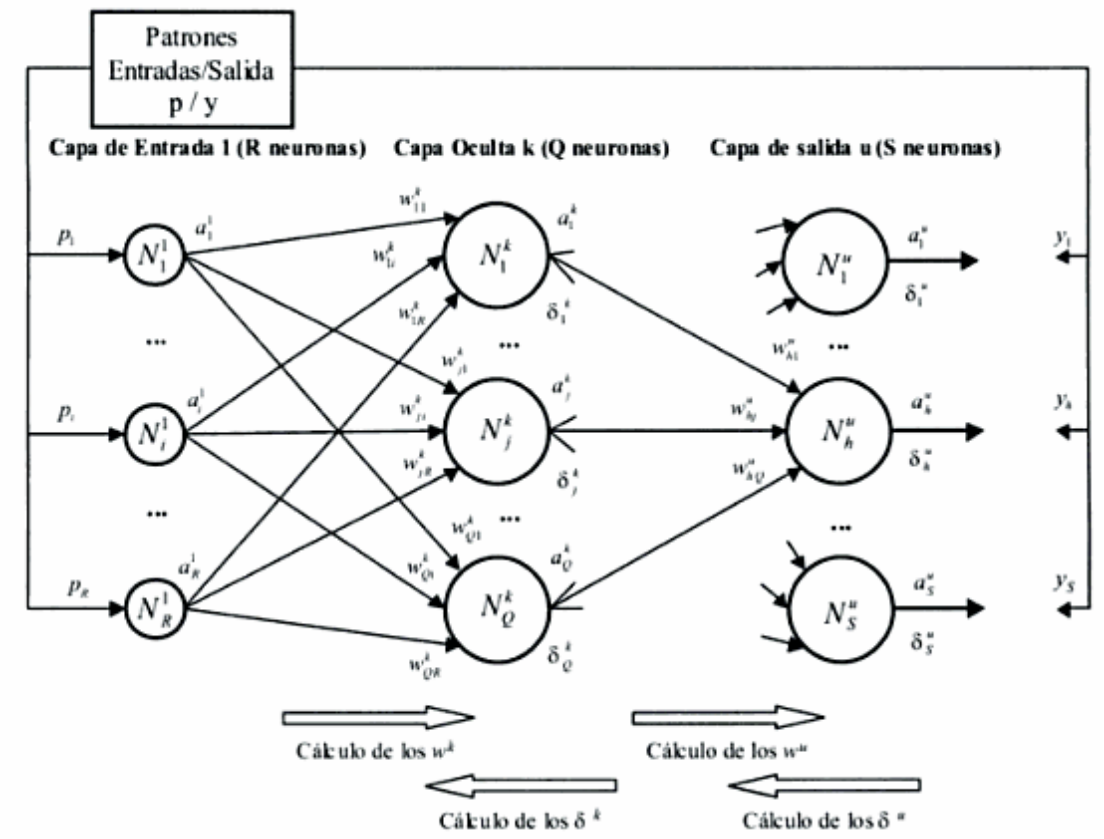

Figura 2.8. *Esquema simplificado del algoritmo Backpropagation. Fuente:* **[8]**.

Para hacer más sencilla la demostración matemática, se va a realizar un conjunto de simplificaciones previas que se presentan a continuación:

- El conjunto de patrones de entrenamiento, formados por *m* pares de valores entrada-salida, se considera compuesto por un solo patrón durante la demostración, para después generalizar a un número mayor de patrones. Por lo tanto, se asume inicialmente $m = 1$.
- El sesgo b no se ha incluido en la demostración, ya que al considerarse asociado a una entrada igual a la unidad, se ha incorporado dentro del conjunto de pesos w_{ji}^{k}.
- Las funciones de activación de las neuronas se asume como derivables.
- A lo largo de la demostración, la notación de capa de neuronas anterior a la actual, es decir la capa entre ésta y la de entrada, se ha denotado con (k-1). De igual manera, la capa posterior (k+1), se empleará para referirse a la capa más cercana a la actual y entre ésta y la de salida.

La representación matemática de la salida de una neurona cualquiera *j* perteneciente a una capa *k,* a la que le llegan conexiones de Q neuronas de la capa anterior k – 1 se corresponde con la siguiente ecuación:

$$a_j^k = f_j(n_j) = f_j\left(\sum_{i=1}^{R} w_{ji}^k \cdot a_i^{k-1}\right) \qquad (2.26)$$

La función objetivo a minimizar durante el entrenamiento será una función de coste E_T y comúnmente se toma el error cuadrático medio (MSE) de las salidas de la red neuronal:

$$E_T = \frac{1}{2m}\sum_{p=1}^{m}\sum_{h=1}^{S}(y_h(p) - a_h^u(p))^2 \qquad (2.27)$$

Puesto que en este caso $m = 1$, la ecuación 2.27 se simplifica:

$$E_T = \frac{1}{2}\sum_{h=1}^{S}(y_h - a_h^u)^2 \qquad (2.28)$$

La solución adoptada para minimizar la función de coste (E_T) es la de descenso por gradiente. Los pesos se modificarán de forma proporcional a la variación relativa del error que se obtiene respecto al peso correspondiente:

$$\Delta w_{ji}^k = -\rho \frac{\partial E_T}{\partial w_{ji}^k} \qquad (2.29)$$

Siendo ρ el factor o tasa de aprendizaje (donde ρ es mayor que cero), que regula la velocidad con la que se modifica los pesos en el proceso de convergencia a la solución deseada. Como se verá en la demostración que se presenta a continuación, a partir de la aplicación de esta ecuación llegaremos a los siguientes resultados para una neurona cualquiera j de una capa oculta k:

$$\Delta w_{ji}^k = \rho \cdot \delta_j^k \cdot a_i^{k-1} \qquad (2.30)$$

En la definición de la ecuación 2.30, el parámetro de retropropagación δ es diferente según se trate de una neurona de una capa oculta o de la salida. Para cualquier neurona de la capa de salida *u* (donde *k* sería el índice de la capa inferior, *k* = *u* - 1) se obtendría:

$$\delta_h^u = f_h^{'} \cdot (y_h - a_h^u) \qquad (2.31)$$

Para una neurona de la capa oculta k (donde u sería el índice de la capa superior con S neuronas, *u* = *k* + *1*), se obtendría:

$$\delta_j^k = f_j^{'} \cdot \sum_{h=1}^{s} \delta_h^u \cdot w_{hj}^u \qquad (2.32)$$

De las ecuaciones anteriores se desprende que existe un parámetro δ_j^k para cada una de las neuronas, de manera que para cualquier capa oculta de la red k, depende de todos los pesos de las neuronas de la capa superior u (sería $k+1$ si hubiera más de una capa oculta) relacionadas con la neurona actual j y con sus respectivos valores de δ_h^u, así como con la derivada de su respectiva función de activación $f_j^{'}$. Esta es una fórmula recurrente, por lo que para obtenerla, se debe empezar por calcular el δ_h^u de la última capa hasta la primera.

Analizando las dos ecuaciones anteriores, donde aparece la derivada de la función de activación ($f^{'}$), se entiende el requisito del algoritmo Backpropagation de que las funciones de activación asociadas a las neuronas sean derivables. El desarrollo matemático necesario para llegar a las ecuaciones 2.31 y 2.32 se describe a continuación.

La demostración del algoritmo Backpropagation se realizará para el caso presentado en la figura 2.8, donde la red tiene una capa de entrada 1, una única capa oculta k y una capa de salida u, aunque las conclusiones son aplicables a redes con más capas ocultas.

Para empezar a iterar, se asigna unos valores iniciales a todos los pesos de las neuronas de la RNA y se calculan las salidas predichas por la RNA. A partir de ahí, se aplica el método de la optimización de descenso por gradiente a la función objetivo E_T entre las salidas predichas y las esperadas. La aplicación directa de este método

permitirá calcular la modificación de los pesos w_{hj}^{u} entre las neuronas *j* de la última capa oculta *k* y las neuronas *h* de la capa de salida u:

$$\Delta w_{hj}^{u} = -\rho \frac{\partial E_T}{\partial w_{hj}^{u}} = -\rho \frac{\partial E_T}{\partial a_h^{u}} \frac{\partial a_h^{u}}{\partial n_h} \frac{\partial n_h}{\partial w_{hj}^{u}} = -\rho(-1)(y_h - a_h^{u}) \cdot \frac{\partial a_h^{u}}{\partial n_h} \frac{\partial n_h}{\partial w_{hj}^{u}} \quad (2.33)$$

Puesto que:

$$a_h^{u} = f_h(n_h) \rightarrow \frac{\partial a_h^{u}}{\partial n_h} = f_h^{'} \quad (2.34)$$

$$n_h = \sum_{j=1}^{Q} (w_{hj}^{u} \cdot a_j^{k}) \rightarrow \frac{\partial n_h}{\partial w_{hj}^{u}} = a_j^{k} \quad (2.35)$$

Se concluye que para la capa de salida u, la modificación de los pesos se realiza según la ecuación:

$$\Delta w_{hj}^{u} = p(y_h - a_h^{u}) \cdot f_h^{'} \cdot a_j^{k} = \rho \cdot \delta_h^{u} \cdot a_j^{k} \quad (2.36)$$

Dada la naturaleza recurrente del algoritmo, una vez se ha demostrado la actualización para los pesos de las neuronas de la capa de salida, se puede pasar ahora a demostrar la actualización de los pesos de las neuronas en una capa oculta. Siguiendo con el ejemplo de la figura 2.8, se va a realizar la demostración para los pesos entre la capa oculta *k* (con neuronas de índice *j*) y otra capa 1 (con neuronas de índice *i*), que para este caso específico es la de entrada, aunque podría ser otra capa oculta anterior en caso de que hubiera más de una capa oculta. Para las neuronas de la capa u posterior a la actual, se utilizará el índice *h*, como se muestra en la figura 2.8.

$$\Delta w_{ji}^{k} = -\rho \frac{\partial E_T}{\partial w_{ji}^{k}} = -\rho \frac{\partial E_T}{\partial a_j^k} \cdot \frac{\partial a_j^k}{\partial n_j} \cdot \frac{\partial n_j}{\partial w_{ji}^{k}} = -\rho \frac{\partial E_T}{\partial a_j^k} \cdot f_j^{'} \cdot a_i^1 \qquad (2.37)$$

$$\Delta w_{ji}^{k} = -\rho \left(\sum_{h=1}^{s} \frac{\partial E_T}{\partial a_j^k} \right) \cdot f_j^{'} \cdot a_i^1 = -\rho \left(\sum_{h=1}^{s} \frac{\partial E_T}{\partial a_h^u} \cdot \frac{\partial a_h^u}{\partial n_h} \cdot \frac{\partial n_h}{\partial a_j^k} \right) \cdot f_j^{'} \cdot a_i^1 \quad (2.38)$$

$$\Delta w_{ji}^{k} = \rho \left(\sum_{h=1}^{S} \delta_h^u \cdot \frac{\partial n_h}{\partial a_j^k} \right) \cdot f_j^{'} \cdot a_i^1 \qquad (2.39)$$

El último paso se justifica a partir del punto anterior de la demostración para las neuronas de la última capa, donde se veía que:

$$\delta_h^u = (y_h - a_h^u) \cdot f_h^{'} = -\frac{\partial E_T}{\partial a_h^u} \frac{\partial a_h^u}{\partial n_h} \qquad (2.40)$$

Por ello y sabiendo que:

$$n_h = \sum_{j=1}^{Q} (w_{hj}^u \cdot a_j^k) \rightarrow \frac{\partial n_h}{\partial a_j^k} = w_{hj}^u \qquad (2.41)$$

Despejamos y obtenemos:

$$\Delta w_{ji}^{k} \left(\sum_{h=1}^{s} \delta_h^u \cdot w_{hj}^u \right) \cdot f_j^{'} \cdot a_i^1 = \rho \cdot \delta_j^k \cdot a_i^1 = \rho \cdot \delta_j^k \cdot a_i^{k-1} \qquad (2.42)$$

Sabiendo que la capa 1 es la anterior a la capa *k* (es decir *k – 1 = 1*), la expresión 2.42 concluye la demostración. Si generalizamos para *m* patrones, la actualización de los pesos se debería hacer según la siguiente ecuación:

$$\Delta w_{ji}^{k} = \rho \cdot \sum_{p=1}^{m} \delta_j^k(p) \cdot a_i^{k-1}(p) \qquad (2.43)$$

2.3.6.5. Método del gradiente

El método del gradiente fue introducido por Cauchy alrededor de 1847. Se basa en una aproximación lineal de la función a minimizar, actualizando la variable a modificar (en este caso los pesos de la RNA) en dirección contraria al gradiente, tal y como se muestra en la ecuación 2.45:

$$E_T(w + \Delta w) \approx E_T(w) + \Delta w \cdot \nabla E_T(w) \qquad (2.44)$$

$$\Delta w = -\rho \cdot \nabla E_T(w) \; \rho > 0 \qquad (2.45)$$

La aplicación de este método junto con la aplicación de la regla de la cadena para propagar hacia las capas iniciales la información del error entre los valores de la red y los previstos por el entrenamiento, es lo que constituye el algoritmo estándar de *Backpropagation*.

El factor de aprendizaje ρ permite controlar el grado de modificación de los pesos en el algoritmo de entrenamiento. Además, la elección de este factor influye tanto en la rapidez como en la eficiencia del propio entrenamiento. Por una parte con un factor grande de aprendizaje se podría obtener mayor rapidez en la optimización, pero a su vez también se observaría mayor tendencia a las oscilaciones en la superficie del error que se intenta minimizar. Por otra parte, con un factor pequeño, la convergencia a un mínimo sería más lenta, pero disminuiría la posibilidad de pasar

por alto un mínimo en el error. Dependiendo también del factor de aprendizaje se puede tener más o menos éxito para evitar el estancamiento del algoritmo en mínimos locales.

Los algoritmos de descenso por gradiente presentan la gran ventaja de ser simples y fáciles de implementar, aunque en algunos casos sus tiempos de convergencias pueden ser muy altos. Para intentar salvar estos inconvenientes también han surgido diversas implementaciones dentro de este tipo de optimización.

2.3.6.6. Aprendizaje con momentum

Para evitar posibles oscilaciones bruscas sobre la superficie del error, se puede introducir en la actualización de los pesos el término del *momentum* (α), que incluye una dependencia con la actualización anterior de los pesos, a modo de inercia. De este modo se consigue que la red no se actualice sólo en función del gradiente local, sino que tenga en cuenta las tendencias más recientes en las superficies del error. La inclusión del *momentum* actúa de forma similar a un filtro de paso-bajo, ya que evita los pequeños cambios en la superficie del error y disminuye las posibilidades de caer en un mínimo local.

La ecuación con la que se actualizan los pesos al incluir el término de *momentum* es la siguiente:

$$\Delta w_{iter+1} = -\rho \cdot \nabla E_T(w_{iter}) + \alpha \cdot \Delta w_{iter} \quad \rho > 0 \qquad 0 < \alpha < 1 \qquad (2.46)$$

Siendo Δw_{iter+1} la modificación del peso en la iteración siguiente, ρ el factor de aprendizaje, $E_T(w_{iter})$ el error de la red correspondiente a los pesos de la iteración actual, α el factor o importancia de *momentum* que controlará la influencia del

término de inercia (que debe estar entre cero y uno) y Δw_{iter} es la modificación del peso correspondiente a la iteración actual **[8]**.

2.3.7. Redes de neuronas de base radial

Las redes de neuronas de base radial son redes multicapa con conexiones hacia adelante, al igual que el perceptrón multicapa. Las redes de base radial se caracterizan porque están formadas por una única capa oculta y cada neurona de esta capa posee un carácter local, en el sentido de que cada neurona oculta de la red se activa en una región diferente del espacio de patrones de entrada. Este carácter local viene dado por el uso de las llamadas funciones de base radial, generalmente la función gausiana, como funciones de activación. Las neuronas de la capa de salida de las redes de base radial simplemente realizan una combinación lineal de las activaciones de las neuronas ocultas.

La mayor contribución a la teoría, diseño y aplicaciones de las redes de neuronas de base radial se debe a Moody y Darken, Renals y a Poggio y Girossi. Uno de los objetivos iniciales de los autores era construir una red de neuronas que requiriese un menor tiempo de aprendizaje que el que necesita el perceptrón multicapa y, de este modo, disponer de una red de neuronas que pudiera ser apropiada para aplicaciones a tiempo real. Esto se consiguió incorporando funciones de activación locales en las neuronas ocultas de la red, lo cual permitía que sólo unas pocas neuronas ocultas tuvieran que ser procesadas para nuevos patrones de entrada. Al igual que el perceptrón multicapa, las redes de neuronas de base radial son aproximadores universales, en el sentido de que pueden aproximar cualquier función continua sobre un compacto de R^n.

Las funciones de base radial definen hiperesferas o hiperelipses que dividen el espacio de entrada. Por tanto, cada neurona oculta de la red de base radial construye una aproximación local y no lineal en una determinada región de dicho espacio.

Puesto que la salida de la red es combinación lineal de las funciones de base radial, las aproximaciones que construyen las redes de base radial son combinaciones lineales de múltiples funciones locales y no lineales. De este modo, se suele decir que las redes de base radial aproximan relaciones complejas mediante una colección de aproximaciones locales menos complejas, dividiendo el problema en varios sub problemas menos complejos. Esto hace que las aproximaciones construidas por las redes de base radial sean de naturaleza diferente a las aproximaciones globales y basadas en hiperplanos que construye el perceptrón multicapa, sin embargo, cada una de estas clases de aproximaciones –redes de base radial y perceptrón multicapa- tiene sus propias características.

Las redes de neuronas de base radial han sido aplicadas a una gran variedad de problemas, aunque es necesario señalar que su aplicación no ha sido tan extendida como el caso del perceptrón multicapa, sin embargo, se han utilizado en diferentes campos, como análisis de series temporales, procesamiento de imágenes, diagnósticos médicos, reconocimiento automático del habla, etc.

2.3.7.1. Arquitectura de las redes de base radial

Las redes de neuronas de base radial están formadas por tres capas de neuronas: la capa de entrada, una única capa oculta y la capa de salida, como se muestra en la figura 2.9. La capa de entrada la componen un conjunto de neuronas que reciben las

señales del exterior, transmitiéndolas a la siguiente capa sin realizar ningún proceso sobre dichas señales. Las neuronas de la capa oculta reciben las señales de la capa de entrada y realizan una transformación local y no lineal sobre dichas señales. Este carácter local es lo que las diferencia del perceptrón multicapa, no solo en cuanto a arquitectura, sino también en cuanto a comportamiento. Esta capa es la única que incluye componentes no lineales en las redes de base radial. Y, finalmente, la capa de salida que realiza una combinación lineal de las activaciones de las neuronas ocultas, que actúa además como salida de la red.

Las redes de neuronas de base radial son redes con conexiones hacia delante, como se observa en la figura 2.9, y estas conexiones se dirigen siempre de una capa a la siguiente capa. La red se caracteriza porque las conexiones de la capa de entrada a la capa oculta no llevan asociado ningún peso, mientras que, y como es habitual, en el contexto de redes de neuronas, las conexiones de la capa oculta a la capa de salida sí llevan asociado un número real o peso de la conexión. En lo referente a los umbrales de las neuronas, en las redes de base radial únicamente las neuronas de salida poseen un umbral, que también se suele tratar al igual que en el perceptrón multicapa como una conexión más de la neurona cuya entrada es constante e igual a 1.

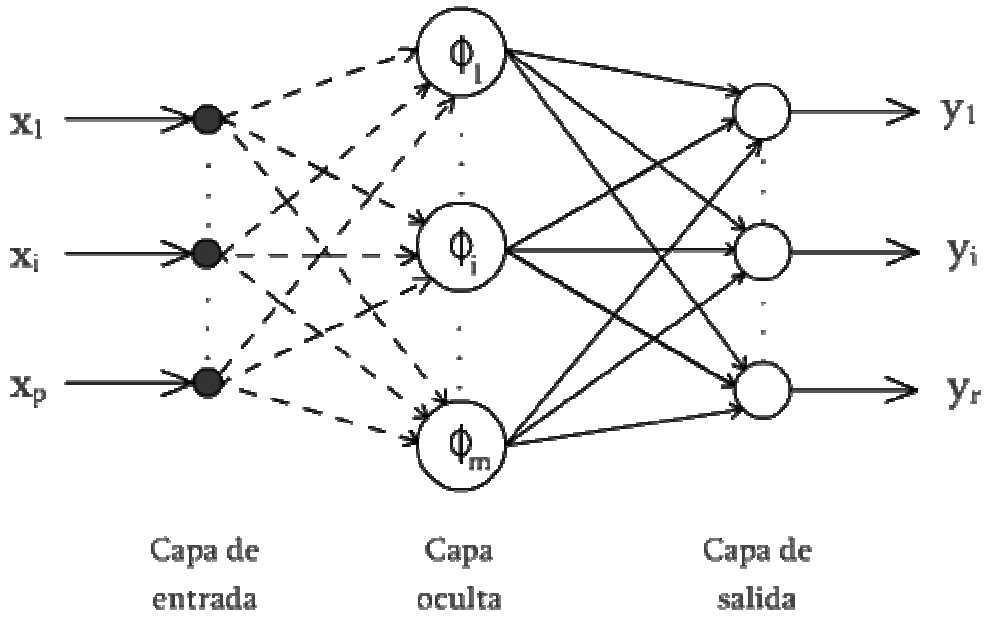

Figura 2.9. *Arquitectura de la red de neuronas de base radial. Fuente: Propia.*

Las redes de neuronas de base radial definen una relación no lineal entre las variables de entradas y las variables de salida de la red, propagando hacia las salidas las señales o muestras recibidas en la entrada.

2.3.7.2. Activación de las neuronas de la red de base radial

Dada una red de neuronas de base radial con *p* neuronas en la capa de entrada *m* neuronas en la capa oculta y *r* neuronas en la capa de salida, las activaciones de las neuronas de salida para el patrón de entrada *n*, $X(n) = (x_1(n), x_2(n), \ldots, x_p(n))$, denotadas como $y_k(n)$, vienen dadas por la siguiente expresión:

$$y_k(n) = \sum_{i=1}^{m} w_{ik} \emptyset_i(n) + u_k \; para \; k = 1,2,\ldots,r \qquad (2.47)$$

Donde w_{ik} representa el peso de la conexión de la neurona oculta *i* a la neurona de salida *k*, u_k es el umbral de la neurona de salida *k*, $\emptyset_i(n)$ son las activaciones de las neuronas ocultas para el patrón de entrada $X(n)$. Se observa en la ecuación 2.46 que las neuronas de salida de la red utilizan la función de activación identidad, realizando una transformación lineal de las activaciones de todas las neuronas ocultas.

Las funciones $\emptyset_i$, también conocidas como funciones de base radial, determinan las activaciones de las neuronas ocultas de la red en función del vector de entrada a la red $X(n)$ y vienen dadas por la siguiente expresión:

$$\emptyset_i(n) = \emptyset\frac{\|X(n) - C_i\|}{d_i} \; para \; i = 1,2,\dots,m \qquad (2.48)$$

Donde $\emptyset$ es una función de base radial; $C_i = (C_{i1}, \dots, C_{ip})$ son vectores que representan los centros de la función de base radial; d_i son números reales que representan la desviación, anchura o dilatación de la función de base radial; y $\|\ \ \|$ es la distancia euclidiana del vector de entrada $X(n)$ al centro C_i, definida como:

$$\|X(n) - C_i\| = \left(\sum_{j=1}^{p}(x_j(n) - c_{ij})^2\right)^{1/2} \qquad (2.49)$$

Por lo tanto, la activación de una neurona oculta en las redes de base radial depende de la distancia del patrón de entrada $X(n)$ al centro C_i de la función de base radial. Estas funciones bases $\emptyset$ poseen un carácter local, pues son funciones que

alcanzan un nivel cercano al máximo de su recorrido cuando el patrón de entrada $X(n)$ está próximo al centro de la neurona; a medida que el patrón se aleja del centro, el valor de función va tendiendo al valor mínimo de su recorrido, como se puede apreciar en la ecuación 2.48.

La función de base radia $\emptyset$ puede adoptar diferentes formas y expresiones (ecuación 2.48), entre ellas:

- Función gausiana:

$$\emptyset(r) = e^{\left(\frac{-r^2}{2}\right)} \qquad (2.50)$$

- Función inversa cuadrática:

$$\emptyset(r) = \frac{1}{1+r^2} \qquad (2.51)$$

- Función inversa multicuadrática:

$$\emptyset(r) = \frac{1}{\sqrt{1+r^2}} \qquad (2.52)$$

En el contexto de redes de neuronas de base radial, la más utilizada es la función gausiana. Por tanto, la activación de las neuronas ocultas de las redes de base radial viene dada, generalmente, por la siguiente expresión:

$$\emptyset(r) = exp^{-\frac{\|x(n)-c_i\|^2}{2d_i^2}} = exp^{-\frac{\sum_{j=1}^{p}(x_j(n)-c_{ij})^2}{2d_i^2}} \quad para\ i = 1,2,\dots,m \qquad (2.53)$$

Las salidas de las redes de neuronas de base radial (ecuación 2.47) son, por tanto, una combinación lineal de gausianas, cada una de las cuales se activa para una determinada porción del espacio definido por los patrones de entrada.

2.3.7.3. Carácter local de las redes de base radial

Las funciones de base radial $\emptyset$ se caracterizan porque poseen un nivel máximo de activación para valores de entrada cercanos a cero y dicho nivel decrece a medida que la variable de entrada se aleja de dicho punto.

En la ecuación 2.53 se observa que la activación de la neurona oculta *i* en una red de base radial viene dada por la función gausiana de centro C_i y desviación d_i. Por tanto, si el patrón de entrada a la red $X(n)$ está en la vecindad del centro C_i, la neurona oculta *i* alcanzará un valor alto de activación. A medida que el patrón $X(n)$ se aleja del centro dependiendo de la desviación la activación de la neurona disminuye, y puede activarse otra neurona oculta de la red.

Por esta razón, se dice que las redes de neuronas de base radial son redes con carácter local, ya que, dado un patrón de entrada a la red, solo aquellas neuronas ocultas cuyos centros estén en la vecindad de dicho patrón se van a activar; el resto de las neuronas ocultas permanecerán inactivas o con un menor nivel de activación. Esto no sucede cuando se utiliza funciones de activaciones sigmoidales, como en el caso del perceptrón multicapa, pues estas se activan en todo un rango de valores. Esto hace que, aunque ambas arquitecturas sean redes multicapa con conexiones hacia adelante, cada una de ellas posee sus propias características, así como sus ventajas e inconvenientes.

2.3.7.4. Aprendizaje de las redes de base radial

El proceso de aprendizaje implica la determinación de todos los parámetros que intervienen en la red de base radial. Estos son: los centros y las desviaciones de las neuronas ocultas y los pesos de la capa oculta a la capa de salida, así como los umbrales de las neuronas de salida.

Debido a que las capas de neuronas en una red de base radial realizan tareas diferentes, es razonable separar el proceso de optimización de los parámetros de la capa oculta y los de la capa de salida mediante la utilización de diferentes técnicas. Así, para los parámetros de la capa oculta –centros y desviaciones- el proceso de aprendizaje debe estar guiado por una optimización en el espacio de patrones de entrada, pues cada una de las neuronas oculta de la red de base radial va a representar una zona diferente del espacio de entrada, sin embargo, para los parámetros de la capa de salida la optimización se debe realizar en base a la salida que se desea obtener o salidas deseadas, ya que las redes de base radial se utilizan para aproximar relaciones entre el conjunto de variables de entrada y salida que definen el problema. Por tanto, uno de los mecanismos más usados para el aprendizaje de las redes de base radial es el llamado método híbrido, que combina dos fases: Una fase no supervisada para la determinación de los centros y otra supervisada para la determinación de los pesos y umbrales.

2.3.7.4.1. Método de aprendizaje híbrido

El método híbrido realiza el aprendizaje de las redes de base radial en dos fases:

- Fase no supervisada: determinación de los centros y amplitudes de las neuronas de la capa oculta.
- Fase supervisada: determinación de pesos y umbrales de la capa de salida.

El proceso de optimización en cada una de las fases se formula con un objetivo diferente y las técnicas para su resolución serán también, por tanto, diferentes.

Fase no supervisada

Puesto que las neuronas ocultas de las redes de base radial se caracterizan porque representan zonas diferentes del espacio de patrones de entrada, los centros y las desviaciones de las funciones de base radial deben ser determinados con este objetivo, es decir, con el objetivo de clasificar el espacio de entrada en diferentes clases. El representante de cada clase será el centro de la función de base radial y la desviación vendrá dada por la amplitud de cada clase.

- **Determinación de los centros: algoritmos de K-medias**

Los centros de las funciones de base radial se determinan mediante un algoritmo de clasificación no supervisada que permita dividir el espacio de patrones de entradas en clases. El número de clases es el número de neuronas ocultas en la red de base radial. El método más utilizado es el algoritmo de K-medias, aunque es necesario destacar que cualquier algoritmo de clasificación no supervisado podría ser utilizado, como, por ejemplo los mapas auto-organizados Kohonen.

El algoritmo K-medias es un algoritmo de clasificación no supervisado mediante el cual el espacio de patrones de entrada se divide en *k* clases o regiones. El representante de cada una de estas clases, C_i, será el centro de la neurona oculta *i*.

Dichos centros se determinan con el objetivo de minimizar las distancias euclidianas entre los patrones de entrada y el centro más cercano, es decir:

$$J = \sum_{i=1}^{K} \sum_{n=1}^{N} M_{in} \|X(n) - C_i\| \qquad (2.54)$$

Donde *N* es el número de patrones, $\|\ \ \|$ es la distancia euclidiana, $X(n)$ es el patrón de entrada n y M_{in} es la función de pertenencia, que vale 1 sí el centro C_i es el más cercano al patrón $X(n)$, y cero en otro caso, es decir:

$$M_{in} \begin{cases} 1 & si\ \|X(n) - C_i\| < \|X(n) - C_s\| \forall\ s \neq i, s = 1,2,\dots,K \\ 0 & en\ otro\ caso \end{cases} \qquad (2.55)$$

Dado *K* el número de clases, $\{X(n) = (x_1(n), x_2(n), \dots, x_p(n))\}_{n=1\dots N}$ el conjunto de patrones de entrada y $\{C_i = (c_{i1}, c_{i2}, \dots, c_{ip}\}_{i=1,\dots,K}$ los centros de las clases, los pasos para la aplicación del algoritmo son los siguientes:

Paso 1: Se inicializan los centros de las *K* clases. Pueden inicializarse a *K* patrones aleatorios del conjunto de patrones disponibles, o bien puede realizarse aleatoriamente, en cuyo caso es conveniente que se inicialicen dentro del rango de valores de los patrones de entrada.

Paso 2: Se asignan N_i patrones de entrada a cada clase *i* del siguiente modo: el patrón $X(n)$ pertenece a las clase *i* si $\|X(n) - C_i\| < \|X(n) - C_s\| \forall\ s \neq i, s = 1,2,\dots,K$. Por tanto, cada clase tendrá asociado un determinado número de patrones de entrada, aquellos más cercanos al centro de la clase.

Paso 3: Se calcula la nueva posición de los centros de las clases como la media de todos los patrones que pertenecen a su clase, es decir:

$$c_{ij} = \frac{1}{N_i}\sum_{n=1}^{N} M_{in}x_j(n) para\ j = 1,2,\dots,p,\quad i = 1,2,\dots,K \qquad (2.56)$$

Paso 4: Se repiten los pasos 2 y 3 hasta que las nuevas posiciones de los centros no se modifiquen respecto a su posición anterior, es decir, hasta que:

$$\left\| C_i^{nuevo} - C_i^{anterior} \right\| < \varepsilon \ \forall\ i = 1,2,\dots,K \qquad (2.57)$$

Siendo ε un número real positivo próximo a cero que marca la finalización del algoritmo.

El algoritmo de K-medias es un método fácil de implementar y usar; suele ser un algoritmo bastante eficiente en problemas de clasificación, pues converge en pocas iteraciones hacia un mínimo de la función *J* dada por la ecuación 2.54, aunque podría tratarse de un mínimo local.

Uno de los inconvenientes o desventajas que se puede atribuir al algoritmo K-medias es su dependencia de los valores iniciales asignados a cada centro, lo cual hace que en muchas ocasiones, y siempre dependiendo del problema, se obtengan soluciones locales.

- **Determinación de las amplitudes**

Una vez determinados los centros de las funciones de base radial, las amplitudes o desviaciones de dichas funciones deben calcularse de manera que cada neurona oculta se active en una región del espacio entrada y de manera que el solapamiento de las zonas de activación de una neurona a otra sea lo más ligero posible, para suavizar así la interpolación.

Las amplitudes de cada función de base radial se pueden determinar usando heurísticas, las cuales permiten que el solapamiento entre las neuronas ocultas sea lo más suave posible. Se puede utilizar diferentes heurísticas, como, por ejemplo:

- Media uniforme de las distancias euclídeas del centro C_i a los p centros más cercanos:

$$d_i = \frac{1}{p}\sum_{p} \|C_i - C_p\| \qquad (2.58)$$

- Otra opción bastante eficiente es determinar la amplitud de la función de base radial como la media geométrica de la distancia del centro a sus dos vecinos más cercanos:

$$d_i = \sqrt{\|C_i - C_t\|\|C_i - C_s\|} \qquad (2.59)$$

Siendo C_t y C_s los dos centros más cercanos al centro C_i.

Fase supervisada

En esta fase se calculan los pesos y umbrales de las neuronas de salida de la red. En este caso, el objetivo es minimizar las diferencias entre las salidas de la red y las salidas deseadas. Por tanto, el proceso de aprendizaje está guiado por la minimización de una función error computada en la salida de la red:

$$E = \frac{1}{N}\sum_{n=1}^{N} e(n) \qquad (2.60)$$

Donde N es el número de patrones o muestras y $e(n)$ es el error cometido por la red para el patrón $X(n)$, que viene dado generalmente por:

$$e(n) = \frac{1}{2}\sum_{k=1}^{r}\left(s_k(n) - y_k(n)\right)^2 \qquad (2.61)$$

Siendo $Y(n) = \left(y_1(n), \ldots, y_r(n)\right) y\ S(n) = \left(s_1(n), \ldots, s_r(n)\right)$ los vectores de salida de la red y salida deseada para el patrón de entrada $X(n)$, respectivamente.

- **Mínimos cuadrados**

Para resolver este problema de optimización se suele utilizar una técnica basada en la corrección del error. En la ecuación 2.47 se observa que las salidas de la red de base radial dependen linealmente de los pesos y umbrales, por lo que un método bastante simple y eficiente es el algoritmo de los mínimos cuadrados. De este modo, los pesos y umbrales de la red se determinan mediante un proceso iterativo gobernado por la siguiente ley:

$$w_{ik}(n) = w_{ik}(n-1) - \alpha_1 \frac{\partial e(n)}{\partial w_{ik}} \qquad (2.62)$$

$$u_k(n) = u_k(n-1) - \alpha_1 \frac{\partial e(n)}{\partial u_k} \qquad (2.63)$$

$para\ k = 1,2,\ldots,r\ y\ para\ i = 1,\ldots,m$

Donde $e(n)$ es el error dado por la ecuación 2.61 y α_1 es la razón o tasa de aprendizaje.

Teniendo en cuenta la expresión del error (ecuación 2.61) y que el peso w_{ik} y el umbral u_k únicamente afectan a la neurona de salida k, se obtiene que:

$$\frac{\partial e(n)}{\partial w_{ik}} = -(s_k(n) - y_k(n)\frac{\partial y_k(n)}{\partial w_{ik}} \qquad (2.64)$$

$$\frac{\partial e(n)}{\partial u_k} = -(s_k(n) - y_k(n)\frac{\partial y_k(n)}{\partial u_k} \qquad (2.65)$$

Derivando la salida $y_k(n)$ de la red de base radial dada en la ecuación 2.47 respecto a los pesos y umbrales, se obtiene que:

$$\frac{\partial e(n)}{\partial w_{ik}} = \emptyset_i(n) \qquad (2.66)$$

Donde $\emptyset_i(n)$ es la activación de la neurona oculta i para el patrón de entrada $X(n)$, y

$$\frac{\partial e(n)}{\partial u_k} = 1 \qquad (2.67)$$

Por tanto, las leyes dadas por las ecuaciones 2.62 y 2.63 para adaptar los pesos y umbrales de la capa de salida de la red de base radial se pueden escribir de la siguiente forma:

$$w_{ik}(n) = w_{ik}(n-1) + \alpha_1(s_k(n) - y_k(n))\emptyset_i(n) \qquad (2.68)$$

$$u_k(n) = u_k(n-1) + \alpha_1(s_k(n) - y_k(n)) \qquad (2.69)$$

$$para\ k = 1,2,\dots,r\ y\ para\ i = 1,\dots,m$$

Cuando se calculan los pesos mediante la ley de aprendizaje dada por las ecuaciones 2.68 y 2.69, la convergencia es bastante rápida, consiguiendo una solución en un número pequeño de iteraciones o ciclos de aprendizaje.

2.3.8. Redes de base radial frente a perceptrón multicapa

Las redes de base radial y el perceptrón multicapa son dos tipos de redes de neuronas no lineales que poseen sus neuronas agrupadas en capas y las conexiones están dirigidas hacia adelante –redes feedforward-. Ambas redes son además aproximaciones universales, pues se ha demostrado que pueden aproximar cualquier función continua.

Estos dos tipos de redes presentan también algunas diferencias, como el número de capas ocultas –una única capa en el caso de las redes de base radial y tantas capas como se desee en el caso del perceptrón multicapa -en las redes de base radial las conexiones de la capa de entrada a la oculta no llevan pesos asociados- y, por ejemplo, también la linealidad en la capa de salida, la cual no es imprescindible para el caso del perceptrón multicapa.

Sin embargo, la principal diferencia entre las redes de base radial y el perceptrón multicapa radica en la función de activación de las neuronas ocultas de la red. Las primeras utilizan funciones de base radial, las cuales hacen que las neuronas ocultas de la red posean un carácter local, activándose cada neurona en una determinada región del espacio entrada. Por otro lado, el perceptrón multicapa emplea funciones de activación sigmoidales, que se activen en todo el espacio de entrada.

El uso de diferentes funciones de activación con las propiedades anteriormente mencionadas, hace que cada una de estas arquitecturas, perceptrón multicapa y redes de base radial, tenga sus propias características, las cuales se analizan a continuación:

El perceptrón multicapa construye aproximaciones globales: debido al uso de funciones de activación sigmoidal, el perceptrón multicapa construye relaciones globales entre los datos de entrada y salida disponibles. Esto hace que el aprendizaje de la red sea lento, pues el cambio en un solo peso de la red provoca cambios en la salida para todos los patrones de entrada presentados anteriormente, reduciéndose así el efecto de previos ciclos de aprendizaje y retrasando la convergencia del algoritmo de aprendizaje.

Las redes de base radial construyen aproximaciones locales: cada neurona oculta de la red de base radial se especializa en una determinada región del espacio de entrada y construyen una aproximación local de dicha región. Por tanto, la relación que definen las redes de base radial entre los datos de entrada y salida es una suma de funciones no lineales y locales para diferentes regiones del espacio de entrada. A diferencia de cuando se construyen aproximaciones globales, la construcción de aproximaciones locales permite que el aprendizaje sea más rápido, ya que el cambio en un solo peso de la red afecta únicamente a la neurona oculta asociada a dicho peso y, por tanto, a un determinado grupo de patrones de entrada, los pertenecientes a la clase que representa la neurona oculta en cuestión.

Debido al carácter local, el aprendizaje de estas redes es, generalmente, menos sensible al orden de presentación de los patrones que en el caso del perceptrón multicapa.

En muchos casos, sin embargo, ocurre que para poder construir una aproximación mediante la suma de aproximaciones locales se requiere un alto número de neuronas ocultas, lo cual podría influir negativamente en la capacidad de generalización de las redes de base radial.

Finalmente, debe señalarse que el número de neuronas ocultas de la red puede aumentar exponencialmente con la dimensión del espacio de entrada. Por tanto, para

aplicaciones que requieren un alto número de variables de entrada, las redes de base radial podrían no ser las más adecuadas **[5]**.

CAPITULO 3

FASE DE ANÁLISIS

3.1. Requerimientos

La definición de los requerimientos es un paso fundamental para comprender de manera clara las necesidades del proyecto que se desea elaborar. En la fase de análisis estos requisitos se refinan, para obtener una descripción de los mismos que sea fácil de mantener y que nos ayude a estructurar el sistema entero, incluyendo su arquitectura.

En este caso, se desea desarrollar una aplicación basada en técnicas de redes neuronales artificiales, para el reconocimiento facial, que sea capaz de memorizar un conjunto de rostros de individuos, donde cada uno dispondrá de un número determinado de imágenes en diferentes perspectivas. También se debe permitir escoger entre dos formas de funcionamiento, es decir, que pueda realizar la tarea de identificación o verificación de individuos. Sin embargo, existen ciertos aspectos que se deben tomar en cuenta, tal como el formato y la resolución de las imágenes faciales basadas en niveles de gris.

Partiendo de lo expuesto y luego de analizar el problema, a continuación se describen los siguientes requerimientos necesarios para la realización del proyecto:

3.1.1. Requisitos funcionales

Describen lo que debe realizar el software.

- Seleccionar las imágenes de los rostros basadas en niveles de gris.

- Cargar las memorias almacenadas, sí existen.
- Configurar los parámetros de las redes neuronales artificiales.
- Entrenar las redes neuronales artificiales con las imágenes de los rostros seleccionados.
- Mostrar el resultado del entrenamiento y el tiempo empleado por cada red para lograr el aprendizaje.
- Memorizar el entrenamiento obtenido o actualizar la memoria, si ya existe, en caso de obtener un mejor resultado para los rostros seleccionados.
- Reconocer (identificar o verificar) los rostros seleccionados, mostrando el porcentaje de acierto presentado por cada red, y el porcentaje de acierto general.
- Configurar el número de sujetos y el número de caras por sujeto.

3.1.2. Requisitos no funcionales

Describen las cualidades generales que ha de tener el software al realizar su función.

- La aplicación debe ser realizada bajo plataforma de software libre basándose en el decreto presidencial 3.390, con el objeto de permitir que el código fuente pueda ser modificado sin restricciones legales además de reducir el costo de desarrollo.
- La estructura y diseño de la aplicación deben adaptarse fácilmente a cualquier cambio o mejora en la misma.

- La aplicación debe poseer una interfaz amigable y de fácil acceso y manejo.
- La aplicación debe ser multiplataforma.

3.1.3. Requisitos de software

Describen los requerimientos mínimos de software que debe cumplir el computador para poder soportar y/o ejecutar la aplicación.

- Sistema Operativo Windows XP (o posterior).
- JRE 1.6 (Java Runtime Environment) como máquina virtual y plataforma de ejecución.

3.1.4. Requisitos de hardware

Describen los requerimientos mínimos de hardware que debe cumplir el computador para poder soportar y/o ejecutar la aplicación.

- Procesador Intel Pentium IV de 2.0 GHz (o posterior).
- 512 MB de memoria principal.
- 10 GB de disco duro.
- Monitor de 15” (o superior).

3.2. Identificación y descripción de riesgos

De manera general y sencilla el riesgo se puede definir como la contingencia o proximidad de un daño. Al efectuar el tratamiento de los riesgos, con el fin de

minimizar los posibles daños, es necesario analizar y conocer los factores de riesgo para reducir sus efectos mediante la realización de las acciones pertinentes. Por lo tanto para identificar los riesgos se deben estudiar las incertidumbres, que equivalen a las probabilidades de que suceda un hecho desfavorable que impida la finalización exitosa del proyecto.

La identificación de los riesgos ayuda a los desarrolladores a planificar e implementar anticipadamente respuestas apropiadas a los problemas que puedan surgir en el proyecto.

El resultado de de la identificación de riesgos para desarrollar la aplicación para el reconocimiento facial mediante redes neuronales artificiales, es una lista que contiene los riesgos que se han determinado y su prioridad correspondiente.

Entre los riesgos identificados se encuentran los siguientes:

1. **Reducir las imágenes faciales**
   - Descripción: Fallos en la técnica seleccionada para la reducción de las imágenes de los rostros, en cuanto a la obtención de las características principales, así como fallos en la recuperación de las imágenes reducidas a su estado original.
   - Prioridad: crítico.
   - Responsabilidad: de los desarrolladores del software.
   - Contingencia: asegurar la efectividad de la técnica, de entre las existentes, para la reducción de imágenes basadas en niveles de gris, antes de seleccionarla para aplicarla al problema.

2. Convergencia de las redes neuronales artificiales

- Descripción: imposibilidad de las redes neuronales artificiales para encontrar la condición que les permita finalizar su entrenamiento, lo que resultaría en fallos al momento de identificar o reconocer sujetos.
- Prioridad: crítico.
- Responsabilidad: de los desarrolladores del software.
- Contingencia: establecer los estados iniciales adecuados para obtener un comportamiento apropiado por parte de las redes neuronales en el momento del entrenamiento, permitiendo alcanzar la convergencia esperada.

3. Manejo de la Aplicación

- Descripción: interfaz poco amigable que dificulte la interacción con el usuario, en cuanto al uso y manejo de la aplicación.
- Prioridad: secundario.
- Responsabilidad: de los desarrolladores del software.
- Contingencia: toma de precauciones en el diseño de la interfaz y elaborar un manual de usuario en donde se describa el manejo y uso de la aplicación.

4. Hardware

- Descripción: fallo en los requerimientos mínimos de hardware para obtener una ejecución apropiada de la aplicación.

- Prioridad: secundario.
- Responsabilidad: de la Jefatura del Departamento de Computación y Sistemas y del desarrollador del software.
- Contingencia: disponer con anticipación del equipo necesario para la ejecución de la aplicación.

3.3. Modelo de dominio

El modelo de dominio involucra a todos los objetos que tienen una correspondencia directa en el área de la aplicación, se representa en clases conceptuales del dominio del problema, recordando que una clase describe un grupo de objetos con estructura y comportamiento común.

El propósito principal del modelo de dominio es formar una base común de entendimiento del desarrollo y no definir el sistema completo, en otras palabras se trata de realizar un modelo del entorno del software, y no del software. Esto último se realiza dentro de otro componente del proceso que se mostrará mas adelante. A continuación se presentan las clases seleccionadas para representar el modelo de dominio:

- **Sujeto:** representa los sujetos cuyas caras conformarán el total de las disponibles para realizar el proceso de aprendizaje o de reconocimiento.
- **Medios de Captura:** representa los medios necesarios para obtener las imágenes que serán utilizadas para el procesamiento y posterior aprendizaje o reconocimiento.
- **Procesamiento de Imágenes:** contiene los procesos requeridos para el tratamiento de las imágenes capturadas.

- **PCA:** contiene los algoritmos necesarios para la reducción de las imágenes de los rostros, manteniendo sus componentes principales, comprende las representaciones de las imágenes de las caras humanas seleccionadas para el aprendizaje o para el reconocimiento.

- **Aprender:** contiene todos los procesos necesarios para indicarle a las redes neuronales que deben iniciar el aprendizaje, suministrándoles los rostros seleccionados y los datos necesarios para tal fin.

- **Reconocer:** contiene todos los procesos necesarios para indicarle a las redes neuronales artificiales, que deben realizar la tarea de identificación o verificación de los rostros seleccionados.

- **Redes Neuronales:** representa las redes neuronales artificiales, esta clase contiene todos los procesos necesarios para realizar el aprendizaje, que permita efectuar un posterior reconocimiento de sujetos.

- **Resultados:** contiene los resultados obtenidos del aprendizaje o del reconocimiento obtenido.

El modelo de dominio de la aplicación para el reconocimiento facial mediante redes neuronales artificiales, se muestra en la figura 3.1. En él se puede observar que inicialmente las imágenes de los sujetos que han sido capturadas son procesadas para extraer sus características principales. Luego de determinar el proceso a realizar, es decir, si se va a ejecutar una operación para aprender o para reconocer sujetos, se entrenarán las redes neuronales artificiales o se les solicitará un reconocimiento, para después mostrar los resultados generados.

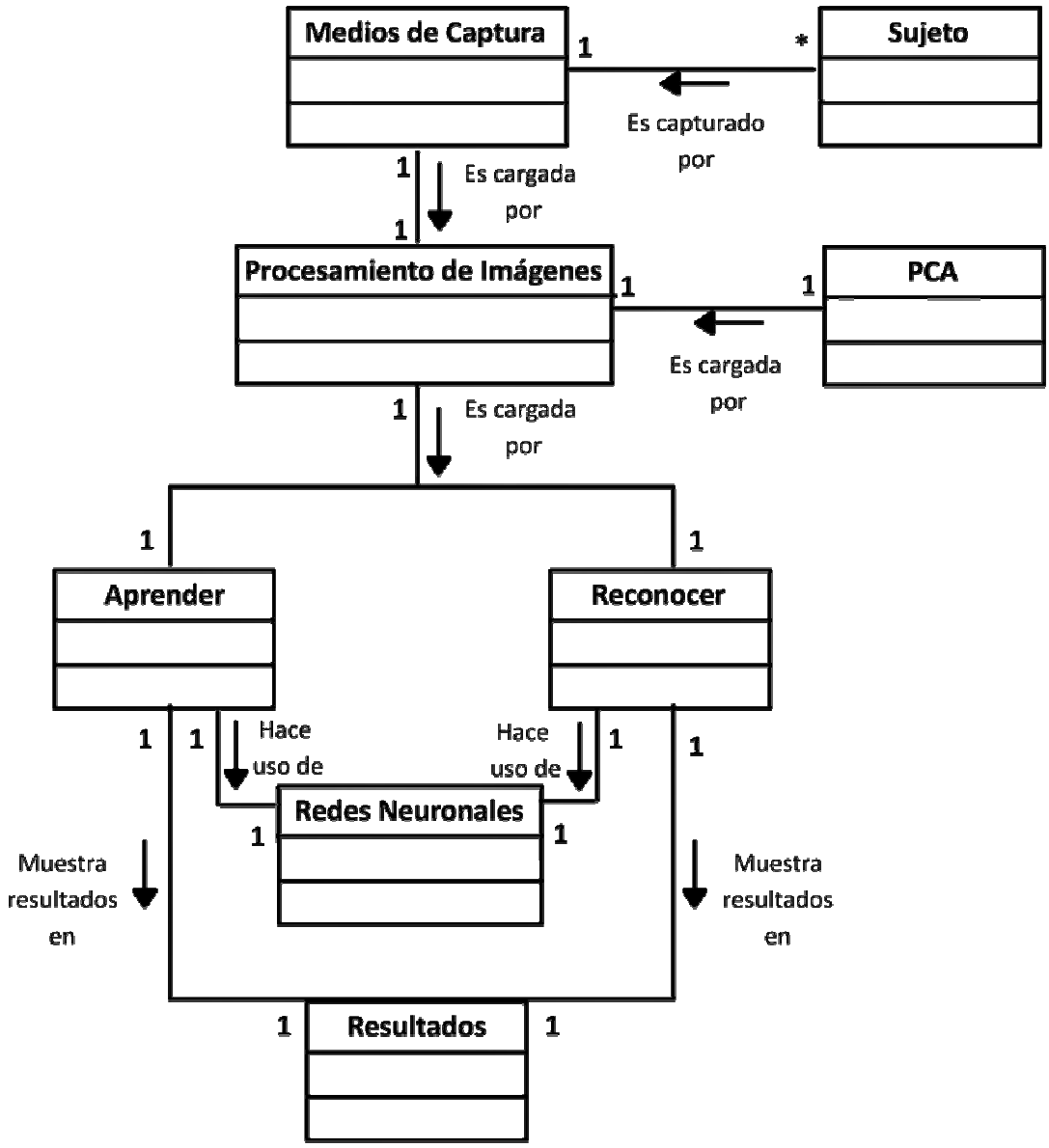

Figura 3.1. *Modelo de dominio de la aplicación. Fuente: Propia.*

3.4. Casos de uso

Una aplicación tiene diversas formas de utilización, cada una de ellas se conoce como caso de uso, es decir, los casos de uso son fragmentos de funcionalidad que la aplicación ofrece para aportar un resultado de valor para sus actores. Entendiéndose por actor, una agrupación uniforme de personas, sistemas o máquinas que interactúan con la aplicación que se está construyendo.

Cada tipo de usuario de la aplicación se representa mediante un actor que define un rol de utilización de la misma. Por lo tanto, antes de avanzar con los casos de uso, es necesario identificar todos los tipos de usuarios diferentes que tiene la aplicación. Los actores modelan el entorno de la aplicación, y los casos de uso la especifican.

3.4.1. Identificación de los actores

Para identificar los actores, se deben determinar las entidades externas que interactúan con el software, en este caso luego de realizar un análisis se reconocieron cuatro (4) actores, los cuales se describen a continuación:

- **Usuario:** representa a cualquier persona que desee utilizar la aplicación. Se encarga de crear o seleccionar una memoria, ya sea para utilizarla, sobre escribirla, o eliminarla. También seleccionan las caras de los sujetos para el aprendizaje o para el reconocimiento. El usuario de igual forma, puede determinar los parámetros para el aprendizaje de las redes neuronales artificiales y si así lo desea, puede modificar la configuración de la aplicación, en cuanto al número disponible de sujetos y de caras por sujeto.
- **Imágenes de selección:** representa los archivos de las caras de los sujetos que son visibles al usuario, para que éste pueda seleccionar las caras que indicaran las imágenes de los rostros que se utilizaran para el aprendizaje o para el reconocimiento.

- **Imágenes de entrada:** representa los archivos de las caras de los sujetos que no son visibles al usuario y que se utilizaran para el aprendizaje o para el reconocimiento. En cualquiera de los dos casos este actor interactúa con la aplicación al cargarse todas las imágenes seleccionadas por el usuario, y así formar los vectores a los que se les extraerá sus componentes principales, para su posterior uso en el proceso de aprendizaje o reconocimiento.
- **Memorias:** representa un archivo digital que contiene resultados generados por aprendizajes anteriores, es decir, en el archivo memorias se encuentran los valores de los pesos de las redes neuronales, los parámetros de entrenamiento, los vectores que representan las imágenes reducidas, de acuerdo a los sujetos seleccionados para ese entrenamiento, entre otros. Interactúa con la aplicación cada vez que se inicia, ya que se cargan las memorias, también cuando se elimina o se genera una nueva como resultado de un aprendizaje.

3.4.2. Identificación de los casos de uso

Luego de haber identificado a los actores de la aplicación, se establece la funcionalidad propia de la aplicación por medio de los casos de uso. Cada caso de uso constituye un flujo completo de eventos que especifican la interacción que toma lugar entre el actor y la aplicación. Al realizar un análisis de los requerimientos y de las funcionalidades de los actores, se identificaron dos (2) flujos o procesos principales, estos son, aprender y reconocer.

El proceso aprender consta de los siguientes sub procesos: cargar imágenes para selección, cargar memorias (contiene aprendizajes anteriores); elegir opciones de memoria (en la que se puede crear una memoria nueva o seleccionar una existente, ya sea para optimizarla, cargar el entrenamiento que posee o eliminarla); seleccionar rostros para entrenamiento, modificar parámetros de aprendizaje (que contiene ajustar

parámetros MLP y ajustar parámetros RBF) para adaptar los valores iniciales de las redes neuronales de acuerdo con los rostros seleccionados para el entrenamiento; aplicar PCA (este sub proceso carga las imágenes que se utilizarán para la obtención de los vectores que representarán rostros reducidos, también se encarga de obtener las salidas esperadas de los patrones de entrenamiento); obtener entrenamiento (que hace uso de las redes neuronales Función de Base Radial y Perceptrón Multicapa); guardar memoria (guarda una memoria nueva o actualiza una existente en caso de que se obtengan mejores resultados).

El proceso reconocer consta de sub procesos como: cargar imágenes para selección, seleccionar rostros para reconocer, cargar rostros reducidos de PCA (no es necesario reducir las imágenes de los rostros nuevamente, debido a que todas las reducciones se llevaron a cabo en el proceso aprender); y por último obtener reconocimiento (hace llamados a las redes Perceptrón Multicapa y Función de Base Radial) para obtener los resultados de la identificación o verificación.

También se identificó un proceso secundario, denominado configurar, en éste el usuario puede conocer la ubicación de las imágenes de los rostros utilizadas para la selección y de las imágenes de entrada, igualmente se puede cambiar el número de sujetos y el número de caras por sujeto disponibles para la aplicación.

En la figura 3.2 se muestra el modelo de casos de uso, con los procesos y sub procesos identificados.

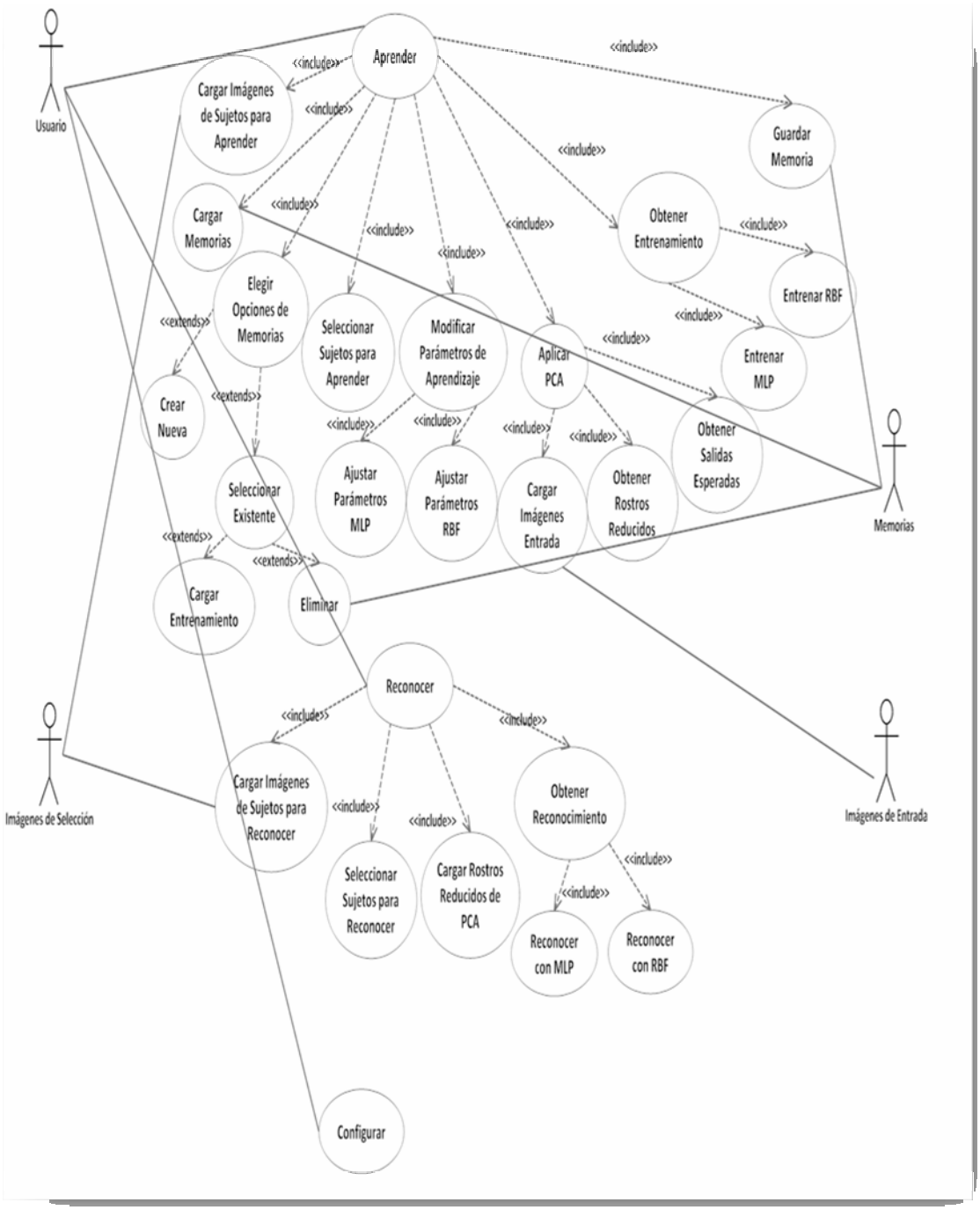

Figura 3.2. *Diagrama de casos de uso. Fuente: Propia*.

3.4.3. Casos de uso detallados

Los casos de uso para la aplicación reconocedora de rostros se describen a continuación en las tablas 3.1, 3.2 y 3.3:

Tabla 3.1. *Caso de uso aprender. (1/2)*

Actor principal	Usuario
Actor secundario	Imágenes de selección, Imágenes de entrada, Memorias.
Descripción	Describe el proceso de aprendizaje de las caras de los sujetos seleccionados por el usuario. Una vez reducidas las caras, se inicia el aprendizaje de las redes neuronales y luego de esto, se guarda el resultado en una memoria.
Flujo principal	1. Cargar las imágenes de los sujetos para que puedan seleccionarse. 2. Cargar las memorias de aprendizajes anteriores. 3. Elegir si se quiere crear una memoria nueva o utilizar una existente. 4. Determinar las caras de los sujetos a utilizar para el aprendizaje. 5. Modificar los parámetros de las redes neuronales, para un mejor rendimiento. 6. Reducir las caras seleccionadas aplicando el análisis de componentes principales. 7. Entrenar las redes neuronales artificiales, Perceptrón Multicapa y Función de Base Radial, con los sujetos seleccionados.

Fuente: Propia.

Tabla 3.1. *Caso de uso aprender.(2/2)*

	8. Guardar la memoria generada, ya sea, actualizando la memoria seleccionada, en el caso de hallar mejores resultados, o almacenando los datos de la memoria creada.

Fuente: Propia.

Tabla 3.2. *Caso de uso reconocer.*

Actor principal	Usuario
Actor secundario	Imágenes de selección, Imágenes de entrada.
Descripción	Describe el proceso de reconocimiento, que realizarán las redes neuronales artificiales, de las caras de los sujetos seleccionados para tal fin.
Flujo principal	1. Cargar las imágenes de los sujetos para que puedan seleccionarse en el reconocimiento. 2. Seleccionar las caras de los sujetos que se deseen reconocer. 3. Cargar los rostros reducidos de los sujetos seleccionados, no es necesario extraer de nuevo las características principales, ya que esto se realizó en la etapa de aprendizaje. 4. Enviarle a cada una de las redes neuronales las caras de los sujetos a reconocer, para que puedan determinar si saben quienes son estos sujetos o no.

Fuente: Propia.

Tabla 3.3. *Caso de uso configurar.*

Actor principal	Usuario
Actor secundario	No tiene.
Descripción	Describe el proceso de configuración de la aplicación.
Flujo principal	1. El usuario podrá conocer la ubicación y el nombre de las carpetas que contienen las imágenes de los sujetos que se utilizaran para la selección y de los que se emplearán para el entrenamiento o para el reconocimiento. 2. Informar al usuario a cerca de la resolución que deben tener las imágenes de las caras de los sujetos. 3. Si el usuario lo desea puede cambiar el número de sujetos y/o el número de caras por sujeto. 4. Finalmente si el usuario decide realizar cambios deberá guardarlos presionando en el botón configurar.

Fuente: Propia.

3.5. Diagrama de clases de análisis

Este diagrama se puede considerar como una estructura de clasificadores (clases de análisis) y relaciones entre ellas. Las clases de análisis se utilizan para realizar una descripción abstracta de la realización de los casos de uso del modelo de casos de uso.

Para la realización del diagrama de clases de análisis es necesario conocer en cual de los estereotipos básicos encajan cada una de las clases que lo conforman, para poder representarlas, estas pueden ser: de interfaz, de control, o entidad. En general,

se desea asignar la funcionalidad global del caso de uso, correspondiente a lo especificado para la aplicación, a las clases de control. Esta funcionalidad depende y afecta al resto de las clases dentro del caso de uso. Por otro lado, las clases interfaz y entidad deben contener funcionalidad local, limitando su efecto en las demás clases.

3.5.1. Diagrama de clases de análisis para caso de uso de la aplicación

Partiendo de los casos de uso mostrados anteriormente y luego de analizar las interacciones entre éstos y los actores, y de las operaciones necesarias para lograr la funcionalidad óptima de la aplicación, se identificaron tres (3) procesos, los cuales son iniciados por el actor "Usuario".

- Aprender: primero el usuario interactúa con la clase de interfaz "IU Aprender", en la que podrá seleccionar las caras de los sujetos, realizar operaciones con las memorias, o modificar los parámetros de aprendizaje de las redes neuronales para luego entrenarlas. Luego se encuentra la clase de control "Gestor Aprender", que se encarga de administrar todos los procedimientos necesarios para indicarles a las otras clases el momento en el que deben realizar sus funciones, así mismo controla las caras de los sujetos que se le están mostrando al usuario. La clase "Gestor Cargar Memorias" como su nombre lo indica, lleva a cabo todos los procesos necesarios para cargar las memorias, de un archivo, generadas anteriormente. También se encuentra la clase de control "Gestor Memorias", que transforma todos los datos cargados a un formato adecuado a las solicitudes a las que debe responder. La clase de control "Aplicar Algoritmo PCA" recibe los indicadores de las caras de los sujetos seleccionados por el usuario, para luego cargar las imágenes de entrada y seguidamente extraerles sus características principales. Las clases de control "Aplicar Algoritmo MLP" y "Aplicar Algoritmo RBF" se encargan de implementar las redes neuronales artificiales Perceptrón Multicapa y

Función de Base Radial, estas redes son las que van a entrenarse con las caras de los sujetos seleccionados. Al final de este proceso se encuentra la clase de control, "Gestor Guardar Memoria", que tiene por tarea almacenar la memoria con todos los datos que la conforman. En el proceso aprender, se encuentran también las clases de entidad "Imágenes de Entrada", "Imágenes de Selección" y "Memorias".

- Reconocer: en este proceso al igual que en el anterior, el usuario interactúa con la clase de interfaz "IU Reconocer" para elegir lo sujetos y las caras correspondientes para identificarlos o para verificarlos, para luego realizar el reconocimiento. También se encuentra la clase de control "Gestor Reconocer", que controla todo lo necesario para llevar a cabo el reconocimiento, cargando y administrando las imágenes de selección, permitiéndole a el usuario escoger los sujetos para el reconocimiento, una vez seleccionados los sujetos, esta clase se encarga de cargar los vectores que contienen dichas imágenes reducidas por el PCA, para luego enviárselos a las redes neuronales artificiales y así, efectuar el reconocimiento. Las clases de control "Aplicar Algoritmo MLP" y "Aplicar Algoritmo RBF" contienen las redes neuronales Perceptrón Multicapa y Función de Base Radial, para efectuar el reconocimiento respectivo. Con este proceso se comunican la clase de entidad "Imágenes de Selección" para que el usuario pueda elegir las caras de los sujetos a reconocer.

- Configurar: en este proceso el usuario puede configurar datos de la aplicación como, número de sujetos y número de caras por sujeto, y puede informarse acerca de la resolución en la que deben estar las imágenes en caso de querer agregar otras. Esto se muestra en la clase de interfaz "IU Configurar" y de todos los procedimientos para llevar a cabo la configuración se encarga la clase de control "Gestor Configurar". La clase

de control “Gestor Almacenar Configuración” se encarga de guardar los cambios, comunicándose con la clase de entidad “Memorias”, que es donde se almacenan dichos datos.

Todos los procesos descritos anteriormente se muestran en la figura 3.3.

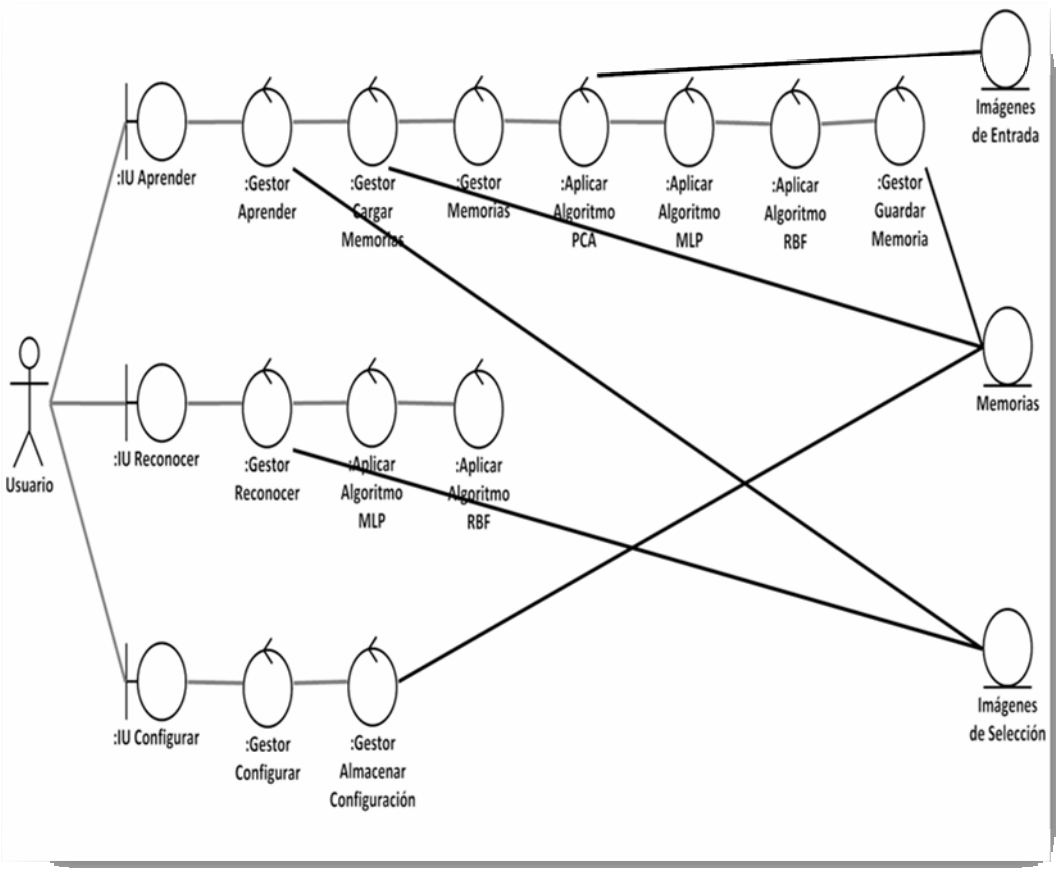

Figura 3.3. *Diagrama de clases de análisis. Fuente: Propia.*

3.6. Diagrama de actividades

Los diagramas de actividades representan una serie de acciones. Una acción consiste en asignar un valor a un atributo, crear o destruir un objeto, efectuar una operación, enviar una señal a otro objeto o a uno mismo, etc. La finalidad de estos diagramas es

modelar los aspectos dinámicos de la aplicación, para especificar los procesos del software.

3.6.1. Actividad gestión aprender

Esta actividad se encarga del aprendizaje de las caras de sujetos, ya sea para optimizar los datos de una memoria seleccionada, o para crear una nueva. Inicia con la selección de las opciones de memoria, si se crea una nueva entonces se deben seleccionar las caras de los sujetos para el aprendizaje, para luego obtener los vectores que representan los rostros reducidos al aplicar el algoritmo PCA, y así poder entrenar las redes neuronales con los algoritmos MLP (perceptrón multicapa) y RBF (función de base radial), para después guardar la memoria obtenida. En caso de seleccionar una memoria para actualizarla, no es necesario aplicar el algoritmo PCA, debido a que la memoria ya contiene los rostros reducidos, pero sí se realiza el entrenamiento de las redes neuronales artificiales, haciendo uso de los algoritmos correspondientes, MLP y RBF, sí los datos obtenidos resultan ser mejores que los que contiene actualmente la memoria (esto es, sí alguna de las redes neuronales obtuvo un mejor entrenamiento, para ese mismo patrón de entrada), entonces se actualizarán sólo los datos óptimos, en caso contrario la memoria queda igual. El diagrama que contiene todo lo mencionado se muestra en la figura 3.4.

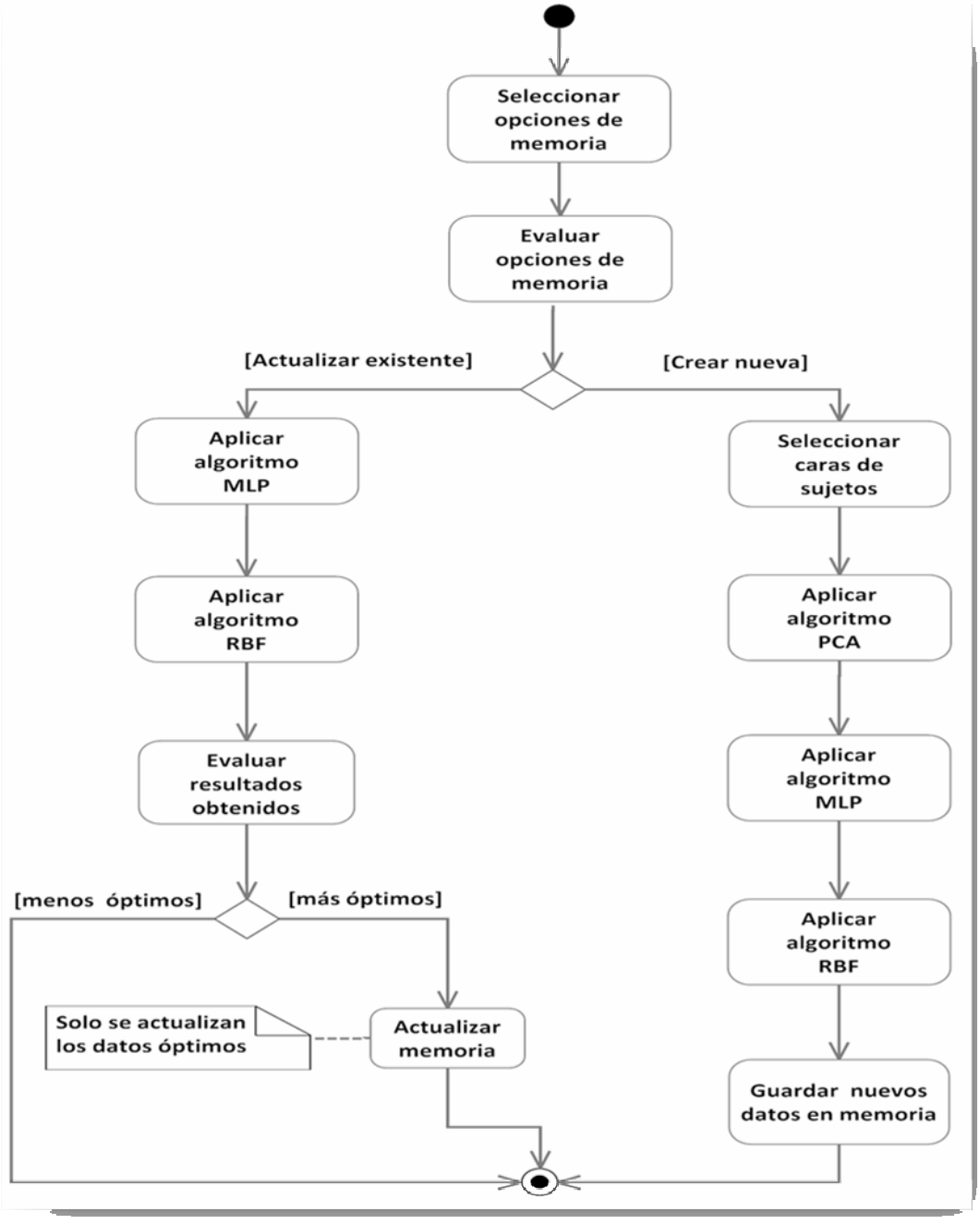

Figura 3.4. *Diagrama de actividades: Gestión aprender. Fuente: Propia.*

3.6.2. Actividad gestión reconocer

Realiza el reconocimiento de las caras de los sujetos elegidos. Inicia con la selección de las caras de los sujetos, seguido de la obtención de los vectores de entrada, que se encuentran en PCA y que representan las caras seleccionadas, para luego aplicar los algoritmos que implementan las redes neuronales artificiales MLP y RBF. En la figura 3.5, se muestra el diagrama de actividades de la gestión reconocer.

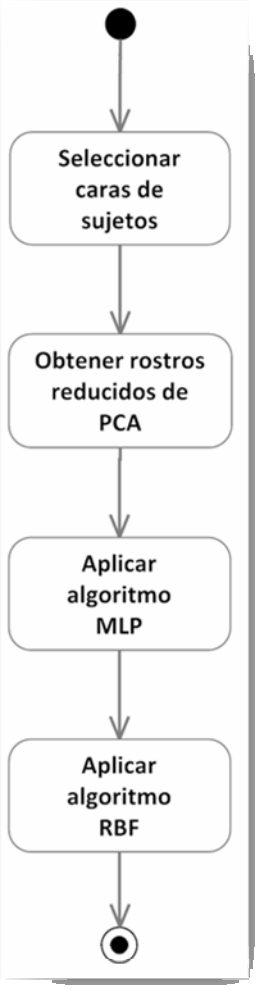

Figura 3.5. *Diagrama de actividades: Gestión reconocer. Fuente: Propia.*

3.7. Identificación de conceptos u objetos

Los conceptos u objetos individuales que conforman la arquitectura de la aplicación, se obtienen de la especificación de requerimientos o más concretamente, del modelo de casos de uso. Esto con el fin de formar el modelo conceptual, que muestra los conceptos significativos en la descripción del problema.

Para la obtención de los conceptos se elaboró una lista de frases nominales, a partir de las descripciones textuales de los casos expandidos de uso. Los conceptos identificados fueron los siguientes:

- InterfazPrincipal.
- InterfazAprender.
- InterfazReconocer.
- InterfazConfigurar.
- AlgoritmoPCA.
- AlgoritmoMLP.
- AlgoritmoRBF.
- Memorias.

El primer concepto InterfazPrincipal, se refiere a la interfaz gráfica que se encarga de administrar las demás interfaces y sobre la que estarán contenidas. El segundo concepto InterfazAprender, representa a la interfaz gráfica que se encarga de interactuar con el usuario para que pueda seleccionar los sujetos a aprender, seleccionar las opciones de memoria y modificar los parámetros de las redes neuronales antes de iniciar el entrenamiento. El concepto InterfazReconocer,

describe la interfaz gráfica con la que el usuario interactuará para seleccionar los sujetos a reconocer. InterfazConfigurar que constituye la interfaz gráfica para que el usuario pueda hacer ajustes en cuanto al número de sujetos y de caras por sujetos disponibles. AlgoritmoPCA, es otro de los conceptos identificados, se encarga de implementar la técnica de análisis de componentes principales, para el procesamiento de las imágenes. Los conceptos AlgoritmoMLP y AlgoritmoRBF describen las redes neuronales perceptrón multicapa y función de base radial. Por último el concepto Memorias, que conforma todo lo necesario para el tratamiento de las memorias generadas por algún aprendizaje.

La identificación de conceptos forma parte de una investigación necesaria para la realización de la aplicación, es por ello que, el modelo conceptual es un artefacto importante que debe crearse durante el análisis.

CAPITULO 4

FASE DE DISEÑO

En esta fase la arquitectura obtenida durante el análisis se transforma en una especializada, donde se considera el ambiente de implantación particular de la aplicación. De la fase de diseño resultan especificaciones muy detalladas de todos los objetos, incluyendo sus operaciones y atributos. En otras palabras, la fase de diseño debe definir todo lo necesario para alcanzar el código final.

4.1. Diagrama de clases de diseño

Describe las especificaciones de las clases de software y de las interfaces en una aplicación. En cada clase se muestra información como, el nombre de la clase, los atributos que posee, y los métodos que utiliza, con sus respectivos modificadores de acceso. Sí el método es público el nombre del método estará precedido por un signo más (+), sí es privado entonces el signo debe ser menos (-) y sí es protegido entonces el símbolo será numeral (#). También existen otras notaciones, que junto a las mencionadas, se muestran en la figura 4.1.

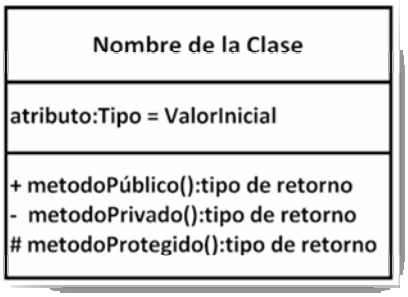

Figura 4.1. *Notación de los detalles de una clase de diseño. Fuente: Propia.*

4.1.1. Diagrama de clases de diseño de la aplicación

A continuación se describen las clases de la aplicación para el reconocimiento facial, se mostrarán en tres grupos, primero las clases que conforman la interfaz de usuario, luego las clases que contienen los algoritmos necesarios para efectuar el aprendizaje y el reconocimiento, y por último, las clases requeridas para realizar el resto de las operaciones que complementen la funcionalidad de la aplicación.

La interfaz de usuario, está conformada por las clases **Interfaz**, **PanelAprender**, **PanelAprendiendo**, **PanelReconocer**, **PanelReconociendo** y por último la clase **PanelConfigurar**, todas estas clases hacen uso de los paquetes java.awt y javax.swing para poder hacer uso de todo lo que respecta al entorno gráfico. La clase **Interfaz**, se encarga del manejo de toda la interfaz de usuario, contiene lo necesario para que el usuario se desplace de un apartado a otro, con apartados se hace referencia a los procesos aprender, reconocer y configurar. La clase **PanelAprender**, contiene todo lo requerido para que el usuario pueda seleccionar las caras de los sujetos que se utilizarán para el entrenamiento, el usuario también podrá realizar operaciones con las memorias (crear una nueva, utilizar una existente o eliminar alguna memoria), y modificar parámetros de las redes neuronales artificiales, tales como, número de neuronas ocultas, factor de aprendizaje y número de capas ocultas (sólo para el perceptrón multicapa). La clase **PanelAprendiendo**, comprende los atributos y métodos fundamentales para mostrar al usuario el resultado del aprendizaje realizado. **PanelReconocer** es la clase que abarca todo lo elemental para que el usuario, pueda seleccionar los rostros a reconocer, ya sea para realizar la identificación y/o la verificación. En la clase **PanelReconociendo**, se muestra al usuario los resultados del reconocimiento determinado en la clase anterior. El usuario también podrá obtener información y realizar configuraciones, en cuanto a la cantidad de sujetos y de caras por sujeto, en la interfaz proporcionada por la clase **PanelConfigurar**. Estas clases se muestran en la figura 4.2.

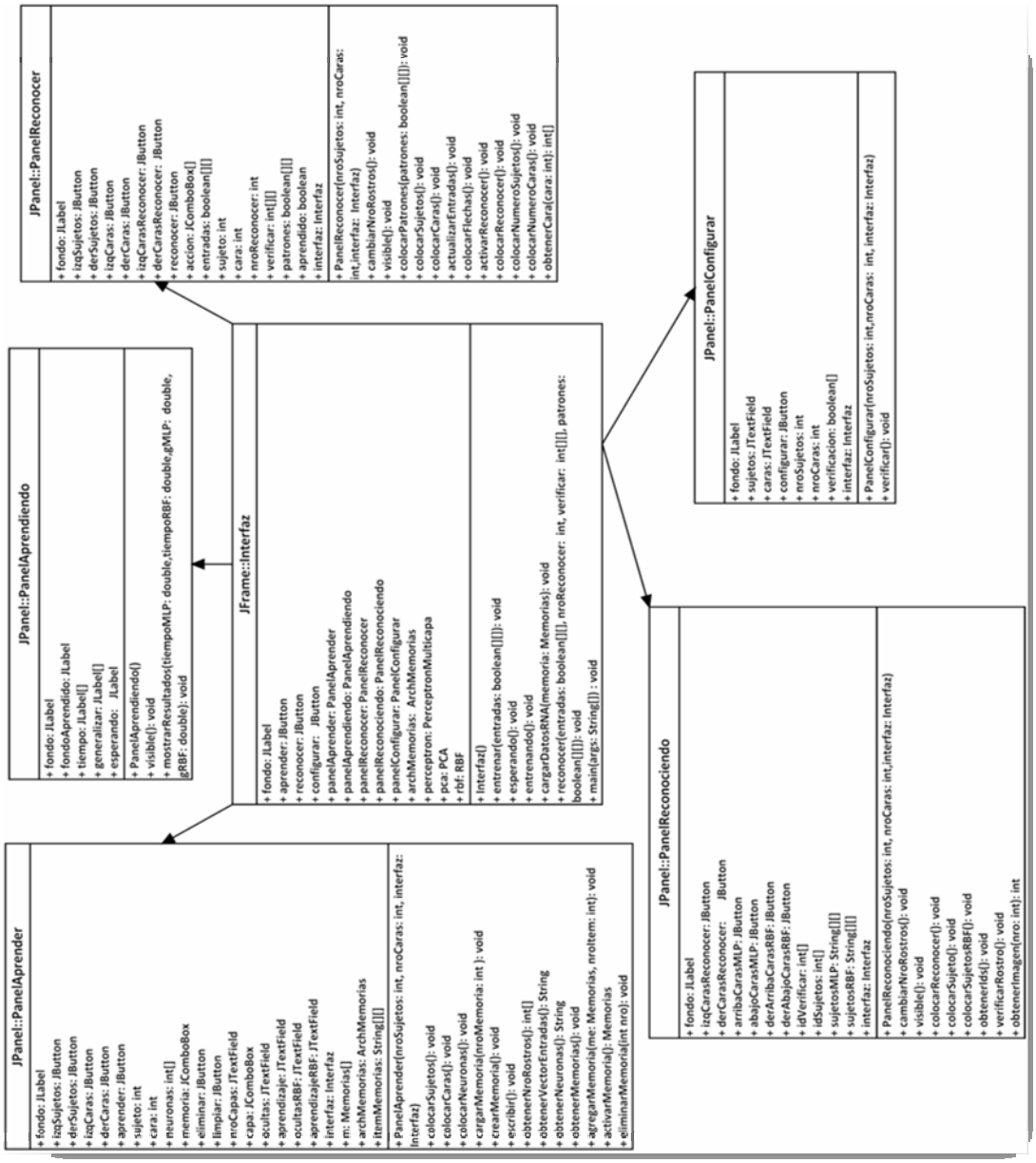

Figura 4.2. *Diagrama de clases de diseño de la aplicación – Parte 1. Fuente: Propia.*

Los algoritmos que hacen posible los procesos de aprender y reconocer están contenidos en las clases **PCA**, **PerceptronMulticapa**, **RBF** y **Pixel**.

PCA, contiene la implementación del análisis de componentes principales, y obtiene los vectores columna necesarios para su funcionamiento, a través de la clase **Pixel**, que lee y adapta las imágenes seleccionadas, para el aprendizaje o el reconocimiento. En las clases **PerceptronMulticapa** y **RBF**, se encuentran las redes neuronales artificiales, que reciben los patrones para entrenar o reconocer luego de ser reducidos por **PCA**. Las clases mencionadas están contenidas en la figura 4.3.

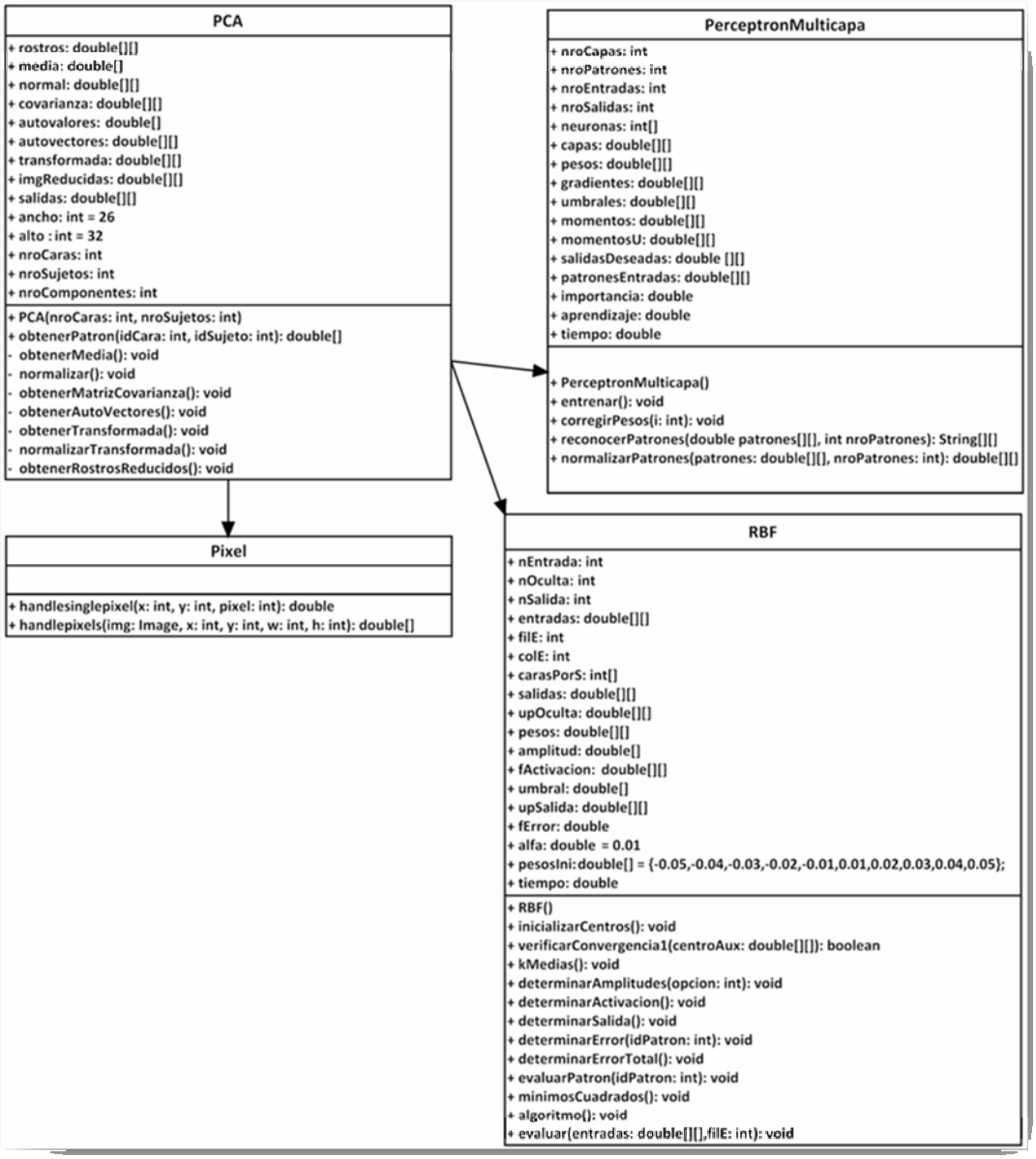

Figura 4.3. *Diagrama de clases de diseño de la aplicación – Parte 2.*

Fuente: Propia.

En la figura 4.4, se encuentran las clases necesarias para completar la funcionalidad de la aplicación, las cuales son, **ArchMemorias**, para el tratamiento del archivo que contiene las memorias; **Memorias**, para el almacenamiento de todas las memorias existentes.

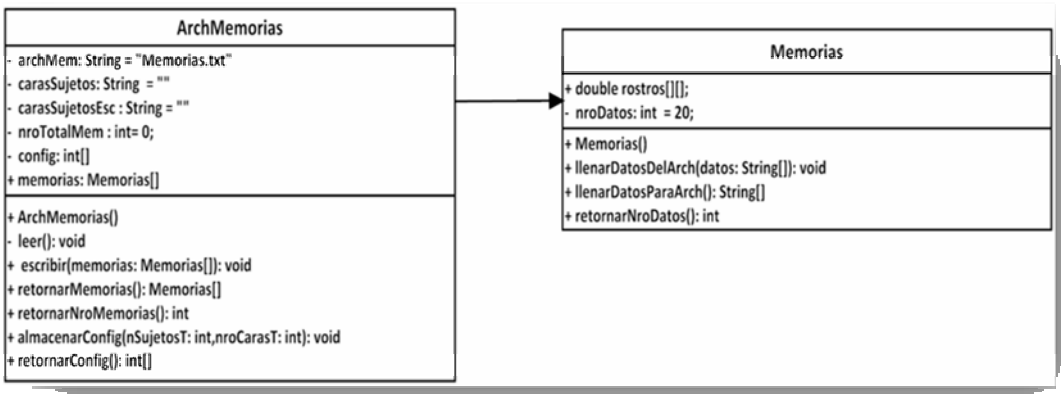

Figura 4.4. *Diagrama de clases de diseño de la aplicación – Parte 3.*

Fuente: Propia.

4.2. Diagrama de paquetes

Los diagramas de paquetes permiten visualizar un conjunto de paquetes y las relaciones existentes entre ellos, contienen todos aquellos paquetes que forman parte de una solución. Los usos más comunes para los diagramas de paquetes son para organizar diagramas de casos de uso y diagramas de clase, a pesar de que el uso de los diagramas de paquetes no es limitado a estos elementos UML.

Los paquetes permiten organizar las clases de un modelo, contienen clases con funciones similares, o incluso pueden tener anidados otros paquetes, así como paquetes subordinados.

Los paquetes se representan como carpetas y contienen los elementos que comparten un espacio de nombre; todos los elementos dentro de un paquete deben tener un identificador único. El paquete debe mostrar el nombre del paquete y puede opcionalmente mostrar los elementos dentro del paquete en compartimientos extras.

En la figura 4.5 se muestra el diagrama de paquetes de la aplicación para el reconocimiento facial, donde se encuentran las clases que conforman la aplicación, las imágenes de los rostros necesarias para su funcionamiento y las utilizadas en la interfaz de usuario.

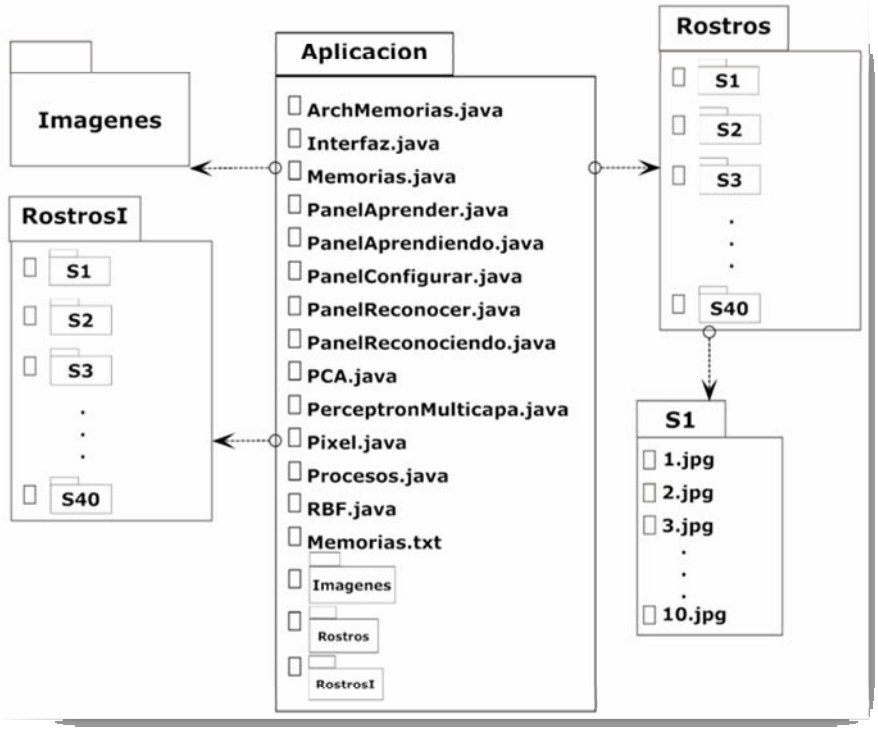

Figura 4.5. *Diagrama de paquetes de la aplicación. Fuente: Propia.*

4.3. Diagrama de capas

El diagrama de capas es una forma de estructurar los componentes del software, cada una de las capas que conforman el diagrama, contiene uno o varios elementos, aquellos que se encuentren en una misma capa podrán comunicarse entre sí, pero se presentan variantes en cuanto a las comunicaciones permitidas entre los elementos de capas diferentes. Existen varias alternativas para el diseño de este diagrama, en este caso se utilizó la arquitectura Bottom-up, en el que las capas inferiores se comunican con las superiores, es decir, se parte del nivel más bajo de la arquitectura del software. A continuación se presentan las capas que constituyen el diagrama:

- **Capa Específica de la Aplicación**: contiene los paquetes internos de la aplicación, que son, Imágenes, Rostros y RostrosI.

- **Capa General de la Aplicación**: esta conformada por el paquete principal de la aplicación.
- **Capa Intermedia**: contiene la máquina virtual de java JRE (Java Runtime Environment), que interpreta y ejecuta la aplicación.
- **Capa de Software**: contiene el sistema operativo en donde se puede utilizar la aplicación, y ya que ésta es multiplataforma, puede ser interpretada en diferentes entornos como en Windows, Linux, Mac OS, Solaris, etc.

El diagrama de capas para la aplicación para el reconocimiento facial, se encuentra en la figura 4.6.

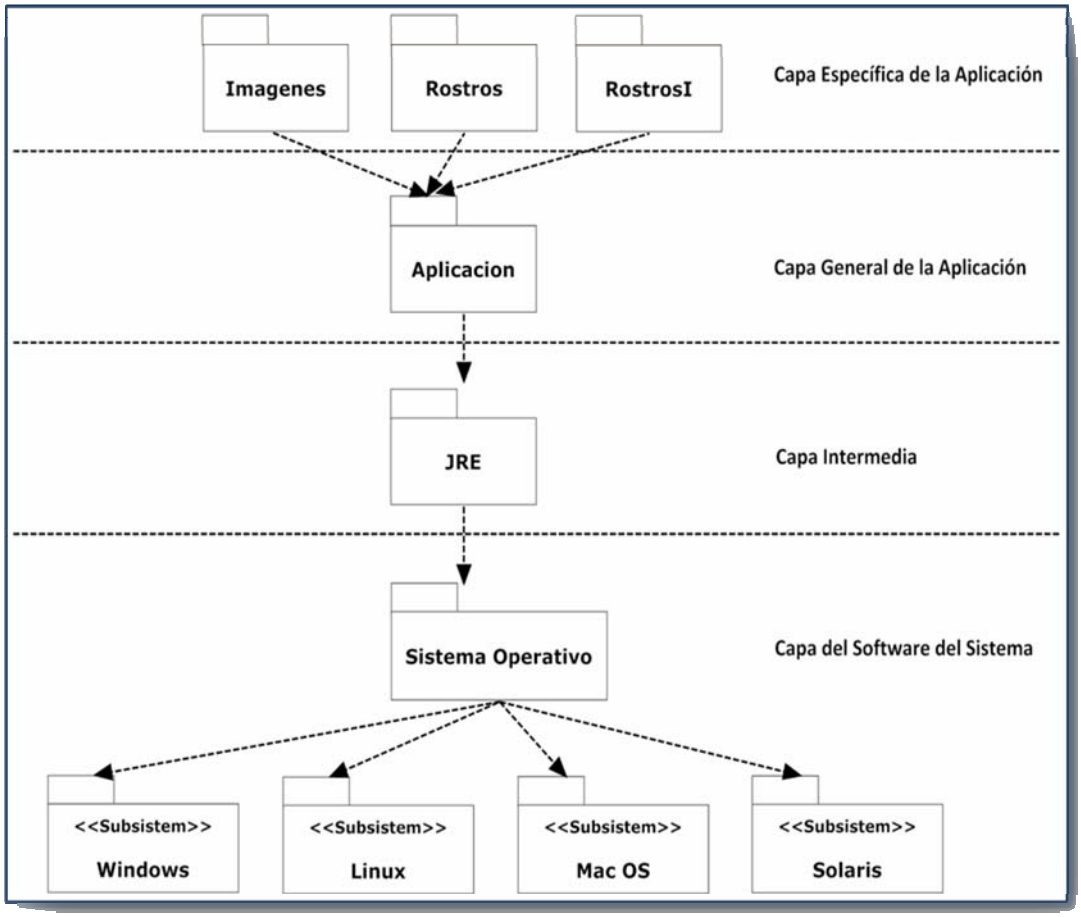

Figura 4.6. *Diagrama de capas de la aplicación. Fuente: Propia.*

4.4. Diagrama de secuencia

El diagrama de secuencia se enfoca en la manera en que los objetos se envían mensajes a través del tiempo. En cuanto a la notación, cada objeto se representa dentro de un rectángulo en la parte superior del diagrama. El nombre del objeto se coloca dentro del rectángulo. Todos los objetos tienen una vida útil, que representa el paso del tiempo, se denota como una línea punteada que se extiende de manera vertical desde un determinado objeto. Cabe destacar que las acciones se llevan a cabo en orden cronológico, es decir, de arriba hacia abajo.

Un mensaje entre dos objetos dentro de un diagrama de secuencia se representa mediante una línea con punta de flecha que se extiende desde el objeto que envía el mensaje hasta el objeto que lo recibe.

En las siguientes secciones se mostrarán los diagramas de secuencia que describen los procesos principales y sub procesos de cada caso de uso.

4.4.1. Diagrama de secuencia para caso de uso cargar imágenes de sujetos para aprender

El diagrama de secuencia que detalla el caso de uso cargar imágenes de sujetos para aprender, se muestra en la figura 4.7. Esta secuencia de eventos se inicia en el proceso aprender, y se especifica a continuación:

- El actor **Usuario** realiza la solicitud para aprender al objeto **obj**, quien es una instancia de la clase **Interfaz**.
- El objeto **obj** se comunica con el objeto **panelAprender,** de la clase **PanelAprender**, para indicarle que ya es visible al usuario.
- El actor **Imágenes de Selección** le envía al objeto **panelAprender** las imágenes de los distintos sujetos y las caras por sujeto, para que el objeto coloque estas imágenes a través de los métodos **colocarSujetos** y **colocarCaras**.
- La interfaz le muestra al usuario los sujetos y las caras disponibles para el aprendizaje.

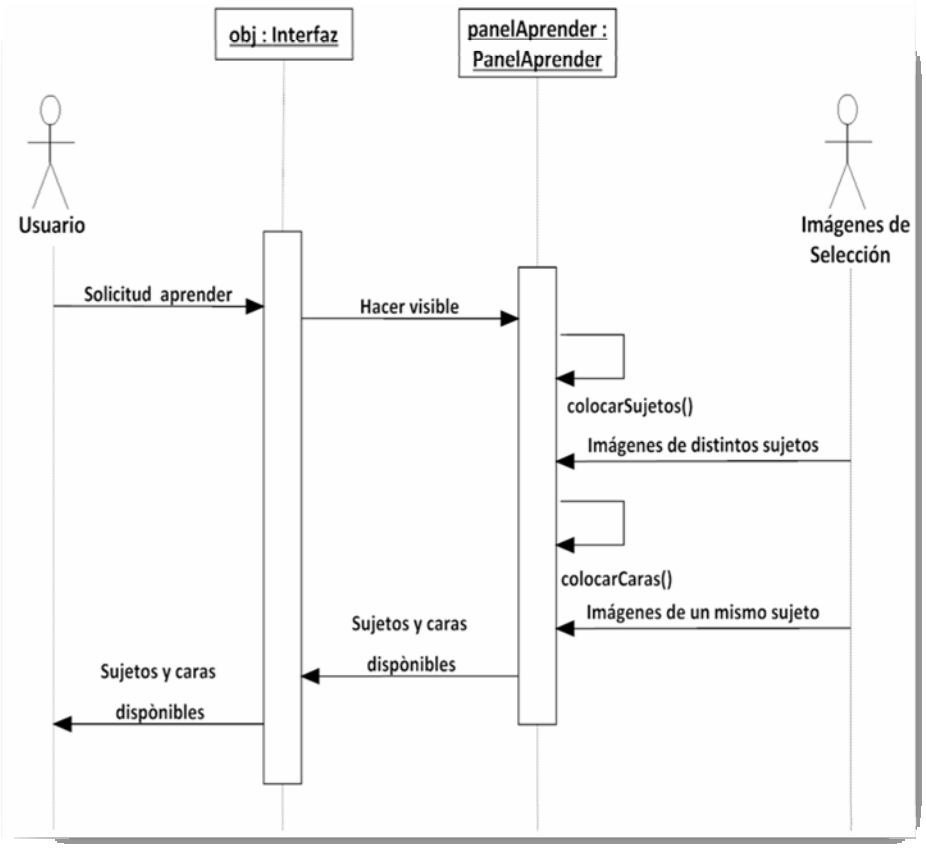

Figura 4.7. *Diagrama de secuencia para caso de uso cargar imágenes de sujetos para aprender. Fuente: Propia.*

4.4.2. Diagrama de secuencia para caso de uso cargar memorias

En este diagrama se muestra la secuencia de eventos necesarios para que se puedan cargar las memorias existentes del archivo que las contiene, se detallan a continuación:

- El actor **Usuario** realiza la solicitud para aprender al objeto **obj**, quien es una instancia de la clase **Interfaz**.

- El objeto **obj** se comunica con el objeto **panelAprender,** de la clase **PanelAprender**, para indicarle que ya es visible al usuario.
- El objeto **panelAprender** le solicita al objeto **archMemorias** que le retorne las memorias disponibles.
- El actor **Memorias** le envía al objeto **archMemorias** los datos del archivo que solicita.
- El objeto **archMemorias** hace uso del objeto **memorias** para almacenar los datos del archivo.
- El objeto **panelAprender** recibe del objeto **archMemorias** las memorias disponibles para luego mostrárselas al usuario a través de la interfaz.

En la figura 4.8, esta contenido el diagrama de secuencia descrito anteriormente, para el caso de uso cargar memorias.

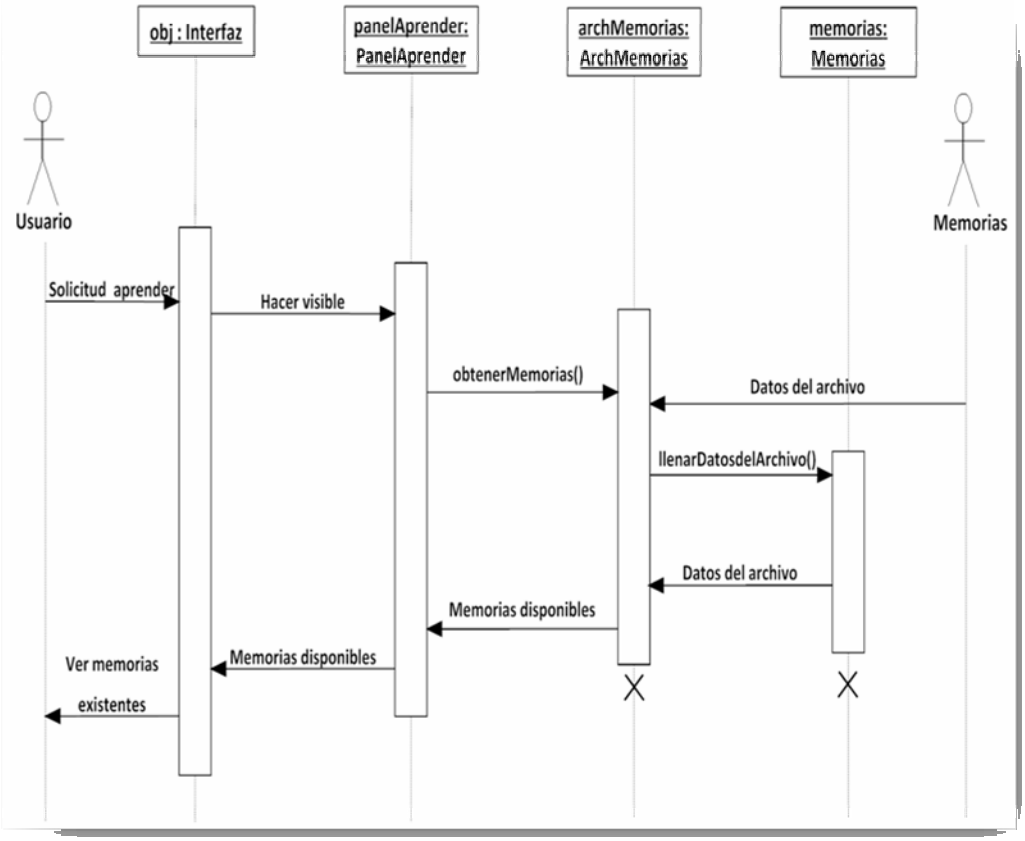

Figura 4.8. *Diagrama de secuencia para caso de uso cargar memorias. Fuente: Propia.*

4.4.3. Diagrama de secuencia para caso de uso elegir opciones de memorias

El diagrama de de secuencia para el caso de uso elegir opciones de memorias, se encuentra contenido en la figura 4.9, en él se muestra cómo se comunican los objetos para llevar a cabo el proceso correspondiente, la secuencia de eventos que conforman el caso de uso señalado, se muestra a continuación:

- El actor **Usuario** realiza la solicitud para realizar el aprendizaje a la interfaz de usuario principal.

- El objeto **obj** le indica al objeto de la clase **PanelAprender** que ya es visible al usuario.
- El actor **Usuario** decide crear una nueva memoria, para lo cual realiza la solicitud al objeto de la clase **Interfaz**, seguido de esto, el objeto **obj** le solicita al objeto **panelAprender** que debe crear una nueva memoria, entonces el objeto **panelAprender** acepta la solicitud actualizando las opciones elegidas, para que luego el **Usuario** pueda observar que su elección ha sido aceptada.
- El actor **Usuario** también puede elegir una memoria existente, haciéndole la solicitud al objeto **obj** de la clase **Interfaz**, la cual le envía la petición al objeto **panelAprender**, para que éste cargue la memoria seleccionada, y la solicitud pueda ser aceptada, mostrándole al **Usuario** los datos de la memoria que ha seleccionado.
- Por otro lado el **Usuario** también puede eliminar una memoria seleccionada, haciendo la solicitud al objeto **obj**, para que a su vez, éste le envíe la solicitud al objeto **panelAprender**, quien le indicará al objeto **archMemorias** que debe eliminar la memoria seleccionada, y la elimina modificando el contenido del actor **Memorias**. Luego el objeto **panelAprender**, elimina la memoria de la interfaz de usuario, para que el **Usuario** pueda ver las memorias disponibles.

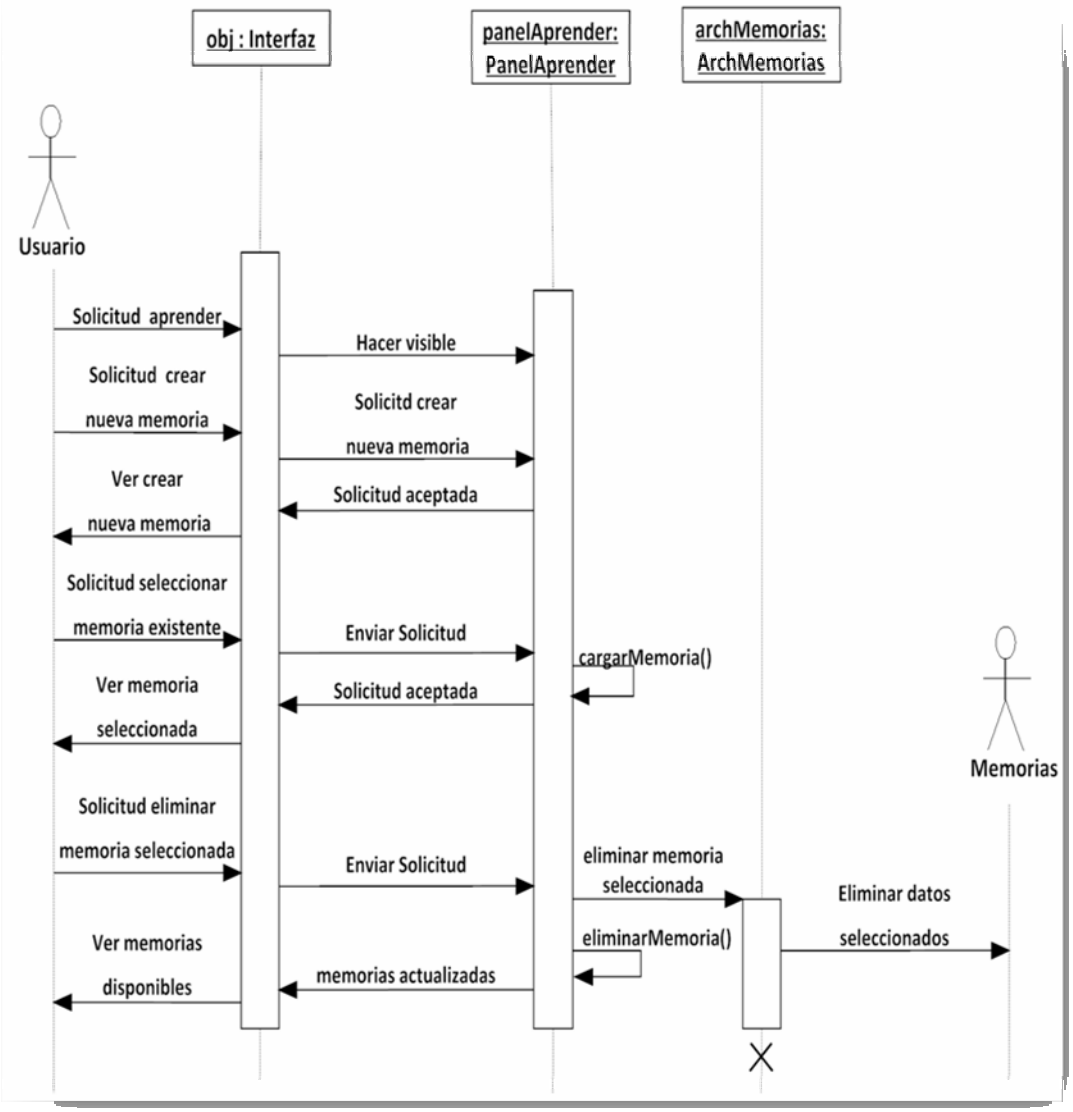

Figura 4.9. *Diagrama de secuencia para caso de uso elegir opciones de memorias. Fuente: Propia.*

4.4.4. Diagrama de secuencia para caso de uso seleccionar sujetos para aprender

En este diagrama se muestran los eventos necesarios para llevar a cabo el caso de uso seleccionar sujetos. La secuencia para lograr el sub proceso se muestra a continuación:

- El actor **Usuario** realiza la solicitud al objeto **obj**, con el fin de seleccionar los sujetos para el aprendizaje.
- El objeto **obj** envía la solicitud a **panelAprender**, para que éste almacene la selección realizada por el **Usuario** mediante el método actualizarEntradas().
- El objeto **panelAprender** acepta la solicitud mostrando los sujetos seleccionados, para que el **Usuario** vea su selección a través de la **Interfaz**.

El diagrama de secuencia para el caso de uso seleccionar sujetos para aprender se puede observar en la figura 4.10.

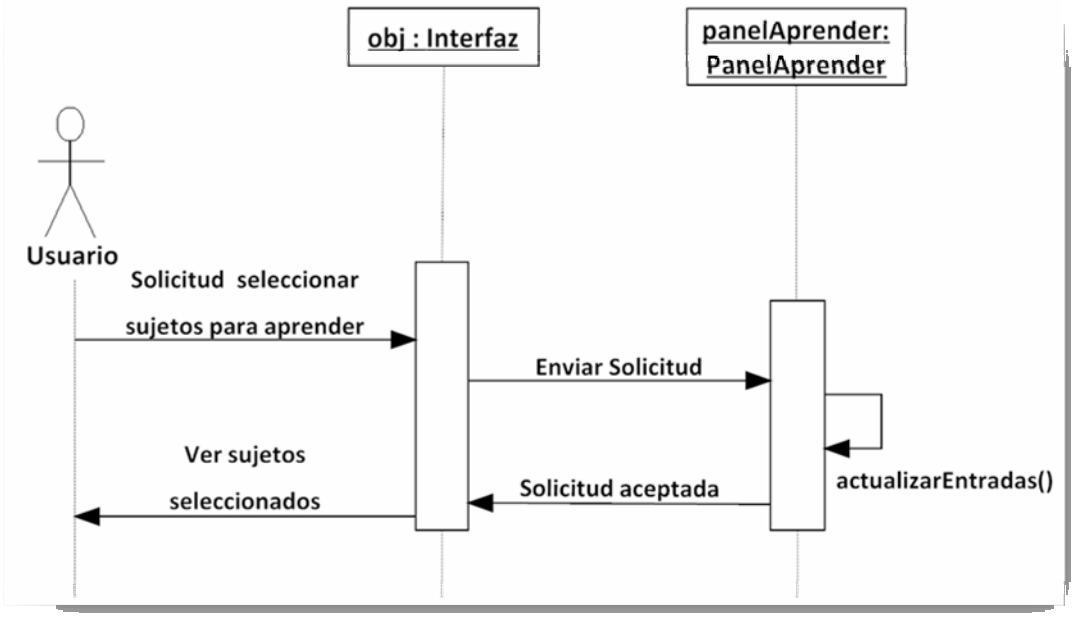

Figura 4.10. *Diagrama de secuencia para caso de uso seleccionar sujetos para aprender. Fuente: Propia.*

4.4.5. Diagrama de secuencia para caso de uso modificar parámetros de aprendizaje

El diagrama de secuencia del caso de uso modificar parámetros de aprendizaje, muestra los mensajes que se envían los objetos de las clases participantes en el sub proceso, estos eventos se muestran a continuación:

- Inicialmente el **Usuario** realiza la solicitud para modificar los parámetros de aprendizaje de una o ambas redes neuronales artificiales, la cual es captada por la clase **Interfaz** y enviada a la clase **PanelAprender** a través de su instancia.
- El objeto **panelAprender** verifica que la solicitud sea válida, si es así, entonces almacena los nuevos parámetros, y los muestra al **Usuario.**

El diagrama de secuencia para el caso de uso modificar parámetros de aprendizaje se puede observar en la figura 4.11.

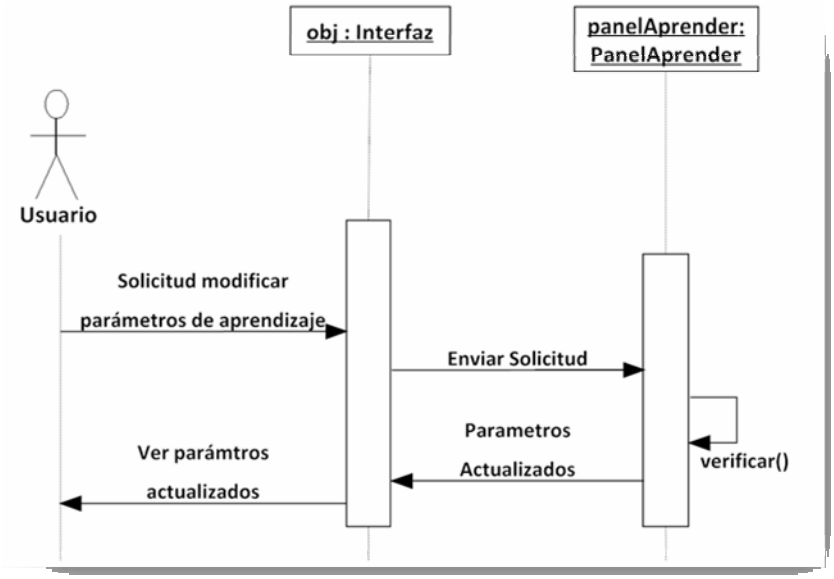

Figura 4.11. *Diagrama de secuencia para caso de uso modificar parámetros de aprendizaje. Fuente: Propia.*

4.4.6. Diagrama de secuencia para casos de uso aplicar PCA, obtener entrenamiento y guardar memoria.

En el diagrama de secuencia para los casos de uso aplicar PCA, obtener entrenamiento y guardar memoria; se muestran las comunicaciones realizadas entre los objetos para llevar a cabo las acciones solicitadas, y se detallan a continuación:

- El actor **Usuario** inicia la secuencia solicitando el entrenamiento de las redes neuronales a la clase **Interfaz**.
- La instancia de la clase **Interfaz** le envía la solicitud al objeto **panelAprender,** quien contiene los datos determinados por el usuario

anteriormente, seguido de esto el objeto **panelAprender** le indica al objeto **obj** que puede entrenar.

- El objeto **obj** crea una instancia de **Procesos** para entrenar las redes neuronales artificiales, para lo cual **procesos** llama al método entrenando() que se encuentra en **Interfaz**.
- El objeto de la clase **Interfaz** le solicita los rostros procesados a la instancia de **PCA**, mediante el método aprender().
- El objeto **pca** obtiene los patrones de la clase **Pixel**, quien solicita las imágenes al actor **Imágenes de Entrada**, para adaptarlas a las necesidades de **pca**.
- Una vez que **obj** obtiene los rostros reducidos solicita el entrenamiento de las redes neuronales artificiales, primero a **perceptronMulticapa** y luego a **rbf**.
- Culminado el entrenamiento de las redes neuronales, el objeto **obj** le indica al objeto **panelAprendiendo** que debe mostrar los resultados, para que el **Usuario** los pueda observar.
- Luego el objeto **obj** se comunica con **panelAprender** para que cree la memoria con los datos resultantes, enseguida **panelAprender** le solicita a **archMemorias** que escriba los datos, y éste lo hace actualizando el contenido del actor **Memorias**.

El diagrama de secuencia que contiene los casos de uso aplicar PCA, obtener entrenamiento y guardar memoria se muestran en la figura 4.12.

4.4.7. Diagrama de secuencia para caso de uso cargar imágenes de sujetos para reconocer.

El diagrama de secuencia para el caso de uso cargar imágenes de sujetos para reconocer se inicia en el proceso reconocer, se muestra en la figura 4.13 y se detalla a continuación:

- El actor **Usuario** realiza la solicitud para aprender al objeto **obj**, quien es una instancia de la clase **Interfaz**.
- El objeto **obj** se comunica con el objeto **panelReconocer,** de la clase **PanelReconocer**, para indicarle que ya es visible al usuario.
- El actor **Imágenes de Selección** le envía al objeto **panelReconocer** las imágenes de los distintos sujetos y las caras por sujeto, para que el objeto coloque estas imágenes a través de los métodos **colocarSujetos** y **colocarCaras**.
- La interfaz le muestra al usuario los sujetos y las caras disponibles para el aprendizaje.

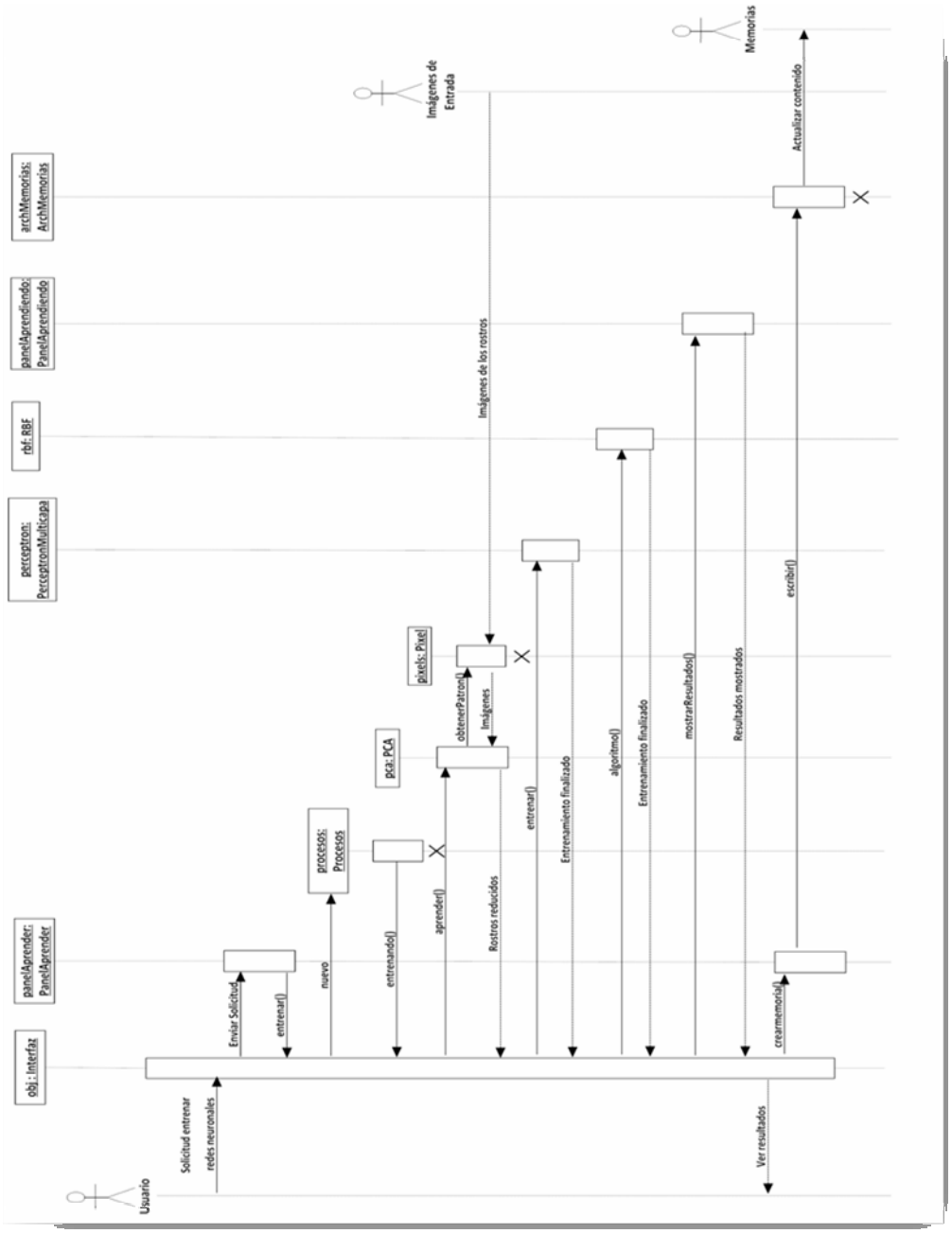

Figura 4.12. *Diagrama de secuencia para casos de uso aplicar PCA, obtener entrenamiento y guardar memoria. Fuente: Propia.*

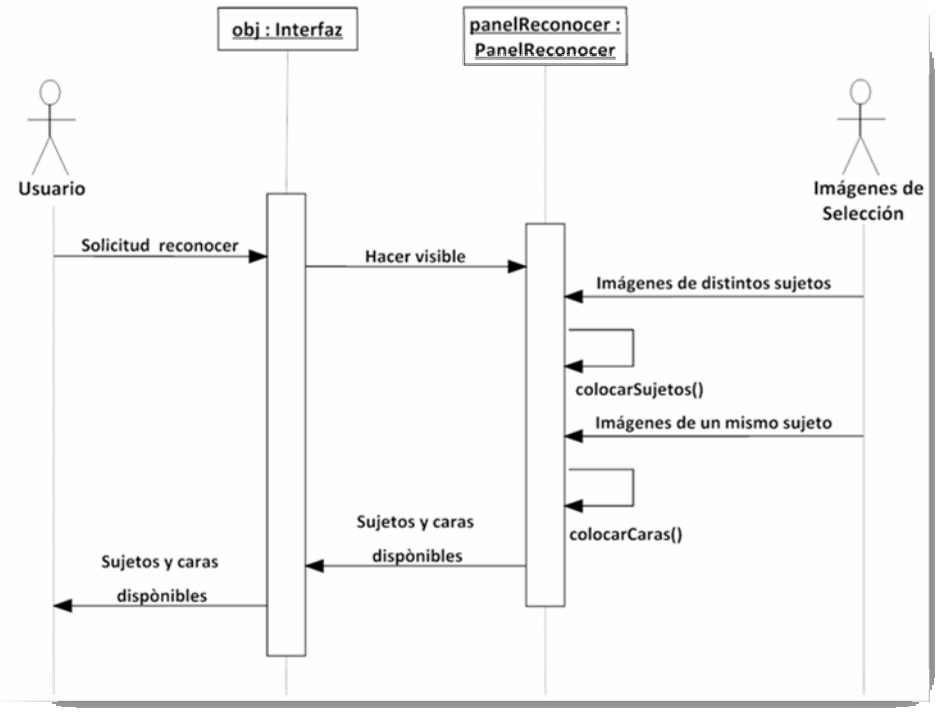

Figura 4.13. *Diagrama de secuencia para caso de uso cargar sujetos para reconocer. Fuente: Propia.*

4.4.8. Diagrama de secuencia para caso de uso seleccionar sujetos para reconocer.

En el diagrama de secuencia para el caso de uso seleccionar sujetos para reconocer, se muestra la cadena de eventos necesaria para realizar el sub proceso, así como los mensajes entre los objetos de las clases participantes. La secuencia se detalla a continuación:

- El actor **Usuario** realiza la solicitud a la clase **Interfaz,** a través de su instancia, para seleccionar los sujetos para reconocer.

- El objeto **obj** envía la solicitud a la instancia de la clase **panelReconocer.**
- El objeto **panelReconocer** actualiza la selección del usuario mediante su método interno actualizarEntradas().
- Una vez que la solicitud es aceptada, el **Usuario** podrá observar la selección realizada.

En la figura 4.14 se encuentra el diagrama de secuencia que muestra el caso de uso seleccionar sujetos para reconocer.

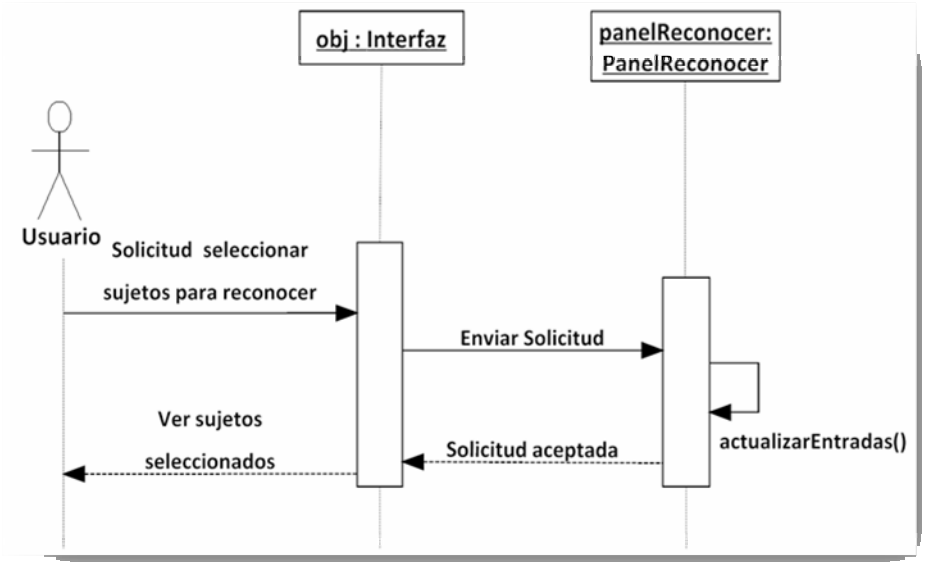

Figura 4.14. *Diagrama de secuencia para caso de uso seleccionar sujetos para reconocer. Fuente: Propia.*

4.4.9. Diagrama de secuencia para casos de uso cargar rostros reducidos de pca y obtener reconocimiento.

En el diagrama de secuencia que se encuentra en la figura 4.15, se muestra la comunicación realizada entre los objetos para llevar a cabo los casos de uso cargar

rostros reducidos de pca y obtener reconocimiento, los detalles se indican a continuación:

- El actor **Usuario** inicia la secuencia solicitando el reconocimiento de los sujetos seleccionados a la clase **Interfaz**.
- El objeto **obj** envía la solicitud a **panelAprender**, ya que éste último contiene todos los datos determinados por el **Usuario** anteriormente para el reconocimiento.
- El objeto **panelAprender** le indica a la instancia de **Interfaz** que puede iniciar el reconocimiento.
- El objeto **obj** le solicita los rostros reducidos, a la instancia de **PCA** mediante el método **reconocer()**.
- Una vez que **pca** responde con los rostros reducidos que se han seleccionado, se le hace el llamado a las redes neuronales para que reconozcan los sujetos seleccionados, primero a la instancia de la clase que representa al perceptrón multicapa, mediante el método **reconocerPatrones()**, y luego a la que representa a la red de función de base radial, a través de su método **reconocerPatrones()**.
- Finalizado el reconocimiento realizado por las redes neuronales artificiales, el objeto **obj** le solicita al objeto **panelReconociendo** que debe mostrar los resultados, para que el **Usuario** los pueda observar.

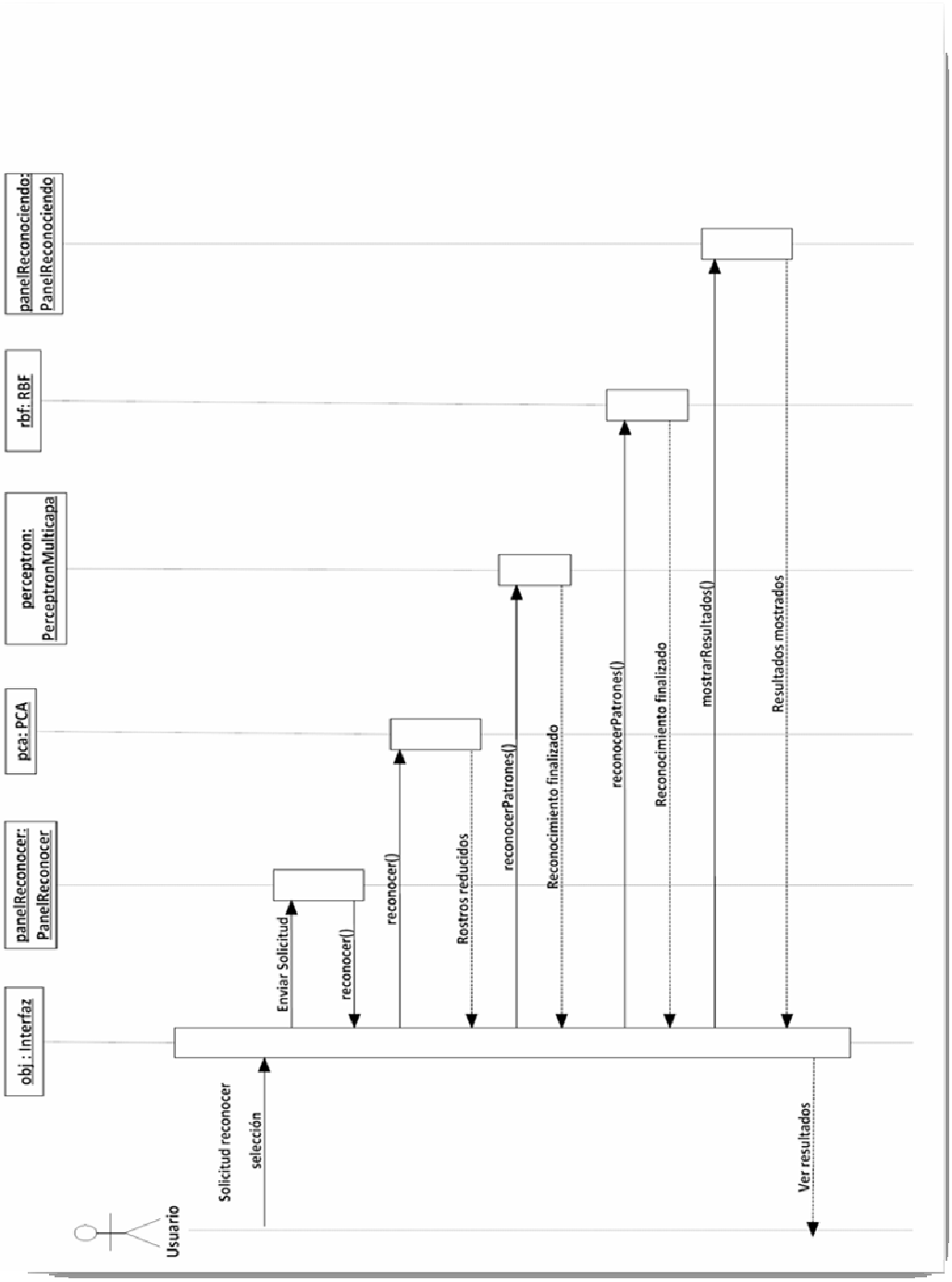

Figura 4.15. *Diagrama de secuencia para caso de uso cargar rostros reducidos de pca y obtener reconocimiento. Fuente: Propia.*

4.5. Técnica de desarrollo de sistema de objetos (TDSO)

La técnica de desarrollo de sistemas de objetos (TDSO) permite la especificación y documentación de clases de objetos, constituyendo una herramienta apropiada para el diseño de software orientado a objetos. Está basada en el método deductivo (MEDE), utilizado para analizar, especificar, documentar, diseñar y probar sistemas programados, y en la metodología de modelado de objetos OMT (Object-oriented modeling technique), que describe todo el proceso de modelado de clases de objetos. Del primero contiene todas las fases, incluyendo además las de especificación formal. De la segunda se toman algunos diagramas que fueron adaptados y transformados para TDSO.

Por otro lado ésta técnica permite definir el universo de clases y tipos de datos abstractos (TDAs), es decir, mediante ésta técnica se detalla cada clase en términos de sus atributos, la especificación sintáctica y semántica de sus métodos y las declaraciones de las instancias de la clase, que serán utilizadas en el escenario de pruebas de la especificación semántica.

El universo de clases de la aplicación para el reconocimiento facial se encuentra en la tabla 4.1, para ello se tomaron en cuenta, las clases descritas anteriormente en los diagramas de clases de diseño.

Las especificaciones de las clases que juegan un papel fundamental en cuanto al aprendizaje y reconocimiento de sujetos, en la aplicación, se describen en las tablas 4.2, 4.3, 4.4 y 4.5. Todo esto utilizando la técnica para la descripción de sistemas de objetos.

En cuanto a las clases participantes en la interfaz de usuario, por ser consideradas de menor relevancia en la resolución del problema, no serán descritas con la técnica de desarrollo de sistemas de objetos.

Tabla 4.1. *Universo de clases de la Aplicación para el Reconocimiento Facial.*

18/03/09		Versión 1.0
Universo de clases para la aplicación para el reconocimiento facial {Colección de clases que son requeridas para implantar la aplicación}		
1	**ArchMemorias**	Clase que permite gestionar la lectura y escritura del archivo que contiene las memorias.
2	**Interfaz**	Clase principal, que también administra la interfaz gráfica de usuario.
3	**Memorias**	Clase que almacena el contenido de cada memoria existente.
4	**PanelAprender**	Clase que contiene la interfaz gráfica de usuario necesaria para realizar el aprendizaje.
5	**PanelAprendiendo**	Clase que muestra el resultado arrojado por un aprendizaje.
6	**PanelConfigurar**	Clase que le permite al usuario configurar ciertos aspectos de la aplicación.
7	**PanelReconocer**	Clase que contiene la interfaz gráfica de usuario necesaria para realizar el reconocimiento.
8	**PanelReconociendo**	Clase que muestra el resultado del reconocimiento realizado.
9	**PCA**	Clase que permite extraer las características principales a las imágenes de los rostros.
10	**PerceptronMulticapa**	Clase que implementa la red neuronal perceptrón multicapa.
11	**Pixel**	Clase que lee y ajusta las imágenes de los sujetos, para la disposición de PCA.
12	**Procesos**	Clase que permite la ejecución de los procesos de la aplicación.
13	**RBF**	Clase que implementa la red neuronal función de base radial.

Fuente: Propia.

Tabla 4.2. *TDSO para la clase PCA.*

18/03/09		Versión 1.0
	9 clase: PCA {Clase que permite extraer las características principales a las imágenes de los rostros }	
	Especificación de atributos rostros: double[][] media: double[] normal: double[][] covarianza: double[][] autovalores: double[] autovectores: double[][] transformada: double[][] imgReducidas: double[][] ancho: int alto: int nroCaras: int nroSujetos: int nroComponentes: int	**rostros**: contiene los vectores de las imágenes de los rostros que se van a procesar. **media**: contiene el promedio de los vectores de las imágenes de los rostros a procesar. **normal**: matriz que representa el conjunto de datos al cual se le desea obtener las componentes principales. **covarianza**: matriz cuadrada que contiene la covarianza entre los elementos de los vectores. **autovalores**: contiene los autovalores necesarios para la obtención de los autovectores. **autovectores**: matriz cuadradada donde cada columna representa un autovector. **transformada**: matriz que contiene la transformación de las entradas. **imgReducidas**: matriz que representa las componentes principales de la matriz de imágenes inicial. **ancho**: ancho de las imágenes. **alto:** alto de las imágenes. **nroCaras**: almacena el número de caras por sujeto. **nroSujetos:** almacena el número de sujetos. **nroComponentes**: contiene el número de autovectores que se tomaran en cuenta.
	Especificación semántica	
1	PCA(int nroCaras, int nroSujetos)	**PCA**: constructor de la clase.
2	obtenerPatron(idCara: int, idSujeto: int)	**obtenerPatron**: obtiene el vector columna de una imagen especificada.
3	obtenerMedia()	**obtenerMedia**: obtiene la media de los vectores de las imágenes.
4	normalizar()	**normalizar**: obtiene la normal de la matriz de rostros.
5	obtenerMatrizCovarianza()	**obtenerMatrizCovarianza**: obtiene la covarianza entre los vectores de la matriz normalizada.
6	obtenerAutoVectores()	**obtenerAutoVectores**: obtiene los autovalores y autovectores aplicando el método de la potencia.
7	obtenerTransformada()	**obtenerTransformada**: obtiene la transformación de las entradas.
8	normalizarTransformada()	**normalizarTransformada**: normaliza la matriz de transformada, para que sus columnas formen vectores ortogonales entre sí.
9	obtenerRostrosReducidos()	**obtenerRostrosReducidos**: determina el número de autovectores a utilizar y solicita la reducción de los rostros.
10	obtenerImgReducidas(rostros:double[][],col: int)	**obtenerImgReducidas**: obtiene la matriz que contiene las componentes principales de las imágenes iniciales.

Fuente: Propia.

Tabla 4.3. *TDSO para la clase PerceptronMulticapa.*

18/03/09		Versión 1.0
	10 clase: PerceptronMulticapa {Clase que implementa la red neuronal perceptrón multicapa}	
	Especificación de atributos nroCapas: int nroPatrones: int nroEntradas: int nroSalidas: int neuronas: int[] capas: double[][] pesos: double[][] gradientes: double[][] umbrales: double[][] momentos: double[][] momentosU: double[][] salidasDeseadas: double[][] patronesEntradas: double[][] importancia: double aprendizaje: double	**nroCapas**: indica el número de capas de la red. **nroPatrones**: número de patrones de entrada que se desea entrenar o reconocer. **nroEntradas**: número de neuronas de la capa de entrada o longitud de cada patrón de entrada. **nroSalidas**: número de neuronas de la capa de salida. **neuronas**: contiene el número de neuronas de todas las capas. **capas**: contiene las salidas de cada neuronas ordenada por capas. **pesos:** contiene todos los pesos de las conexiones entre las neuronas. **gradientes:** almacena el gradiente calculado para una neurona en específico. **umbrales:** contiene todos los umbrales de todas las neuronas. **momentos:** contiene el factor momento para la corrección de los pesos. **momentosU:** contiene el factor momento para la corrección de los umbrales. **salidasDeseadas:** almacena las salidas que se desean para los patrones de entrada. **patronesEntradas:** contiene el conjunto de patrones que se desean reconocer o aprender. **importancia:** factor que determina la importancia del momento. **aprendizaje:** factor de aprendizaje empleado para el aprendizaje de la red. **PerceptronMulticapa:** constructor de la clase. **entrenar**: inicia el entrenamiento de la red. **corregirPesos**: corrige los pesos y umbrales durante el entrenamiento. **reconocerPatrones**: obtiene las salidas de un conjunto de patrones. **normalizarPatrones**: normaliza los patrones de entradas.
	Especificación semántica	
1	PerceptronMulticapa()	
2	entrenar()	
3	corregirPesos(i: int)	
4	reconocerPatrones(patrones:double [][],nroPatrones: int)	
5	normalizarPatrones(patrones double[][], nroPatrones:int)	

Fuente: Propia.

Tabla 4.4. *TDSO para la clase Pixel.*

18/03/09		Versión 1.0
	11 clase: Pixel {Clase que lee y ajusta las imágenes de los sujetos, para la disposición de PCA}	
	Especificación de atributos **Especificación semántica**	**handlesinglepixel**: obtiene un pixel de una imagen. **handlepixels**: obtiene el vector de una imagen especificada.
1	handlesinglepixel(x: int, y: int, pixel: int)	
2	handlepixels(img: Image, x: int, y: int, w: int,h: int)	

Fuente: Propia.

Tabla 4.5. *TDSO para la clase RBF.*

18/03/09		Versión 1.0
	13 clase: RBF {Clase que implementa la red neuronal función de base radial}	
	Especificación de atributos nEntrada: int nOculta: int nSalida: int filE: int colE: int entradas: double[][] carasPorS: int[] salidas: double[][] upOculta: double[][] pesos: double[][] momentos: double[][] amplitud: double[] fActivacion: double[][] umbral: double[] momentosU: double[] upSalida: double[][] fError: double alfa: double pesosIni: double[] importancia: double	**nEntrada**: número de neuronas de la capa de entrada. **nOculta**: número de neuronas de la capa oculta. **nSalida**: número de neuronas de la capa de salida. **filE**: número de patrones de entrada. **colE**: tamaño de los patrones de entrada. **entradas**: matriz que contiene los patrones de entrada. **carasPorS**: contiene el número de caras por cada sujeto. **salidas**: contiene las salidas esperadas de los patrones de entrada. **upOculta**: matriz que contiene los valores de las neuronas ocultas de la red. **pesos**: contiene los valores de los pesos de las conexiones entre la capa oculta y la de salida de la red. **momentos:** contiene el momentum para evitar posibles oscilaciones bruscas en los pesos. **amplitud**: contiene las amplitudes de los centros. **fActivacion**: contiene las activaciones de las neuronas ocultas de la red. **umbral:** contiene los umbrales de las neuronas de salida. **momentosU:** contiene el momentum para evitar posibles oscilaciones bruscas en los umbrales. **upSalida:** contiene las salidas de las red. **fError**: contiene el error en la salida de la red. **alfa:** contiene el factor de aprendizaje de la red. **pesosIni:** contiene el rango de valores iniciales para los pesos de la red. **importancia**: contiene la prioridad que se le dará al momento.
	Especificación semántica	
1	RBF()	**RBF**: constructor de la clase.
2	inicializarCentros()	**inicializarCentros**: inicializa los centros de la red.
3	kMedias()	**kMedias**: contiene el algoritmo para determinar los centros de la red.
4	determinarAmplitudes()	**determinarAmplitudes**: determina el espacio en el que se puede activar cada neurona oculta.
5	determinarActivacion()	**determinarActivacion**: determina las activaciones de las neuronas ocultas de la red.
6	determinarSalida()	**determinarSalida**: obtiene las salidas de la red .
7	determinarSalida(idPatron: int)	**determinarError**: obtiene el error para un patrón de entrada.
8	determinarError(idPatron: int)	**determinarErrorTotal**: obtiene el error para todos los patrones de entrada.
9	determinarErrorTotal()	**evaluarPatron**: corrige los pesos y umbrales de la red, para un patrón.
10	evaluarPatron(idPatron: int)	**minimosCuadrados**: corrige los pesos y los umbrales de la red, para todos los patrones.
11	minimosCuadrados()	**algoritmo**: obtiene el entrenamiento de la red.
12	algoritmo()	**evaluar**: obtiene el reconocimiento de la red.
13	evaluar(entradas: double [][],filE: int)	

Fuente: Propia.

Luego de describir las clases más importantes en el aprendizaje y reconocimiento de sujetos, haciendo uso de la técnica de desarrollo de sistemas de objetos, se procede a mostrar el contenido de los métodos que conforman dichas clases.

A continuación se muestran los métodos de la clase PCA, que como se ha mencionado anteriormente, esta clase permite extraer las características principales a las imágenes de los rostros. Para obtener las características principales de las imágenes, primero se le debe especificar a la clase mediante su constructor, el número total de sujetos y de caras disponibles por sujeto, luego de haber especificado los sujetos y las caras por cada uno, que participaran en el aprendizaje, se obtiene la media de todas las imágenes para poder normalizarlas, con la finalidad de obtener la matriz de covarianza, y así calcular la matriz transformada que luego de haberla normalizado, se podrán extraer las características principales de las imágenes de las caras que se deseen. En las tablas 4.6, 4.7, 4.8, 4.9, 4.10, 4.11, 4.12, 4.13, 4.14 y 4.15 están contenidos los métodos que conforman la clase PCA.

Tabla 4.6. *TDSO para el método 1 de la clase 9.*

18/03/09		Versión 1.0
9, 1 (Método, Público) **PCA(entero nroCaras, entero nroSujetos)** {Constructor de la clase} {**Pre:** nroCaras: número de caras disponibles por sujeto; nroSujetos: número de sujetos disponibles}		{**Post:** inicializa las variables de la clase}

1
2
3
	this.nroCaras = nroCaras;
	this.nroSujetos = nroSujetos;
	fil = ancho*alto;
		this.nroCaras: variable de la clase, almacena el número de caras por sujeto.
		this.nroSujetos: variable de la clase, almacena el número de sujetos.
		fil: variable que contiene la longitud de los vectores de las imágenes.

Fuente: Propia.

Tabla 4.7. *TDSO para el método 2 de la clase 9.*

18/03/09		Versión 1.0
9, 2 (Método, Público) **obtenerPatron(entero idCara, entero idSujeto): arreglo de doubles[]** {obtiene el vector columna de una imagen especificada}		
{**Pre:** idCara: número de la cara de un sujeto; idSujeto: número de un sujeto}		{**Post:** obtiene el vector de una imagen especificada}
1 2 3 4 5	Pixel pixels = new Pixel(); String url = "Rostros/s"+(idSujeto)+"/"+(idCara)+".jpg"; ImageIcon img= new ImageIcon(getClass().getResource(url)).getImage(); return pixels.handlepixels(img,0,0,ancho,alto);	**pixels**: variable de la clase, almacena el número de caras por sujeto. **url**: variable de la clase, almacena el número de sujetos. **img**: variable que contiene la longitud de los vectores de las imágenes.

Fuente: Propia.

Tabla 4.8. *TDSO para el método 3 de la clase 9.*

18/03/09		Versión 1.0
9, 3(Método, Privado) **obtenerMedia()** {obtiene la media de los vectores de las imágenes}		
{**Pre:** estar inicializadas las variables de la clase}		{**Post:** obtiene la media de las imágenes contenidas en rostros}

1
2
3
4
5
6
7
	media = new double[fil];
	para i=0 hasta i<fil hacer
	para j=0 hasta j<col hacer
	media[i] += rostros[i][j];
	fin para
	media[i] = media[i]/col;
	fin para
		media: contiene el promedio de los vectores de las imágenes de los rostros a procesar.
		i: variable para recorrer el primer bucle.
		j: variable para recorrer el segundo bucle.

Fuente: Propia.

Tabla 4.9. *TDSO para el método 4 de la clase 9.*

18/03/09		Versión 1.0
9, 4(Método, Privado) **normalizar()** {obtiene la normal de la matriz de rostros}		
{**Pre:** rostros y media ya deben estar inicializados}		{**Post:** obtiene los rostros normalizados}

1
2
3
4
5
6
	normal = new double[fil][col];
	
	para i=0 hasta i<fil hacer
	para j=0 para j<col hacer
	normal[i][j] = rostros[i][j] - media[i];
	fin para
	fin para
		normal: matriz que representa el conjunto de datos al cual se le desea obtener las componentes principales.
		i: variable para recorrer el primer bucle.
		j: variable para recorrer el segundo bucle.

Fuente: Propia.

Tabla 4.10. *TDSO para el método 5 de la clase 9.*

18/03/09		Versión 1.0
	9, 5(Método, Privado) **obtenerMatrizCovarianza()** {obtiene la covarianza entre los vectores de la matriz normalizada} {**Pre:** estar inicializadas las variables de la clase y la matriz normal} {**Post:** obtiene la matriz de covarianza}	

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
	matriz transpuesta de double= obtenerTranspuesta(normal);
	covarianza = multiplicarMatrices(transpuesta, normal, col, fil, col);
	double norma = 0;
	para i=0 hasta i<col hacer
	para j=0 hasta j<col hacer
	norma += covarianza[i][j]*covarianza[i][j];
	fin para
	norma = raiz(norma);
	fin para
	para i=0 hasta i<col hacer
	para j=0 hasta j<col hacer
	covarianza[i][j] /= norma;
	fin para
	fin para
		transpuesta: contiene la transpuesta de la normal.
		obtenerTranspuesta: obtiene la transpuesta de una matriz.
		norma: contiene la normal de un vector específico de la matriz resultante de la multiplicación entre la transpuesta y la normal.
		i: variable para recorrer las filas de la matriz de covarianza.
		j: variable para recorrer las columnas de la matriz de covarianza.

Fuente: Propia.

Tabla 4.11. TDSO para el método 6 de la clase 9. (1/2).

18/03/09		Versión 1.0
	9, 6(Método, Privado) **obtenerAutoVectores()** {obtiene los autovalores y autovectores aplicando el método de la potencia} {**Pre:** estar inicializadas las variables de la clase y la matriz de covarianza} {**Post:** obtiene los autovectores}	

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
	autovalores = new double[col];
	autovectores = new double[col][col];
	int nroIteraciones = 100;
	arreglo w de doubles = new double[col];
	double norma = 0;
	para nro=0 hasta nro<col hacer
	para i=0 hasta i<col hacer
	autovectores[i][nro] = 1/raiz(col);
	fin para
	para i=0 hasta i<nroIteraciones hacer
	norma = 0;
	para j=0 hasta j<col hacer
	w[j] = 0;
	para k=0 hasta k<col hacer
	w[j] += covarianza[j][k]*autovectores[k][nro];
	fin para
		autovalores: contiene los autovalores necesarios para la obtención de los autovectores.
		autovectores: matriz cuadradada donde cada columna representa un autovector.
		nroIteraciones: número de iteraciones en las que se deben obtener los autovectores.
		w: contiene el valor del autovector que se está calculando.
		norma: contiene la norma de w.
		nro: variable para recorrer las columnas de la matriz de autovectores.
		i: variable para recorrer el bucle.

Fuente: Propia.

Tabla 4.11. TDSO para el método 6 de la clase 9. (2/2).

17	norma+=potencia(w[j],2);	**j**: variable para recorrer el bucle.
18	fin para	**mult**: contiene el resultado de la multiplicación de un vector propio por su transpuesta.
19	norma = raiz(norma);	**mult2**: contiene el resultado de la transpuesta de un autovector por el mismo.
20	para j=0 hasta j<col hacer	
21	autovectores[j][nro] = w[j]/norma;	
22	fin para	
23	para i=0 hasta i<col hacer	
24	w[i] = 0;	
25	para j=0 hasta j<col hacer	
26	w[i] += autovectores[j][nro]*covarianza[j][i];	
27	fin para	
28	autovalores[nro] += w[i]*autovectores[i][nro];	
29	fin para	
30	matriz mult de double= new double[col][col];	
31	double mult2 = 0;	
32	para i=0 hasta i<col hacer	
33	para j=0 hasta j<col hacer	
34	mult[i][j] =	
35	autovectores[i][nro]*autovectores[j][nro];	
36	fin para	
37	fin para	
38	para i=0 hasta i<col hacer	
39	mult2 += autovectores[i][nro]*autovectores[i][nro];	
40	fin para	
41	para i=0 hasta i<col hacer	
42	para j=0 hasta j<col hacer	
43	mult[i][j] = (mult[i][j]/mult2)*autovalores[nro];	
44	covarianza[i][j] = covarianza[i][j] - mult[i][j];	
45	fin para	
45	fin para	
	fin para	

Fuente: Propia.

Tabla 4.12. *TDSO para el método 7 de la clase 9.*

18/03/09		Versión 1.0
9, 7(Método, Privado) **obtenerTransformada()** {obtiene la transformación de las entradas} {**Pre:** estar inicializadas las variables de la clase y las matrices normal y autovectores}		{**Post:** obtiene la matriz de transformada}
1	transformada = multiplicarMatrices(normal,autovectores,fil,col,col);	**transformada**: matriz que contiene la transformación de las entradas. **multiplicarMatrices**: multiplica dos matrices y retorna el resultado.

Fuente: Propia

Tabla 4.13. *TDSO para el método 8 de la clase 9.*

18/03/09		Versión 1.0
	9, 8(Método, Privado) **normalizarTransformada()** {normaliza la matriz de transformada, para que sus columnas formen vectores ortogonales entre sí}	
	{**Pre:** estar inicializadas las variables de la clase y la matriz transformada}	{**Post:** obtiene la matriz de transformada normalizada}

1
2
3
4
5
6
7
8
9
10
	double norma = 0;
	para i=0 hasta i<col hacer
	para j=0 hasta j<fil hacer
	norma += potencia(transformada[j][i],2);
	fin para
	norma = raiz(norma);
	para j=0 hasta j<fil hacer
	transformada[j][i]/=norma;
	fin para
	fin para
		norma: contiene la normal de un vector de la matriz de transformada.

Fuente: Propia.

Tabla 4.14. *TDSO para el método 9 de la clase 9.*

18/03/09		Versión 1.0
	9, 9(Método, Privado) **obtenerRostrosReducidos()** {determina el número de autovectores a utilizar y solicita la reducción de los rostros}	
	{**Pre:** estar inicializadas las variables de la clase y la matriz rostros}	{**Post:** determina el numero de autovectores a utilizar y solicita los rostros reducidos}

1
2
3
4
5
6
7
	si col/3 < 15 entonces
	nroComponentes = col-1;
	fin si
	sino entonces
	nroComponentes = col/3;
	fin sino
	imgReducidas = obtenerImgReducidas(rostros,col);
		nroComponentes: contiene el número de autovectores que se tomaran en cuenta.
		imgReducidas: matriz que representa las componentes principales de la matriz de imágenes inicial.

Fuente: Propia.

Tabla 4.15. *TDSO para el método 10 de la clase 9.*

18/03/09		Versión 1.0
9, 10(Método, Privado) **obtenerImgReducidas(rostros:double[][],col: int): matriz de double** {obtiene la matriz que contiene las componentes principales de las imágenes iniciales} {**Pre:** estar inicializadas las variables de la clase y las matrices rostros y transformada}		{**Post:** obtiene la matriz que contiene los rostros procesados}

1
2
3
4
5
 	
6
	matriz reduccion de doubles= new double[col][nroComponentes];
	
	matriz transRostros de doubles = obtenerTranspuesta(rostros,fil,col);
	matriz transTrans de doubles = obtenerTranspuesta(transformada);
	para i = 0 hasta i<col hacer
	reduccion[i] =
	multiplicarMatrizVector(transTrans,transRostros[i],nroComponentes,fil);
	return reduccion;
		reduccion: contiene los rostros procesados.
		transRostros: contiene la transpuesta de la matriz rostros.
		transTrans: contiene la transpuesta de la matriz transformada.
		i: variable para recorrer el bucle.
		multiplicarMatrizVector: multiplica una matriz por un vector y retorna el resultado.

Fuente: Propia.

Ya se mostraron los TDSO para la clase que extrae las características principales de los rostros, ahora se mostraran los TDSO de una de las redes neuronales que participan en el proceso de aprendizaje y reconocimiento, se trata del perceptrón multicapa, sus métodos se encuentran contenidos en las tablas 4.16, 4.17, 4.18, 4.19 y 4.20.

Tabla 4.16. *TDSO para el método 1 de la clase 10.*

18/03/09		Versión 1.0
10, 1(Método, Público) **PerceptronMulticapa()** {constructor de la clase} {**Pre:}**		{**Post:** crea una instancia de la clase}

Fuente: Propia.

Tabla 4.17. *TDSO para el método 2 de la clase 10.*

18/03/09		Versión 1.0
	10, 2(Método, Público) **entrenar()** {inicia el entrenamiento de la red, ajustando los pesos y umbrales para lograr el reconocimiento facial} {**Pre:** estar inicializadas las variables y las matrices empleadas en la clase} {**Post:** obtiene la matriz de pesos y umbrales necesarias para el reconocimiento}	

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
 	
20
21
22
23
24
25
26
27
	double cambio = 0;
	booleano error;
	cont = 0;
	hacer
	error = false;
	para i=0 hasta i<nroPatrones hacer
	capas[0] = patronesEntradas[i];
	para j=0 hasta j<nroCapas hacer
	para k=0, w=0 hasta k<neuronas[j+1] hacer
	capas[j+1][k] = umbrales[j][k];
	para l=0 hasta l<neuronas[j] hacer
	capas[j+1][k] += capas[j][l] * pesos[j][w];
	fin para
	capas[j+1][k] = 1/(1+exponente(capas[j+1][k]*(-1)));
	
	fin para
	fin para
	cambio = 0;
	para s=0 hasta s<nroSalidas hacer
	cambio+=potencia((salidasDeseadas[i][s]-capas[nroCapas][s]),2);
	fin para
	si cambio>0.1 entonces
	error = true;
	corregirPesos(i);
	fin si
	++cont;
	fin para
	mientras error;
		cambio: indica el error cometido por la red.
		error: indica si se alcanzó el error mínimo permitido para todos los patrones.
		cont: indica el número de iteraciones realizadas durante el proceso de aprendizaje.
		i: variable empleada para recorrer la matriz de patrones de entradas.
		j: variable empleada para recorrer el número de capas de la red.
		k: variable empleada para recorrer el número de neuronas de entrada para cada capa de la red.
		l: variable empleada para recorrer el número de neuronas de salida para cada capa de la red.
		capas: matriz empleada para almacenar el valor de las salidas de las neuronas a medida que se actualizan los pesos y umbrales.
		neuronas: matriz que contiene el número de neuronas por cada capa.
		umbrales: matriz que contiene el valor de los umbrales de las neuronas de la red.
		pesos: matriz que contienen todos los pesos entre las conexiones sinápticas.
		s: variable para recorrer el número de salidas deseadas.
		salidasDeseadas: matriz que contiene todas las salidas deseadas para cada patrón.

Fuente: Propia.

Tabla 4.18. *TDSO para el método 3 de la clase 10. (1/2)*

18/03/09 Versión 1.0

10, 3(Método, Público)

corregirPesos(i: int)

{corrige los pesos y umbrales a medida que se presenta cada patrón a la red}

{**Pre:** estar inicializadas las variables y las matrices empleadas en la clase}

{**Post:** corrige la matriz de pesos y umbrales y calcula los nuevos valores de momentos}

```
1  double sumatoria = 0;
2  double peso = 0;
3  double umbral = 0;
4  para j=nroCapas hasta  j>0 hacer
5       para k=0, w=0 hasta k<neuronas[j] hacer
6            si j==nroCapas entonces
7                 gradientes[j-1][k] = (salidasDeseadas[i][k]-
   capas[j][k])*capas[j][k]*(1-capas[j][k]);
8
9            fin si
10           sino
11                sumatoria = 0;
12                para m=0  hasta m<neuronas[j+1] hacer
13                     sumatoria +=
14 gradientes[j][m]*pesos[j][k+(neuronas[j]*m)];
15                fin para
16                gradientes[j-1][k] = capas[j][k]*(1-
17 capas[j][k])*sumatoria;
           fin sino
18         umbral = umbrales[j-1][k];
19         umbrales[j-1][k] = umbrales[j-1][k] + (aprendizaje *
20 gradientes[j-1][k]) + (importancia*momentosU[j-1][k]);
21         momentosU[j-1][k] = umbrales[j-1][k] - umbral;
           para l=0 hasta l<neuronas[j-1] hacer
22              peso = pesos[j-1][w];
23              pesos[j-1][w] = pesos[j-1][w] + (aprendizaje *
24 gradientes[j-1][k] * capas[j-1][l]) + (importancia*momentos[j-1][w]);
25              momentos[j-1][w] = pesos[j-1][w] - peso;
           fin para

     fin para
fin para
```

sumatoria: almacena la suma de todos los gradientes presentes en la capa siguiente.
peso: variable empleada para el cálculo de los momentos utilizados por los pesos.
umbral: variable empleada para el cálculo de los momentos utilizados por los umbrales.
j: variable empleada para recorrer todas las capas de la red para el cálculo de los gradientes.
gradientes: matriz con todos los gradientes calculados para cada neurona.
salidasDeseadas: matriz que contiene las salidas deseadas para cada patrón.
capas: matriz que contiene los valores para cada salida de la neurona.
neuronas: matriz que indica el número de neuronas por cada capa.
pesos: matriz que almacena todos los pesos de la red, a medida que son modificados.
umbrales: matriz que contiene todo los umbrales de las neuronas.
aprendizaje: factor de aprendizaje utilizado para la modificación de los pesos y umbrales.
importancia: factor de momento utilizado para la modificación de los pesos y umbrales.

Fuente: Propia.

Tabla 4.18. *TDSO para el método 3 de la clase 10. (2/2)*

		momentosU: matriz de momentos utilizada para la modificación de los umbrales. **momentos:** matriz de momentos utilizada para la modificación de los pesos.

Fuente: Propia.

Tabla 4.19. *TDSO para el método 4 de la clase 10. (1/2)*

18/03/09 Versión 1.0

10, 4(Método, Público)

reconocerPatrones(patrones:double[][], nroPatrones: int): matriz de cadenas

{hace uso de los pesos y umbrales para reconocer los patrones de entrada }

{**Pre:** realizar el entrenamiento de la red } {**Post:** obtiene una matriz resultado con todos los rostros reconocidos}

1	matriz resultado de cadenas = new cadena[nroPatrones][];	**resultado**: almacena para todos los patrones de entradas los rostros reconocidos. **i:** recorre todos los patrones a ser reconocidos. **patrones:** contiene todos los patrones que se desean reconocer. **capas:** utilizadas para almacenar cada patrón a reconocer y calcular las salidas de las neuronas. **j:** recorre todas las capas de la red. **k:** recorre todas las neuronas de entrada de la red. **l:** recorre todas las neuronas de salida de la red. **neuronas:** indica el número de neuronas por cada capa. **activadas:** variable empleada para conocer cuantas neuronas de la capa de salida están activadas. **suma:** variable empleada durante el cálculo de reconocimiento. **pesos:** matriz de pesos empleada para el reconocimiento.
2	entero activadas;	
3	double suma;	
4	patrones = normalizarPatrones(patrones, nroPatrones);	
5	para i=0 hasta i<nroPatrones hacer	
6	capas[0] = patrones[i];	
7	para j=0 hasta j<nroCapas hacer	
8	para k=0, w=0 hasta k<neuronas[j+1] hacer	
9	capas[j+1][k] = umbrales[j][k];	
10	para l=0 hasta l<neuronas[j] hacer	
11	capas[j+1][k] += capas[j][l] * pesos[j][w];	
12	fin para	
13	capas[j+1][k] = 1/(1+Math.exp(capas[j+1][k]*(-1)));	
14	fin para	
15	fin para	
16	activadas = 0;	
17	suma = 0;	
18	para j=0 hasta j<neuronas[nroCapas] hacer	
19	si redondear(capas[nroCapas][j])==1 entonces	
20	activadas++;	
21	suma += capas[nroCapas][j];	
22	fin si	
23	fin para	
24	resultado[i] = new cadena[activadas*2];	
25	para j=0, k=0 hasta j<neuronas[nroCapas] hacer	
26	si redondear(capas[nroCapas][j])==1 entonces	
27	resultado[i][k] = ""+(j+1);	
28	resultado[i][k+1] =	
29	""+(capas[nroCapas][j]*100/suma);	
30	k+=2;	
31	fin si	

Fuente: Propia.

Tabla 4.19. *TDSO para el método 4 de la clase 10. (1/2)*

32	fin para	**umbrales:** matriz de umbrales empleada para el reconocimiento.
33	fin para	
34	for(int i=0 ; i<nroPatrones ; i++)	
35	resultado[i] = ordenar(resultado[i]);	
	retornar resultado;	

Fuente: Propia.

Tabla 4.20. *TDSO para el método 5 de la clase 10.*

18/03/09		Versión 1.0
	10, 5(Método, Público)	
	normalizarPatrones(patrones double[][], nroPatrones:int): matriz de doubles	
	{normaliza todos los patrones de entradas}	
{**Pre:** estar inicializadas las variables y las matrices empleadas en la clase }		{**Post:** obtiene la matriz de patrones normalizada }
1	double mayor = -1;	**mayor**: variable para indicar el valor que se utilizará para normalizar todos los patrones de entrada.
2	para i=0 hasta i<nroPatrones hacer	**i:** variable utilizada para recorrer la matriz de patrones.
3	mayor = -1;	**j:** variable empleada para recorrer las componentes de cada patrón.
4	para j=0 hasta j<nroEntradas entonces	**patrones:** resultado obtenido después de la normalización de los patrones iniciales.
5	si valorAbsoluto(patrones[i][j])>mayor entonces	
6	mayor = valorAbsoluto(patrones[i][j]);	
7	fin para	
8	para j=0 hasta j<nroEntradas hacer	
9	patrones[i][j]/=mayor;	
10	fin para	
11	fin para	
12	retornar patrones;	

Fuente: Propia.

Los TDSO que siguen a continuación corresponden a la clase Pixel, que se encarga de leer una imagen especificada para transformarla en un vector. Los métodos necesarios para llevar a cabo éste propósito se encuentran en las tablas 4.21 y 4.22.

Tabla 4.21. *TDSO para el método 1 de la clase 11.*

18/03/09		Versión 1.0
11, 1(Método, Público) **handlesinglepixel(x: int, y: int, pixel: int): double** {obtiene un pixel de una imagen} {**Pre:** ubicación del pixel a obtener}		{**Post:** obtiene un pixel}
1 2	double blue = (pixel) & 0xff; retornar blue;	**blue** : pixel encontrado.

Fuente: Propia.

Tabla 4.22. *TDSO para el método 2 de la clase 11.*

18/03/09		Versión 1.0
11, 2(Método, Público) **handlepixels(img: Image, x: int, y: int, w: int,h: int): arreglo de doubles** {obtiene el vector de una imagen especificada} {**Pre:** ubicación de la imagen, posición a partir de la cual se obtendrán los pixeles y las dimensiones de la imagen}		{**Post:** obtiene el vector formado por los pixeles de la imagen}

1
2
3
4
5
6
7
8
9
10
	entero pixels[] = new entero[w * h];
	PixelGrabber pg = new PixelGrabber(img, x, y, w, h, pixels, 0, w);
	pg.grabPixels();
	patron: double[] = new double[w*h];
	para j = 0 hasta j < h hacer
	  para i = 0 hasta i < w hacer
	  patron[j * w + i] = handlesinglepixel(x+i, y+j, pixels[j * w + i]);
	  fin para
	fin para
	retornar patron;
		pixels: arreglo en donde se encontraran los pixeles de una imagen.
		pg: variable que permitirá convertir una imagen en un arreglo de valores que corresponden a sus pixeles.
		patron: arreglo que contiene el vector de pixeles de una imagen.
		j: variable para recorrer el bucle.
		i: variable para recorrer el bucle.

Fuente: Propia.

En las tablas 4.23, 4.24, 4.25, 4.26, 4.27, 4.28, 4.29, 4.30, 4.31, 4.32, 4.33, 4.34 y 4.35, se muestran los TDSO de la red neuronal artificial función de base radial, estos TDSO contienen los métodos que hacen posible que la red pueda realizar el aprendizaje o el reconocimiento de sujetos. El aprendizaje se lleva a cabo mediante el método algoritmo, que se encarga de dirigir el entrenamiento de la red, primero obteniendo los centros con KMedias (aprendizaje no supervisado), luego determinando la amplitud y la activación de cada centro, para luego corregir los

pesos con minimosCuadrados (aprendizaje supervisado), terminando así el proceso de aprendizaje de la red. En cuanto al reconocimiento, el método evaluar se encarga de realizar los pasos necesarios para que ocurra, primero obtiene la activación de cada centro con respecto a los patrones a reconocer y luego obtiene las salidas de la red para cada patrón, por medio de las cuales se conocerá sí el patrón (cara de un sujeto) fue reconocido o no por la red. A continuación se muestran los métodos que conforman la red neuronal función de base radial.

Tabla 4.23. *TDSO para el método 1 de la clase 13.*

18/03/09		Versión 1.0
	13, 1(Método, Público) **RBF()** {constructor de la clase}	
{**Pre**:}		{**Post:** crea una instancia de la clase}

Fuente: Propia.

Tabla 4.24. *TDSO para el método 2 de la clase 13. (1/2)*

18/03/09		Versión 1.0
	13, 2(Método, Público) **inicializarCentros()** {inicializa los centros de la red}	
{**Pre:** estar inicializadas las variables de la clase}		{**Post:** inicializa los centros de la red}

1
2
3
4
5
6
7
8
9
10
11
12
13
14
	double menor = entradas[0][0];
	double mayor = entradas[0][0];
	boleano marcador[] = new boleano[filE];
	entero seleccion = 0;
	para i = 0,j=0 hasta i<nOculta hacer
	hacer
	si j>=filE entonces
	romper;
	sino si j<nSalida entonces
	seleccion = j+(entero)(random()*carasPorS[j]);
	sino entonces
	seleccion = (entero)(random()*filE);
	fin si
	}mientras marcador[seleccion]
		menor: contiene el menor número de un patrón.
		mayor: contiene el menor número de un patrón.
		marcador: se utiliza para conocer las posiciones que se han seleccionado.
		seleccion: contiene una posición seleccionada aleatoriamente del arreglo marcador.
		i: variable para recorrer el bucle.
		j: variable para recorrer el bucle.

Fuente: Propia.

Tabla 4.24. *TDSO para el método 2 de la clase 13. (2/2)*

15	para k = 0 hasta k<nEntrada hacer	
16	si j<filE entonces	
17	si entradas[j][k]>mayor entonces	
18	mayor = entradas[j][k];	
19	sino si entradas[j][k]<menor entonces	
20	menor = entradas[j][k];	
21	fin si	
22	upOculta[i][k] = entradas[seleccion][k];	
23	marcador[seleccion] = verdadero;	
24	sino entonces	
25	si mayor>1 entonces	
26	upOculta[i][k] =	
27	menor+random()*(mayor-1);	
28	sino entonces	
29	upOculta[i][k] = menor+random();	
30	fin si	
31	fin si	
32		
	fin para	
	fin para	

Fuente: Propia.

Tabla 4.25. *TDSO para el método 3 de la clase 13. (1/2)*

18/03/09		Versión 1.0
	13, 3(Método, Público) **kMedias()** {contiene el algoritmo para determinar los centros de la red}	
	{**Pre:** estar inicializadas las variables de la clase}	{**Post:** obtiene los valores de las neuronas ocultas de la red}
1	double distancias[] = new double[nOculta];	**distancias:** contiene las distancias de un patrón a cada uno de los centros.
2	double menor;	**menor:** contiene la distancia del patrón más cercano a un centro.
3	entero indiceMenor;	**indiceMenor**: contiene la posición del centro más cercano a un patrón.
4	double centroAux[][] = new double[nOculta][nEntrada];	**centroAux**: matriz que contiene una copia de los centros de la red.
5	entero contador[] = new entero[nOculta];	**contador**: contiene el número de patrones más cercanos a cada centro.
6	entero cont = 0;	**cont**: contiene el numero de iteraciones que se han realizado.
7	boleano convergencia;	**i**: variable para recorrer el bucle.
8	inicializarCentros();	**j**: variable para recorrer el bucle.
9	hacer	**m**: variable para recorrer el bucle.
10	convergencia = verdadero;	
11	para i = 0 hasta i<nOculta hacer	
12	para j = 0 hasta j<nEntrada hacer	
13	centroAux[i][j] = 0;	
14	fin para	
15	fin para	
16	para m = 0 hasta m<filE hacer	
17	para i = 0 hasta i<nOculta hacer	
18	distancias[i] = 0;	

Fuente: Propia.

Tabla 4.25. *TDSO para el método 3 de la clase 13.(2/2)*

19	para d1 = 0 hasta d1<nEntrada hacer	**d1**: variable para recorrer el bucle.
20	distancias[i] += potencia(entradas[idMuestra][d1]-upOculta[idOculta][d1],2);	
21	fin para	
23	distancias[i] = raiz(distancias[i]);	
23	fin para	
24	menor = distancias[0];	
25	indiceMenor = 0;	
26	para i = 1 hasta i<nOculta hacer	
27	si distancias[i]<menor entonces	
28	menor = distancias[i];	
29	indiceMenor = i;	
30	fin si	
31	fin para	
32	para i = 0 hasta i<nEntrada hacer	
33	centroAux[indiceMenor][i] += entradas[m][i];	
34	fin para	
35	++contador[indiceMenor];	
36	fin para	
37	para i = 0 hasta i<nOculta hacer	
38	si contador[i]>0 entonces	
39	para j = 0 hasta j<nEntrada hacer	
40	centroAux[i][j] /= contador[i];	
41	fin para	
42	contador[i] = 0;	
43	fin si	
44	fin para	
45	si !verificarConvergencia1(centroAux)	
46	para i = 0 hasta i<nOculta hacer	
47	para j = 0 hasta j<nEntrada hacer	
48	upOculta[i][j] = centroAux[i][j];	
49	centroAux[i][j] = 0;	
50	fin para	
51	fin para	
52	convergencia = falso;	
53	fin si	
54	++cont;	
55	mientras (!convergencia && cont<10000 || cont<2);	
56	para i = 0 hasta i<nOculta hacer	
57	para j= 0 hasta j<nEntrada hacer	
58	upOculta[i][j] = centroAux[i][j];	
59	fin para	
60	fin para	

Fuente: Propia.

Tabla 4.26. *TDSO para el método 4 de la clase 13.*

18/03/09		Versión 1.0
	13, 4(Método, Público) **determinarAmplitudes()** {determina el espacio en el que se puede activar cada neurona oculta} {**Pre:** estar inicializadas las variables de la clase y calculados los valores de las neuronas ocultas} {**Post:** obtiene las amplitudes de los centros de la red}	
1	double distancia = 0;	**distancia:** contiene la distancia de una neurona oculta a otra.
2	double distancia1 = 0;	**distancia1:** contiene la distancia más cercana de una neurona oculta a la que se está evaluando.
3	double distancia2 = 0;	**distancia2:** contiene la siguiente distancia más cercana de una neurona oculta a la que se está evaluando.
4	para i = 0 hasta i<nOculta hacer	**i**: variable para recorrer el bucle.
5	d = 0;	**j**: variable para recorrer el bucle.
6	distancia1 = Double.MAX_VALUE;	
7	distancia2 = Double.MAX_VALUE;	
8	para j = 0 hasta j<nOculta hacer	
9	si i!=j entonces	
10	distancia = obtenerDistancia(i,j);	
11	si distancia<distancia1	
12	distancia1 = distancia;	
13	fin si	
14	fin para	
15	para j = 0 hasta j<nOculta hacer	
16	si i!=j entonces	
17	distancia = obtenerDistancia(i,j);	
18	si distancia>distancia1 && distancia<distancia2	
19	entonces	
20	distancia2 = distancia;	
21	fin si	
22	fin si	
23	fin para	
24	si distancia2== Double.MAX_VALUE entonces	
25	amplitud[i] = distancia1;	
26	sino entonces	
27	amplitud[i] = raiz(distancia1*distancia2);	
28	fin si	
	fin para	

Fuente: Propia.

Tabla 4.27. *TDSO para el método 5 de la clase 13.*

18/03/09		Versión 1.0
	13, 5(Método, Público) **determinarActivacion()** {determina las activaciones de las neuronas ocultas de la red} {**Pre:** estar inicializadas las variables de la clase, calculados los valores de las neuronas ocultas y determinadas las amplitudes de cada una} {**Post:** obtiene las activaciones de las neuronas ocultas para los patrones de entrada}	

1
2
3
4
5
6
7
8
9
10
11
	double distancia = 0;
	para i = 0 hasta i<nOculta hacer
	para j = 0 hasta j<filE hacer
	distancia = 0;
	para k = 0 hasta k<nEntrada hacer
	distancia += potencia(entradas[j][k]-upOculta[i][k],2);
	fActivacion[i][j] = distancia/potencia(amplitud[i],2);
	fActivacion[i][j] = -1*fActivacion[i][j]/2;
	fActivacion[i][j] = exp(fActivacion[i][j]);
	fin para
	fin para
		distancia: contiene la distancia entre un patrón y una neurona oculta.
		i: variable para recorrer el bucle.
		j: variable para recorrer el bucle.
		k: variable para recorrer el bucle.

Fuente: Propia.

Tabla 4.28. *TDSO para el método 6 de la clase 13.*

18/03/09		Versión 1.0
	13, 6(Método, Público) **determinarSalida()** {obtiene las salidas de la red} {**Pre:** estar inicializadas las variables de la clase y determinadas las activaciones de las neuronas ocultas} {**Post:** obtiene las salidas de la red para todos los patrones de entrada}	

1
2
3
4
5
6
7
8
9
	para i = 0 hasta i<nSalida hacer
	para j = 0 hasta j<filE hacer
	upSalida[i][j] = 0;
	para k = 0 hasta k<nOculta hacer
	upSalida[i][j] += pesos[i][k]*fActivacion[k][j];
	fin para
	upSalida[i][j] += umbral[i];
	fin para
	fin para
		i: variable para recorrer el bucle.
		j: variable para recorrer el bucle.
		k: variable para recorrer el bucle.

Fuente: Propia.

Tabla 4.29. *TDSO para el método 7 de la clase 13.*

18/03/09		Versión 1.0
13, 7(Método, Público) **determinarSalida(idPatron: int)** {obtiene las salidas de la red} {**Pre:** estar inicializadas las variables de la clase y determinadas las activaciones de las neuronas ocultas}		{**Post:** obtiene las salidas de la red para un patrón de entrada}

1
2
3
4
5
6
7
	para i = 0 hasta i<nSalida hacer
	upSalida[i][idPatron] = 0;
	para k = 0 hasta k<nOculta hacer
	upSalida[i][idPatron] += pesos[i][k]*fActivacion[k][idPatron];
	fin para
	upSalida[i][idPatron] += umbral[i];
	fin para
		i: variable para recorrer el bucle.
		j: variable para recorrer el bucle.
		k: variable para recorrer el bucle.

Fuente: Propia.

Tabla 4.30. *TDSO para el método 8 de la clase 13.*

18/03/09		Versión 1.0
13, 8(Método, Público) **determinarError(idPatron: int)** {obtiene el error para un patrón de entrada} {**Pre:** estar inicializadas las variables de la clase y calculadas las salidas de la red}		{**Post:** obtiene las salidas de la red para un patrón de entrada}

1
2
3
4
	error[idPatron] = 0;
	para i = 0 hasta i<nSalida hacer
	error[idPatron] += potencia(salidas[idPatron][i] - upSalida[i][idPatron],2);
	error[idPatron] /=2;
		i: variable para recorrer el bucle.
		j: variable para recorrer el bucle.
		k: variable para recorrer el bucle.

Fuente: Propia.

Tabla 4.31. *TDSO para el método 9 de la clase 13.*

18/03/09		Versión 1.0
13, 9(Método, Público) **determinarErrorTotal()** {obtiene el error para todos los patrones de entrada} {**Pre:** estar inicializadas las variables de la clase y calculados los errores por cada patrón}		{**Post:** obtiene las salidas de la red para los patrones de entrada}

1
2
3
4
5
	fError = 0;
	para i = 0 hasta i<filE hacer
	fError+= error[i];
	fin para
	fError /=filE;
		i: variable para recorrer el bucle.

Fuente: Propia.

Tabla 4.32. *TDSO para el método 10 de la clase 13.*

18/03/09		Versión 1.0
	13, 10(Método, Público) **evaluarPatron(idPatron: int)** {corrige los pesos y umbrales de la red, para un patrón}	
{**Pre:** estar inicializadas las variables de la clase y calculadas las activaciones de las neuronas ocultas}		{**Post:** corrige los pesos de la capa oculta a la de salida y los umbrales de la red, para un patrón}

1
2
3
4
5
6
7
8
9
10
11
12
 	
13
14
	double peso = 0;
	double umbralM = 0;
	determinarSalida(idPatron);
	para i = 0 hasta i<nSalida hacer
	para j = 0 hasta j<nOculta hacer
	peso = pesos[i][j];
	pesos[i][j] = pesos[i][j] + (alfa*(salidas[idPatron][i] - upSalida[i][idPatron])*fActivacion[j][idPatron])+(importancia*momentos[i][j]);
	momentos[i][j] = pesos[i][j] - peso;
	fin para
	umbralM = umbral[i];
	umbral[i] = umbral[i] + alfa*(salidas[idPatron][i] - upSalida[i][idPatron]) + (importancia*momentosU[i]);
	momentosU[i] = umbral[i] - umbralM;
	fin para
		peso: contiene el valor del peso antes del cambio.
		umbralM: contiene el valor del umbral antes del cambio.
		i: variable para recorrer el bucle.
		j: variable para recorrer el bucle.

Fuente: Propia.

Tabla 4.33. *TDSO para el método 11 de la clase 13.*

18/03/09		Versión 1.0
	13, 11(Método, Público) **minimosCuadrados()** {corrige los pesos y los umbrales de la red, para todos los patrones}	
{**Pre:** estar inicializadas las variables de la clase y calculados los centros, las amplitudes y las activaciones}		{**Post:** corrige los pesos y umbrales de la red para todos los patrones de entrenamiento}

1
2
3
4
5
6
7
8
9
10
11
12
13
14
	double ErrorAux=0;
	inicializarPesosYUmbrales();
	hacer
	fErrorAux = fError;
	para i = 0 hasta i<filE hacer
	evaluarPatron(i);
	fin para
	determinarSalida();
	para i = 0 hasta i<filE hacer
	determinarError(i);
	fin para
	determinarErrorTotal();
	++contMC;
	mientras !(fErrorAux==fError || fError<0.001);
		ErrorAux: contiene una copia del error total antes de ser modificado.
		umbralM: contiene el valor del umbral antes del cambio.
		i: variable para recorrer el bucle.

Fuente: Propia.

Tabla 4.34. *TDSO para el método 12 de la clase 13.*

18/03/09		Versión 1.0
	13, 12(Método, Público) **algoritmo()** {obtiene el entrenamiento de la red}	
{**Pre:** estar inicializadas las variables de la clase}		{**Post:** obtiene el enteramiento de la red para los patrones de entrada}

1
2
3
4
5
	inicializar();
	kMedias();
	determinarAmplitudes();
	determinarActivacion();
	minimosCuadrados();	

Fuente: Propia.

Tabla 4.35. *TDSO para el método 13 de la clase 13.*

18/03/09		Versión 1.0
	13, 13(Método, Público) **evaluar(entradas: double [][],filE: int)** {obtiene el entrenamiento de la red}	
{**Pre:** estar entrenada la red}		{**Post:** obtiene el reconocimiento de la red para los patrones de entrada}

1
2
3
4
5
6
	this.filE = filE;
	this.entradas = entradas;
	fActivacion = new double[nOculta][filE];
	upSalida = new double[nSalida][filE];
	determinarActivacion(opcionAct);
	determinarSalida();	

Fuente: Propia.

4.6. Implementación de la Aplicación para el Reconocimiento Facial.

Existe un conjunto de consideraciones a tener en cuenta para la implementación de esta aplicación, como lo son: el preprocesado de las entradas, la inicialización y actualización de los pesos, las condiciones de parada del entrenamiento y la influencia que tiene sobre este proceso la arquitectura de la RNA, así como las funciones de activación utilizadas; todos estos son factores decisivos a la hora de mejorar el entrenamiento de las RNA y por consiguiente el de la aplicación.

Para la resolución del problema en cuestión, se ha determinado utilizar dos tipos de redes neuronales, la red de perceptrón multicapa y la red de base radial, debido a que estas redes neuronales engloban muchas de las características de otras redes, entre las características más importantes para este estudio está la forma de representar los datos, una vez analizados o aprendidos. En cuanto al perceptrón multicapa este construye aproximaciones globales y la red de neuronas de base radial construye aproximaciones locales, entre los datos de entrada y salida. De esta manera nuestra investigación abarcará las diversas formas de representar la información y a su vez permitirá comparar que forma resulta más adecuada para este tipo de problema, es decir, que tipo de representación (global o local) se comporta de mejor manera para el aprendizaje y el reconocimiento de rostros.

Antes de explicar las técnicas utilizadas para el desarrollo de esta aplicación se presenta a continuación, en forma general, una aproximación del proceso necesario para el reconocimiento facial.

4.6.1. Visión general

El proceso para el reconocimiento facial consta de dos fases, la fase de aprendizaje y la fase de reconocimiento. Como se puede observar en la figura 4.16, en la fase de aprendizaje, los patrones de entradas, es decir, los rostros de todos los individuos, son preprocesados por el PCA, para luego ser presentados a la red perceptrón multicapa y a la red de neuronas de base radial, para su aprendizaje y posterior reconocimiento, como se muestra en la figura 4.17.

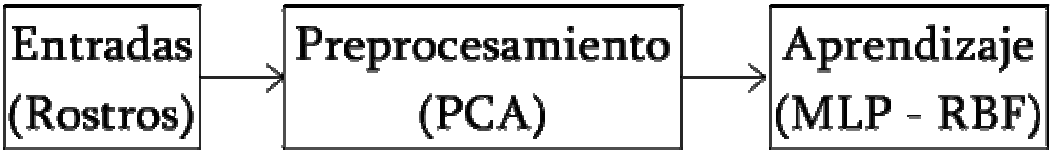

Figura 4.16. *Visión general del proceso de aprendizaje.*

Figura 4.17. *Visión general del proceso de reconocimiento.*

4.6.2. Fase de Aprendizaje

En esta fase la aplicación para el reconocimiento facial se aprende todos los rostros que forman parte del conjunto de entrenamiento, y que posteriormente deberá poder identificar en la fase de reconocimiento.

Como lo indica la figura 4.16, las entradas deben ser preprocesadas, antes de ser aprendidas por las redes neuronales. Estas entradas representan el conjunto de imágenes de los diferentes rostros de los sujetos que se utilizarán para el entrenamiento de la red neuronal.

Para el desarrollo de esta aplicación se dispuso de un conjunto de 40 sujetos, 10 caras por sujeto, con un total de 400 rostros con una resolución de 26x32 píxeles,

estos rostros fueron obtenidos de bases de datos públicas utilizadas normalmente para este tipo de estudio**.** A demás de estos rostros, se utilizó también un conjunto de rostros obtenido manualmente, haciendo uso de una cámara digital en un ambiente controlado.

Estas imágenes de los rostros son localizadas manualmente, es decir, la mayor parte de la imagen representa el rostro y la aplicación no debe ocuparse de esta labor. Los rostros utilizados para esta aplicación presentan un conjunto de expresiones gestuales, tomando en cuenta que en todas las imágenes de los rostros, los sujetos, se encuentran frente a la cámara.

Preprocesamiento de las entradas

Desde un punto de vista bastante global lo que se busca en esta etapa es aumentar la eficiencia del entrenamiento, ya que al preprocesar las entradas de las RNA, mejoran las propiedades numéricas en el proceso de optimización, evitando o disminuyendo notablemente los posibles problemas de saturación de las funciones de activación de las neuronas artificiales, además de eliminar el problema de la dependencia de los pesos iniciales a la escala de los datos de entrada. El procesado de patrones ayuda a que las diferencias relativas entre los datos sea el factor que domine durante el proceso de entrenamiento, en vez de ser las diferencias absolutas entre ellos.

La técnica utilizada en esta aplicación para el preprocesamiento es el Análisis de Componentes Principales (PCA, como se indica en la figura 4.16), con el fin de reducir la dimensión de las entradas. A continuación se explican los pasos citados en el apartado 2.2.1 para aplicar un PCA:

1. Normalización de la matriz de entrada X.

Supóngase que la matriz de entrada X esta formada por un conjunto de rostros, como se muestra en la figura 4.18, que representan los patrones de entradas utilizados para el entrenamiento de las redes neuronales, como se observa en la figura 4.17.

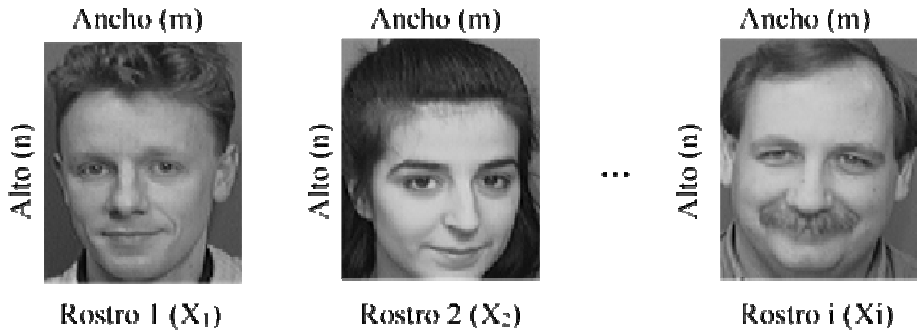

Figura 4.18. *Conjunto de rostros de entrada. Fuente: Propia.*

Del conjunto de rostros de entrada (figura 4.18), se obtiene la matriz de Rostros a partir de las caras de los sujetos que formarán parte del conjunto de entrenamiento (figura 4.19).

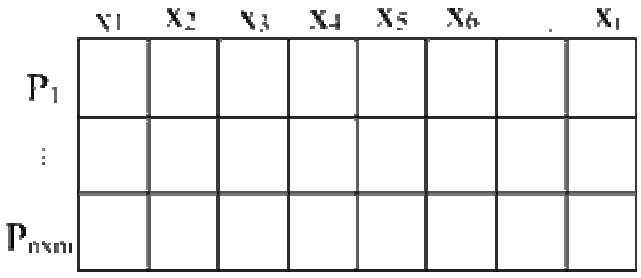

Figura 4.19. *Matriz de Rostros. Fuente: Propia.*

Una vez que se tenga la matriz de rostros se procede a su normalización, para esto se calcula el vector media de la matriz de rostros y éste se le resta a cada columna de la matriz, dando origen a una matriz *Normal,* que será la matriz normalizada (como se explica en el apartado *2.2.1.1 Matriz normalizada*).

Este vector media puede graficarse, viéndose de la siguiente forma (figura 4.20), dependiendo del conjunto de rostros.

Figura 4.20. *Representación gráfica del vector media de la matriz rostros. Fuente: Propia.*

2. Cálculo de la matriz de covarianza.

Como se muestra en la figura 4.21, para obtener esta matriz se multiplica la transpuesta de la matriz normalizada ($Normal^T$) por la matriz normalizada (Normal), obteniendo una matriz de menor dimensión (como se explica en el apartado *2.2.1.2 Matriz de covarianza*).

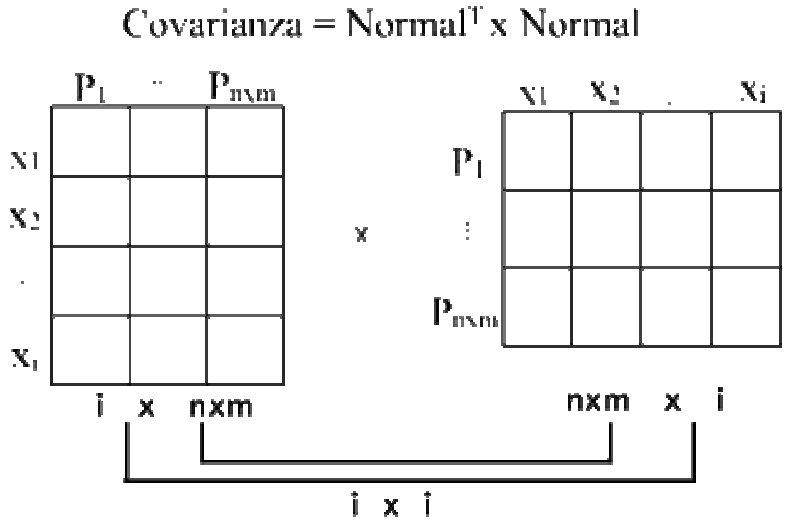

Figura 4.21. *Matriz de covarianza. Fuente: Propia.*

3. Cálculo de los autovalores y autovectores.

Como se puede observar en la figura 4.21, la matriz de covarianza tiene una dimensión menor que la matriz normalizada, en otras palabras, se podría decir que la dimensión de la matriz de covarianza será de ixi, donde i es el número de rostros a entrenar, de esta manera no importa la resolución que se utilice para el conjunto de rostros, si no, el número de rostros que intervengan en el entrenamiento. Esto se realiza de esta manera porque habitualmente el número de rostros en el entrenamiento (i) es mucho menor que la dimensión de la imagen del rostro (nxm). Por lo tanto, si se calcularan los nxm autovalores, solo $(i-1)$ serían distintos de cero. Para prevenir este cálculo innecesario, se calcula sólo los autovalores de una matriz de tamaño ixi (matriz de covarianza).

Para realizar el cálculo de los autovalores y autovectores asociados se hace uso del método de la potencia y de un método de deflación, como se explican en el apartado 2.2.1.3.1 y 2.2.1.3.2, resumiendo estos apartados se podría decir, que el método de la potencia obtiene de una matriz el autovalor de mayor módulo y el autovector asociado a este, y el método de deflación obtiene a partir de una matriz otra matriz con los autovalores restantes de la matriz original, es decir, este método hace cero el autovalor de mayor módulo que se encuentra en la matriz, y permite obtener los autovalores y autovectores de forma decreciente mediante la aplicación del método de la potencia, de la siguiente manera:

A. Se aplica el método de la potencia a la matriz de covarianza y se obtiene el autovalor más dominante (el mayor) y su autovector asociado.
B. Se aplica el método de deflación a la matriz de covarianza, haciendo cero el autovalor encontrado anteriormente. Se vuelve al paso A hasta encontrar todos los autovalores.

Una vez que se apliquen estos métodos se obtienen i autovalores y sus correspondientes autovectores. Estos autovalores no están relacionados entre sí, por lo tanto, los autovectores asociados son ortogonales entre sí, es decir, su producto escalar debe ser igual a cero.

4. Cálculo de la matriz de transformación W.

Para la obtención de los componentes principales se necesita de una matriz que extraiga características fundamentales de la matriz de entrada y a su vez disminuya su dimensión, para obtenerla se hace uso de los autovectores obtenidos y de la matriz normalizada, de la siguiente manera:

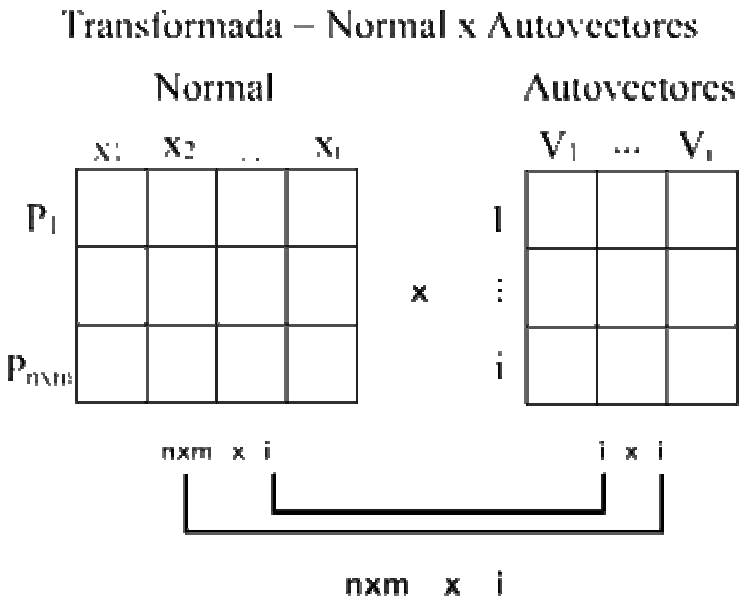

Figura 4.22. *Matriz de transformación. Fuente: Propia.*

Como se puede observar en la figura 4.22, se obtiene una matriz de dimensión [nxm x i], la cual puede variar dependiendo del número de autovectores que se deseen emplear para la representación de las imágenes, ya que estos expresan gran parte de

las características principales de las imágenes, por lo tanto, no es necesario hacer uso de todos, si no, de aquellos que tengan asociados autovalores de gran tamaño, en otras palabras, se hace uso solo de los autovectores que corresponden a los primeros autovalores de mayor módulo. Por lo tanto, la matriz transformada puede tener una dimensión [nxm x 2], [nxm x 3], [nxm x 4],…, [nxm x i].

Como los autovectores se clasifican en orden decreciente, los últimos autovalores (y sus autovectores) pueden eliminarse. Esta es la reducción real de dimensiones que ofrece PCA. En efecto, significa que se pueden eliminar algunos de los componentes principales porque éstos solo explican una pequeña porción de la información contenida en los datos iniciales, mientras que la mayor parte de la información se encuentra en los otros componentes principales.

Una vez decidido el número de autovectores a emplear y obtenida la matriz de transformación, es necesario normalizar esta matriz, debido a que la multiplicación entre diferentes columnas debe ser igual a cero, ya que estas columnas deben formar vectores ortogonales entre sí. Para esto se calcula la norma de cada columna y se divide entre cada valor de las columnas.

5. Cálculo de la matriz de entradas reducidas.

Para reducir la matriz de entradas se hace uso de la matriz de transformación, para esto se multiplica la transpuesta de la matriz de transformación por la matriz de rostros y se obtiene una nueva matriz, donde las columnas representan los rostros reducidos, que serán las entradas a la red neuronal.

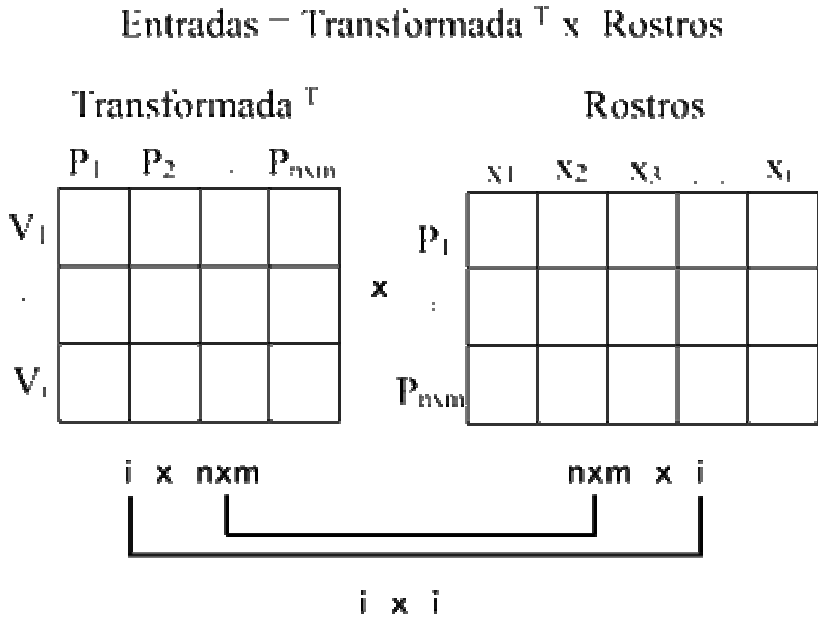

Figura 4.23. *Matriz de entradas reducidas. Fuente: Propia.*

De esta manera se obtiene una matriz de [i x i], donde las filas corresponden al número de componentes por rostro y las columnas al número de rostros. Tantos las filas como las columnas de esta matriz pueden variar, las filas dependen del número de autovectores seleccionados para la representación de los rostros y las columnas al número de imágenes seleccionadas para el entrenamiento, es decir, para un conjunto de rostros de 25 sujetos y 2 caras por sujeto se tendrán 50 rostros y 50 autovectores, por lo tanto, la dimensión de la matriz de entrada será de 50x50, si se usan todos los autovectores, pero esto puede reducirse disminuyendo el número de autovectores. Para determinar el mejor número de autovectores a utilizar se puede realizar de manera gráfica, por ejemplo para el rostro de la figura 4.24, se calculan y grafican los rostros generados a medida que se aplican todos los autovectores gradualmente, como se muestra en la figura 4.25.

Figura 4.24. *Rostro de prueba. Fuente: Propia.*

El resultado de aplicar gradualmente los autovectores se muestran a continuación, de izquierda a derecha, de arriba abajo, los autovectores están ordenados por sus autovalores en forma decreciente, por lo tanto la información proporcionada por los primeros es mucho mayor e importante en comparación con los siguientes.

Figura 4.25. *Resultado de aplicar los autovectores gradualmente a un rostro. Fuente: Propia.*

Como se puede observar en la figura 4.25, aproximadamente después de aplicar los primeros 30 autovectores al rostro, se comienza a distinguir la mayoría de las características de éste, por lo tanto, se podría reducir el número de autovectores, lo cual disminuirá la dimensión de la matriz de entrada a [30x50].

En resumen, se podría decir que después de aplicar el PCA a la matriz de entrada se obtiene otra matriz de menor dimensión, la cual es empleada para el entrenamiento de la red. En el *apartado 5.4 Pruebas y resultados computacionales. Prueba 9. Número de autovectores,* se realizan diversas pruebas cambiando el número de autovectores, demostrando que se puede reducir notablemente el número de autovectores, sin reducir la calidad de los resultados.

4.6.2.1. Aprendizaje de las entradas

Una vez que las entradas son preprocesadas, éstas son aprendidas por las redes neuronales perceptrón multicapa y de base radial, como se indica en la figura 4.16. Para este aprendizaje se buscó la formación de redes neuronales eficientes, capaces de aprender y generalizar las entradas, tomando en cuenta la arquitectura de las redes, la inicialización de los pesos, sesgos o umbrales, las funciones de activaciones más adecuadas y la técnica empleada para el aprendizaje. A continuación se describen las redes neuronales utilizadas.

4.6.2.1.1. Red de neuronas perceptrón multicapa

En la figura 2.7 se muestra la arquitectura del perceptrón multicapa empleado para la resolución de este problema, donde el número de capas C utilizado fue de tres (3) y donde cada neurona disponía de un sesgo o umbral como se explica en el apartado *2.3.3.2 Conexiones y pesos sináptico.* Su funcionamiento se detalla en el apartado *2.3.3.4 Elemento de procesado: neurona,* y se muestra su funcionamiento en la figura

2.2. En conclusión de estos apartados la red de neuronas perceptrón multicapa empleada se muestra en la figura 4.26.

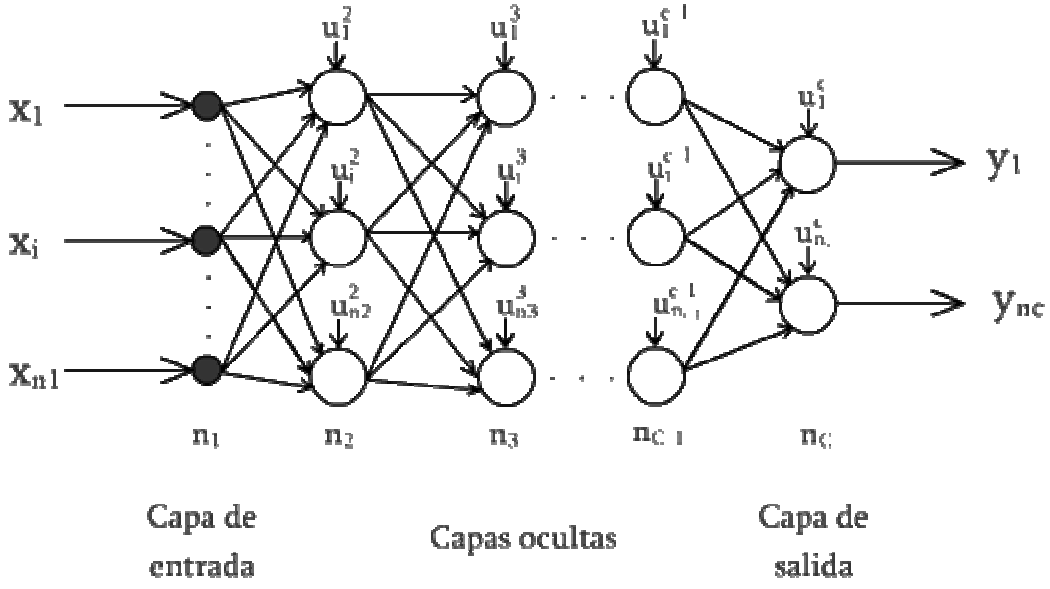

Figura 4.26. *Arquitectura del perceptrón multicapa con umbral. Fuente: Propia.*

Como se puede observar en la figura 4.26, el perceptrón multicapa puede tener tantas capas como se desee. Para el desarrollo de la aplicación se determinó que con una capa oculta resultaba suficiente, por lo tanto, la red perceptrón multicapa está formada de sólo tres capas, la de entrada, la oculta y la de salida, sin embargo, es posible cambiar el número de capas, así como también el número de neuronas para las capas ocultas. El número de neuronas para la capa de entrada viene dado por el número de variables que tengan los patrones de entrada, es decir, el número de componentes principales que se utilizó durante el preprocesamiento de los rostros. Para determinar el número de neuronas de la capa oculta se realizaron diversas pruebas, variando el número de neuronas, hasta dar con el número de neuronas más apropiado, el cual fue de 20 neuronas (*apartado 5.4 Pruebas y resultados computacionales. Prueba 1. Número de neuronas*) y para la capa de salida el número

de neuronas será definido por el número de individuos a memorizar, es decir, cada neurona representará a un individuo diferente.

Para la propagación de los patrones de entradas (*apartado 2.3.6.2 Propagación de los patrones de entrada*), es decir, de los rostros preprocesados por el PCA, en la capa de entrada se hizo uso de la ecuación 2.21, para transmitir hacia la red las señales recibidas del exterior, en las capas ocultas se hizo uso de la ecuación 2.22, para procesar la información recibida de la capa de entrada, aplicando una función de activación f a la suma de los productos de las activaciones que recibe por sus correspondientes pesos, y en la capa de salida se utilizó la ecuación 2.23, que funciona de igual forma que en el caso anterior. La función de activación empleada fue la función sigmoidal (ecuación 2.24), una de las principales razones por la cual se hizo uso de esta función es porque arrojó los mejores resultados (*véase apartado 5.4 Pruebas y resultados computacionales. Prueba 2. Función de activación*) y por ser compatible con el algoritmo de aprendizaje utilizado (*Backpropagation*), el cual exige que la función de activación sea derivable.

Para la corrección del error provisto por el algoritmo supervisado, durante la propagación de las entradas, se hizo uso del Algoritmo de *Backpropagation (apartado 2.3.6.3 Algoritmo Backpropagation, Retropropagación)*. Para la corrección de los pesos se hizo uso de la ecuación 2.30, donde el gradiente δ depende de la ubicación de los pesos, es decir, si los pesos a corregir pertenecen a una neurona en la capa de salida, entonces se utiliza la ecuación 2.31 y si se trata de una neurona en una capa oculta se emplea la ecuación 2.32. Este algoritmo también se empleó para la modificación de los umbrales, lo cual, mejoró notablemente en el rendimiento y en la velocidad de aprendizaje (véase *apartado 5.4 Pruebas y resultados computacionales. Prueba 6. Umbrales*).

Para aumentar la eficiencia del algoritmo Backpropagation se adicionó a este el término *momentum* o momento, descrito en el apartado *2.3.6.6 Aprendizaje con momentum*, lo que se busca es que la red no se actualice sólo en función del gradiente local, sino que tenga en cuenta las tendencias más recientes en las superficies del error, entonces, para la modificación de los pesos y umbrales se realizó como se muestra en la ecuación 2.46. Generalizando se podría decir, que el resultado de aplicar la ecuación 2.30 para la corrección de los pesos y la ecuación 2.46 para la inclusión del momento se puede apreciar en las siguientes ecuaciones:

$$\textit{Para la correción de los pesos: } \Delta w_{ji}^{k} = \rho \cdot \delta_j^k \cdot a_i^{k-1} + \alpha \Delta w_{iter} \qquad (4.1)$$

$$\textit{Para la correción de los umbrales: } \Delta u_{ji}^{k} = \rho \cdot \delta_j^k + \alpha \Delta u_{iter} \qquad (4.2)$$

Para el desarrollo de la aplicación para el reconocimiento facial, el factor de momentum α utilizado fue de 0.9, para la obtención de este factor se realizaron diversas pruebas, como se muestra en el *apartado 5.4 Pruebas y resultados computacionales. Prueba 5. Factor momento*, donde se tomó en cuenta el número de iteraciones necesarias para el entrenamiento, así como también la capacidad de generalización. Los valores de los factores utilizados se ubican en el intervalo [0, 1] y como resultado se seleccionó el factor con menor número de iteraciones y mejor generalización.

En resumen de todo lo explicado anteriormente, se muestra a continuación el diseño del algoritmo para el aprendizaje de la red perceptrón multicapa.

1. Se inicializan los pesos y los umbrales
2. Para todos los patrones de entradas
    2.1. Si el error cometido por la red > 0.1, entonces, (ecuación 2.27)
        2.1.1. Corregir los pesos y umbrales (ecuaciones 4.1 y 4.2)

2.2. Si no, se continúa con el siguiente patrón

3. Si hay correcciones, se repite el paso 2
4. Si no, se finaliza.

Como se puede observar todo el proceso de aprendizaje consiste en una continua supervisión del error cometido por la red, cuando esta no se equivoque (el error sea menor a 0.1), entonces, no ocurrirán correcciones y el algoritmo de aprendizaje finalizará. Cabe destacar que el valor 0.1 se obtuvo después de una serie de pruebas minuciosas, como se puede apreciar en el *apartado 5.4 Pruebas y resultados computacionales. Prueba 4. Error mínimo.*

Para la inicialización de los pesos se debe tener presente que los algoritmos de entrenamiento basados en el gradiente son técnicas de búsqueda local, que se quedarán atrapados en puntos de la superficie de error donde el gradiente sea nulo, condición que asegura que se ha encontrado un mínimo, pero que no asegura que se ha llegado al mínimo global de la función, por lo tanto, los pesos elegidos para la minimización del error serán determinantes tanto para la calidad del mínimo encontrado como para las iteraciones necesarias para encontrarlos.

No hay una solución determinista que asegure la buena selección de los pesos iniciales, por lo tanto, se realizaron varias pruebas de inicialización para conocer el rango promedio de valores en el que la red obtenía mejores resultados, para luego implementar una inicialización aleatoria de los pesos, distribuida uniformemente sobre el rango encontrado.

Cabe mencionar que para valores altos de los pesos se saturaría la función de activación de la red, anulando las derivadas del algoritmo de *Backpropagation* y, por lo tanto, estancando el proceso de entrenamiento y en caso de valores pequeños, las salidas generadas por la red serían próximas a cero y el entrenamiento sería muy lento (*apartado 5.4 Pruebas y resultados computacionales. Prueba 7. Pesos*).

4.6.2.1.2. Red de neuronas de base radial

En la figura 2.9 se muestra la arquitectura de la red de neuronas de base radial empleada para la resolución de la aplicación para el reconocimiento facial, estas redes, como se detalla en el apartado 2.3.7.1, solo constan de tres capas, una de entrada, una oculta y una de salida. A diferencia de la red perceptrón multicapa, esta red posee umbrales o sesgos solo en las neuronas de la capa de salida, como se describe en el apartado *2.3.3.2 Conexiones y pesos sináptico.* A continuación se muestra la red de base radial con los umbrales incluidos:

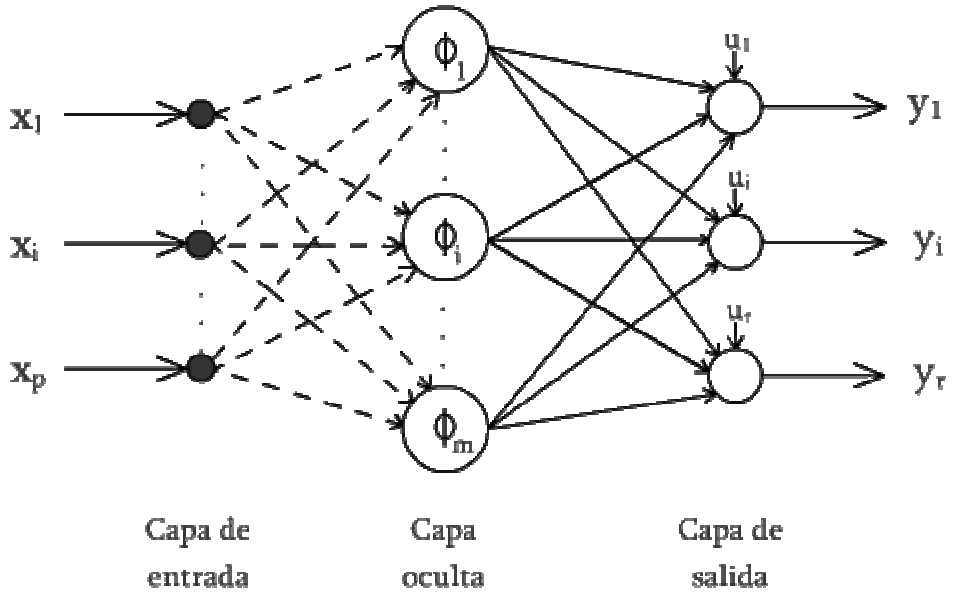

Figura 4.27. *Arquitectura de la red de neuronas de base radial con umbral.*

Fuente: Propia.

Como se puede observar en la figura 4.25, la red de neuronas de base radial solo dispone de tres capas, en la capa de entrada el número de neuronas viene dado por la cantidad de variables o componentes que tenga el patrón de entrada (rostro preprocesado), en la capa oculta, el número de neuronas puede ser arbitrario según el

criterio del investigador o del usuario de la aplicación, sin embargo, para determinar el mejor número de neuronas se realizaron diversas pruebas, variando el número de neuronas hasta conseguir una red capaz de resolver el problema y obtener buenos resultados, llegando a la conclusión que la mejor aproximación sería colocar el número de neuronas igual al número de rostros del conjunto de entrenamiento. Y, el número de neuronas en la capa de salida será igual al número de clases, es decir, al número de sujetos que la aplicación será capas de reconocer.

Cabe destacar que el número de neuronas de las capas de entrada y salida no pueden ser cambiadas por el usuario, ya que estas dependen de la magnitud del problema, en cambio el número de neuronas en la capa oculta puede ser cambiado si así se desea. En el *apartado 5.4 Pruebas y resultados computacionales. Prueba 1. Número de neuronas,* se muestran diversas pruebas usando números de neuronas diferentes.

El proceso de aprendizaje en la red de base radial, requiere de la determinación de un conjunto de parámetros, como son: los centros, las desviaciones o amplitudes de las neuronas ocultas, los pesos de la capa oculta a la capa de salida y los umbrales de las neuronas de salida (*apartado 2.3.7.4 Aprendizaje de las redes de base radial*). Para el cálculo de todos estos parámetros se empleó un método de aprendizaje híbrido (*véase 2.3.7.4.1 Método de aprendizaje híbrido*), por lo tanto, este consta de una fase no supervisada y otra supervisada.

En la fase no supervisada, para la determinación de los centros se utilizó el algoritmo de K-medias (explicado en el apartado 2.3.7.4.1) y para la determinación de las amplitudes se probó con dos heurísticas, descritas en las ecuaciones 2.58 y 2.59, de las cuales, se tomó la heurística basada en la media geométrica, por su mayor desempeño y mejor comportamiento (como se muestra en el *apartado 5.4 Pruebas y resultados computacionales. Prueba 8. Heurísticas*).

En la fase supervisada, se hizo uso de la ecuación 2.47 para determinar las activaciones de las neuronas de salida (*apartado 2.3.7.2 Activación de las neuronas de la red de base radial*), donde las activaciones de las neuronas ocultas $\emptyset_i$, se determina mediante la ecuación 2.48, la cual describe en su interior una función de base radial denotada $\emptyset$, que puede adoptar diferentes formas y expresiones, sin embargo, para este estudio se utilizó la función gausiana (ecuación 2.50) por presentar mejores resultados. Para mayor información véase el *apartado 5.4 Pruebas y resultados computacionales. Prueba 2. Función de activación.*

Durante el proceso de aprendizaje la ecuación 2.60 sirvió de guía, para determinar cuando se debía corregir un peso o un umbral, o cuando se debía detener el proceso de aprendizaje, como condición de parada se determinó que si el error se mantenía constante con respecto al anterior, entonces, el proceso se detendría. Para la corrección de los pesos y umbrales, se empleó la técnica de mínimos cuadrados (descrita en el apartado 2.3.7.4.1), según la ecuación 2.68 y 2.69 respectivamente, como se puede observar en las ecuaciones anteriores, la magnitud de la corrección de los pesos y umbrales va a estar determinada por un factor α denominado tasa de aprendizaje, el cual, fue inicializado en un valor de 0.01 (véase *apartado 5.4 Pruebas y resultados computacionales. Prueba 3. Factor de aprendizaje*). Este proceso de corrección de pesos y umbrales se realiza hasta que el error se estabilice.

El diseño del algoritmo para el aprendizaje de la red de base radial se muestra a continuación:

1. Se inicializan los pesos y los umbrales
2. Se calculan los centros C_i (ecuación 2.56)
3. Se calculan las amplitudes d_i (ecuación 2.58)
4. Se calculan las activaciones de las neuronas ocultas $\emptyset_i$ (ecuación 2.48)

5. Para todos los patrones de entrada
   5.1 Se corrigen los pesos y umbrales (ecuaciones 2.68 y 2.69)
6. Si el error se estabiliza o el error cometido por la red < 0.01, se finaliza,
7. Si no, se repite el paso 5.

Como se puede observar en el algoritmo anterior, en el paso 6 el error cometido por la red se compara con 0.01, cabe destacar que este valor puede variar, es decir, puede ser un valor más pequeño o más grande, dependiendo del margen de error que se le quiera permitir a la red durante el entrenamiento de los patrones de entrada, para este estudio se fijó el valor de 0.01 por arrojar buenos resultados a la hora de generalizar. Para mayor información véase el *apartado 5.4 Pruebas y resultados computacionales. Prueba 4. Error mínimo.*

Para aumentar la eficiencia del algoritmo de mínimos cuadrados, descrito en la ecuación 2.62 y 2.63, para la corrección de los pesos y umbrales, se adicionó a éste un término de momento para acelerar el proceso de aprendizaje, permitiendo de esta manera la modificación de los pesos y umbrales mediante el uso de las ecuaciones 4.3 y 4.4, descritas a continuación:

$$w_{ik}(n) = w_{ik}(n-1) - \alpha_1 \frac{\partial e(n)}{\partial w_{ik}} + \beta \Delta w_{iter} \qquad (4.3)$$

$$u_k(n) = u_k(n-1) - \alpha_1 \frac{\partial e(n)}{\partial u_k} + \beta \Delta w_{iter} \qquad (4.4)$$

El factor de aprendizaje α utilizado para la aplicación fue de 0.01 y el factor de momentum β fue de 0.7, para la obtención de estos factores se realizaron diversas

pruebas, como se muestran en el *apartado 5.4 Pruebas y resultados computacionales. Prueba 3. Factor de Aprendizaje y prueba 5. Factor momento*.

Para la inicialización de los pesos, al igual que en la red perceptrón multicapa, se realizaron varias pruebas de inicialización para conocer el rango promedio de valores en el que la red obtenía mejores resultados, para luego implementar una inicialización aleatoria de los pesos, distribuida uniformemente sobre el rango encontrado. Véase *apartado 5.4 Pruebas y resultados computacionales. Prueba 7. Pesos.*

4.6.3. Reconocimiento de las entradas

En esta fase se le presentan los rostros que se desean identificar o verificar a las redes neuronales artificiales, para su reconocimiento. Tanto la red perceptrón multicapa como la red de neuronas de base radial, deben estar entrenadas para garantizar el reconocimiento de los sujetos.

Para el reconocimiento, al igual que para el aprendizaje, se debe preprocesar los rostros que se desean reconocer, como lo indica la figura 4.17. Una vez que los rostros estén preprocesados, son presentados a las redes y éstas haciendo uso de los pesos y los umbrales logran la identificación o verificación de los rostros.

Como se ha mencionado, cada neurona de la capa de salida (de ambas redes) representa a un individuo en particular, es decir, si se le presenta a las redes un rostro denominado “sujeto 2“, estas deberán encender la neurona correspondiente al sujeto presentado, como se muestra en la figura 4.28.

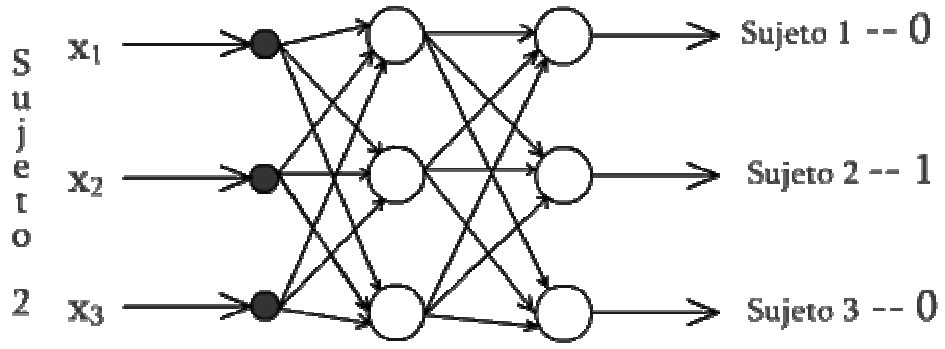

Figura 4.28. *Ejemplo de reconocimiento unánime. Fuente: Propia.*

Como se puede observar en la figura 4.28, lo más lógico es que encienda sólo una de las salidas, debido a que se le presenta a la red sólo un rostro o patrón a la vez, sin embargo, es posible que se enciendan más de una. En el caso que se encienda más de una neurona se deberá examinar su valor de activación exacto, es decir, su valor dentro del rango [0, 1], teniendo en cuenta que una neurona se activa o se enciende cuando su valor de activación es mayor o igual a 0.5, por lo tanto, si más de una neurona se activa se tomarán sus valores exactos de activación y se calculará el porcentaje que corresponde para cada salida, para determinar cual de todas las neuronas corresponde al patrón presentado.

En la figura 4.29 se muestra un ejemplo en el que se activan dos neuronas de salida.

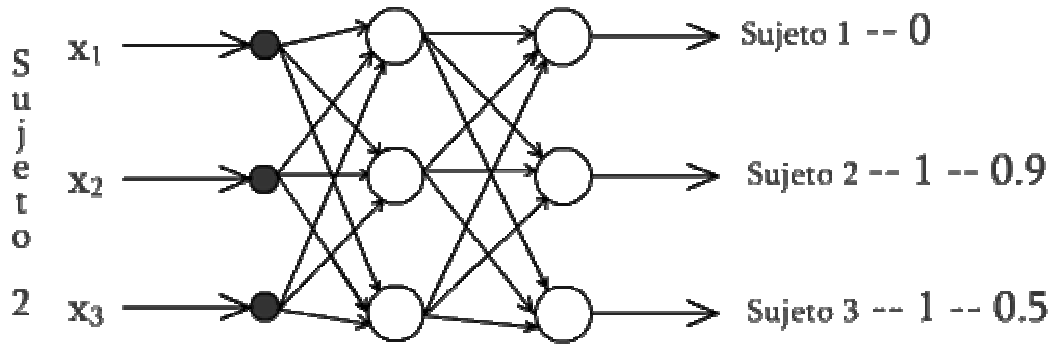

Figura 4.29. *Ejemplo de reconocimiento no unánime. Fuente: Propia.*

Como se puede observar en la figura 4.29, se le presenta a la red el patrón "Sujeto 2" y esta tiene dos salidas activadas, una salida sugiere que el patrón presentado corresponde al sujeto 2 y la otra, que corresponde al sujeto 3, en este caso, para determinar cuál de las dos salidas activadas tiene mayor credibilidad se hace uso de su valor exacto de activación, el cual es de 0.9 para una y de 0.5 para la otra. Para esto se dice que 0.9 + 0.5 = 1.4 corresponde al 100% y que 0.9 representa el 64.28% y 0.5 representa el 35.72%, por lo tanto, se llega a la conclusión que el patrón presentado corresponde al sujeto 2 por un 64.28% y al sujeto 3 por un 35.72%, estos porcentajes se han denominado porcentajes de acierto y al conjunto de porcentajes de acierto correctos se ha denominado porcentaje de generalización de la red.

Cuando el porcentaje de generalización alcanza el 100% significa que la red es capaz de reconocer a todos los rostros de pruebas, es decir, todas las salidas de las neuronas de la última capa se encuentran apagadas excepto la que corresponde al sujeto presentado (100% de acierto para cada sujeto), sin embargo, si éste porcentaje de generalización disminuye, significa que existe alguna duda en cuanto al reconocimiento de algún sujeto, como es el caso mostrado en la figura 4.29, donde varias salidas de la red se encienden y el porcentaje de acierto se distribuye.

A pesar que el porcentaje de generalización sea menor a 100%, esto no significa que la red no reconozca a todos los sujetos, esto sólo significa que la red arroja varios resultados para un sólo sujeto, por lo tanto, se podría decir que aunque la red no generalice totalmente el problema, la probabilidad de acertar todas las salidas, tal vez alcance porcentajes mayores, como se puede apreciar en el *apartado 5.4 Pruebas y resultados computacionales. Prueba 10. Comparación de las redes neuronales.*

Cabe destacar que este proceso de reconocimiento se aplica de igual forma para la red de perceptrón multicapa y la red de neuronas de base radial, el éxito de esta fase depende de la fase de aprendizaje, mientras mejor se realice el proceso de entrenamiento, mejor serán los resultados obtenidos.

CAPÍTULO 5

FASE DE IMPLEMENTACIÓN Y PRUEBAS

En la fase de implementación se toma el resultado de las fases anteriores para generar el código final, es aquí donde se selecciona un lenguaje de programación, según las especificaciones del diseño y de las propiedades del lenguaje, para la construcción de un producto totalmente operacional, donde los componentes y todas sus características sean integrados y probados.

5.1. Elección del lenguaje de programación

El lenguaje elegido para la codificación de la aplicación fue Java, usando el JDK 1.6. A continuación se presentan muchas de las características por la cual se escogió este lenguaje de programación:

- Orientado a objetos: Java fue diseñado como un lenguaje orientado a objetos desde el principio. Los objetos se agrupan en estructuras encapsuladas, tanto sus datos como los métodos que manipulan esos datos.
- Distribuido: Java proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir sockets y establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas.
- Interpretado y compilado a la vez: Java es compilado, en la medida en que su código fuente se transforma en una especie de código máquina, los bytecodes, semejantes a las instrucciones de ensamblador. Por otra parte, es interpretado, ya que los bytecodes se pueden ejecutar directamente sobre

cualquier máquina a la cual se hayan portado el intérprete y el sistema de ejecución en tiempo real.

- Robusto: Java fue diseñado para crear software altamente fiable. Para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Sus características de memoria liberan a los programadores de una familia entera de errores, ya que se ha prescindido por completo de los punteros, y la recolección de basura elimina la necesidad de liberación explícita de memoria.
- Indiferente a la arquitectura: Java está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos de red, desde Unix a Windows NT, pasando por Mac y estaciones de trabajo, sobre arquitecturas distintas y con sistemas operativos diversos.
- Portable: la indiferencia a la arquitectura representa sólo una parte de su portabilidad. Además, Java especifica los tamaños de sus tipos de datos básicos y el comportamiento de sus operadores aritméticos, de manera que los programas son iguales en todas las plataformas. Estas dos últimas características se conocen como la Máquina Virtual Java (JVM).
- Multihebra: hoy en día ya se ven como terriblemente limitadas las aplicaciones que sólo pueden ejecutar una acción a la vez. Java soporta sincronización de múltiples hilos de ejecución (multithreading) a nivel de lenguaje.
- Dinámico: el lenguaje Java y su sistema de ejecución en tiempo real son dinámicos en la fase de enlazado. Las clases sólo se enlazan a medida que son necesitadas. Se pueden enlazar nuevos módulos de código bajo demanda, procedente de fuentes muy variadas, incluso desde la Red.

El entorno de desarrollo integrado (IDE) utilizado fue JCreator Pro V3, por su facilidad de uso y bajos requerimientos. El cual permite una programación total por parte del usuario, facilitando su entendimiento.

5.2. Codificación

Durante esta etapa se realiza la tarea denominada comúnmente como programación; que consiste en llevar a código fuente, en el lenguaje de programación elegido, en este caso, Java, todo lo diseñado en las fases anteriores, siguiendo por completo los lineamientos impuestos en el diseño y considerando siempre los requisitos funcionales y no funcionales.

El paradigma de la programación orientada a objetos fue el empleado para la codificación de la aplicación, por lo tanto, se realizó una abstracción de los objetos con sus atributos para representar sus características o propiedades, y sus métodos para representar los comportamientos o acciones que realizan, donde, todas las propiedades y métodos comunes a los objetos se encapsularon en clases.

La programación o codificación de la aplicación se llevo a cabo cumpliendo ciertas normas como son: el uso de nombres significativos para las clases, los atributos y los métodos, organización del código mediante el uso de tabulaciones y espacios que brinden claridad tanto al programador como a un lector, inicialización de los atributos al principio de la clases e información de los diseñadores al principio de cada clase.

Para la integración de todos los módulos que conforman la aplicación se hizo uso de los siguientes métodos:

El método de integración de arriba hacia abajo o descendente: consiste en integrar la aplicación empezando con el módulo principal, y a este se le irá adjuntando los módulos subordinados, probando la aplicación cada vez que se le añada un módulo, usando al programa principal como el manejador de las pruebas necesarias.

El método de integración de abajo hacia arriba o ascendente: consiste en probar primero los módulos inferiores, que no necesitan la ayuda de otro módulo, sirviendo como soporte de otro procedimiento. De esta forma se van creando grupos de módulos cada vez que se une un módulo con otro, hasta que, por último, se integran todos estos grupos con el programa principal. Es necesario realizar por los menos una prueba para cada integración.

El método utilizado para la integración de la aplicación fue el método descendente, se consideró que el establecer una serie de niveles de mayor a menor complejidad para la resolución del problema daría como resultado una aplicación óptima y con un alto desempeño, evitando problemas durante la integración de todos los módulos.

5.3. Implementación de la interfaz gráfica

La interfaz gráfica de usuario (Graphical User Interface, GUI) será el artefacto tecnológico de la aplicación que posibilitará, a través del uso y la representación visual, una interacción amigable con la aplicación informática. La interfaz gráfica de usuario utiliza un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles de la aplicación.

Para la implementación de la interfaz gráfica de usuario para la aplicación de reconocimiento facial mediante el uso de redes neuronales artificiales, se hizo uso de los requerimientos y los casos de uso del sistema, además de esto su diseño gráfico se pensó sobre la base de una apariencia profesional, agradable y sencilla al momento de ser manipulada por cualquier tipo de usuario, permitiendo a usuarios inexpertos su fácil uso y a usuarios expertos la posibilidad de continuar realizando investigación avanzadas, permitiendo la realización de pruebas que contribuyan a la obtención de nuevos conocimientos.

Como se puede observar en la figura 5.1, se muestra la interfaz gráfica de usuario correspondiente a la fase de aprendizaje de la aplicación.

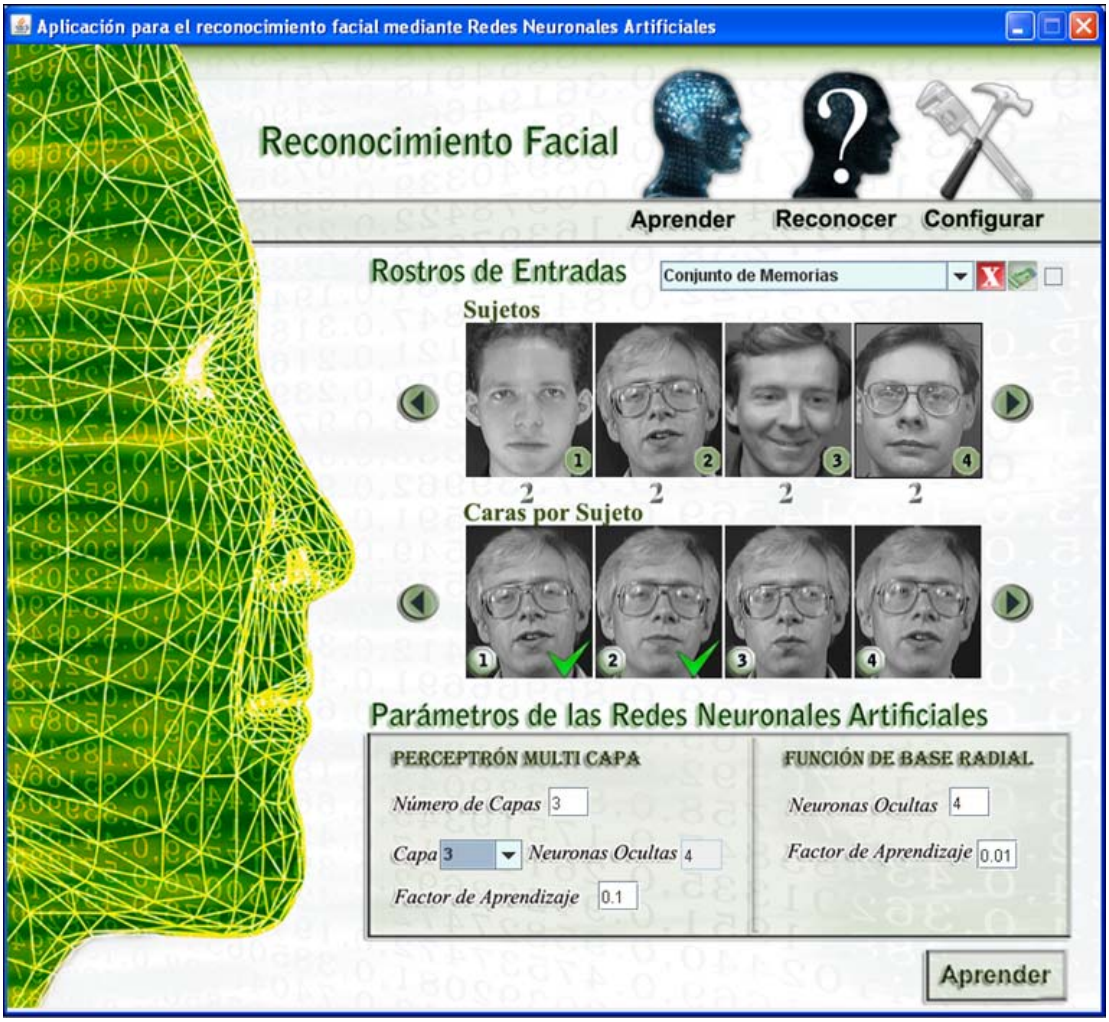

Figura 5.1. *Interfaz gráfica de usuario – fase de aprendizaje. Fuente: Propia.*

En esta interfaz es donde el usuario le indica cuantos sujetos deben aprenderse las redes neuronales artificiales y cuantas caras de cada uno, e indica los parámetros con los que desea que trabaje la red. También es posible indicarle a la red que haga uso de alguna memoria anterior para agilizar el aprendizaje o para cargarlo en las

redes. A continuación se explica con más detalle cada parte de la interfaz de usuario mostrada en la figura 5.1.

Para cambiar entre las diferentes pantallas que ofrece la aplicación se hace uso de los botones aprender, reconocer y configurar (figura 5.2). El botón aprender se encarga de mostrar en pantalla la interfaz gráfica de usuario correspondiente a la fase de aprendizaje, es decir, la mostrada en la figura 5.1. El botón reconocer se encarga de mostrar la pantalla correspondiente a la fase de reconocimiento (figura 5.7) y el botón configurar se encarga de mostrar una pantalla para la configuración de algunos parámetros de la aplicación (figura 5.11).

Figura 5.2. *Botones aprender, reconocer y configurar. Fuente: Propia.*

En la figura 5.3 el combo box muestra una lista de los diferentes aprendizajes realizados hasta el momento por la aplicación, estos aprendizajes son almacenados y guardados según su calidad, es decir, según su porcentaje de generalización, formando de esta manera un conjunto de memorias que pueden ser utilizadas posteriormente para cargar un aprendizaje realizado con anterioridad o para tratar de mejorar el porcentaje de generalización obtenido hasta ese momento. Si se desea cargar un aprendizaje realizado anteriormente, sin la necesidad de entrenar nuevamente, sólo basta con seleccionarlo de la lista del combo box y activar la casilla de verificación, la cual indica que debe omitirse el entrenamiento. Ahora bien, si se

desea mejorar un entrenamiento se debe indicar cual y no se debe activar la casilla de verificación. El botón eliminar se utiliza cuando se desea borrar un aprendizaje de la lista del combo box y el botón limpiar deselecciona todos los sujetos y caras seleccionadas por el usuario hasta el momento.

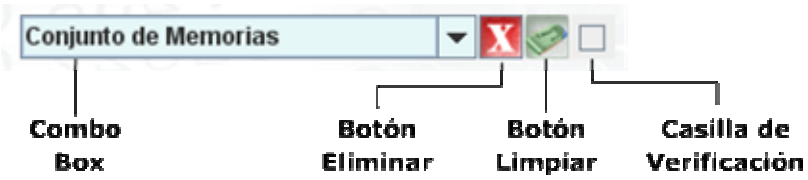

Figura 5.3. *Memorias de la aplicación. Fuente: Propia.*

Como se puede observar en la figura 5.4, ésta consta de dos apartados, sujetos y caras, en el apartado de sujetos se indica el sujeto al cual se le desea seleccionar alguna cara para que forme parte del conjunto de entrenamiento, para esto se presiona clic sobre el rostro del sujeto y posteriormente se presiona sobre las caras que desean seleccionar, las caras seleccionadas tendrán una flecha verde y el número de caras seleccionadas será indicado por el contador de caras.

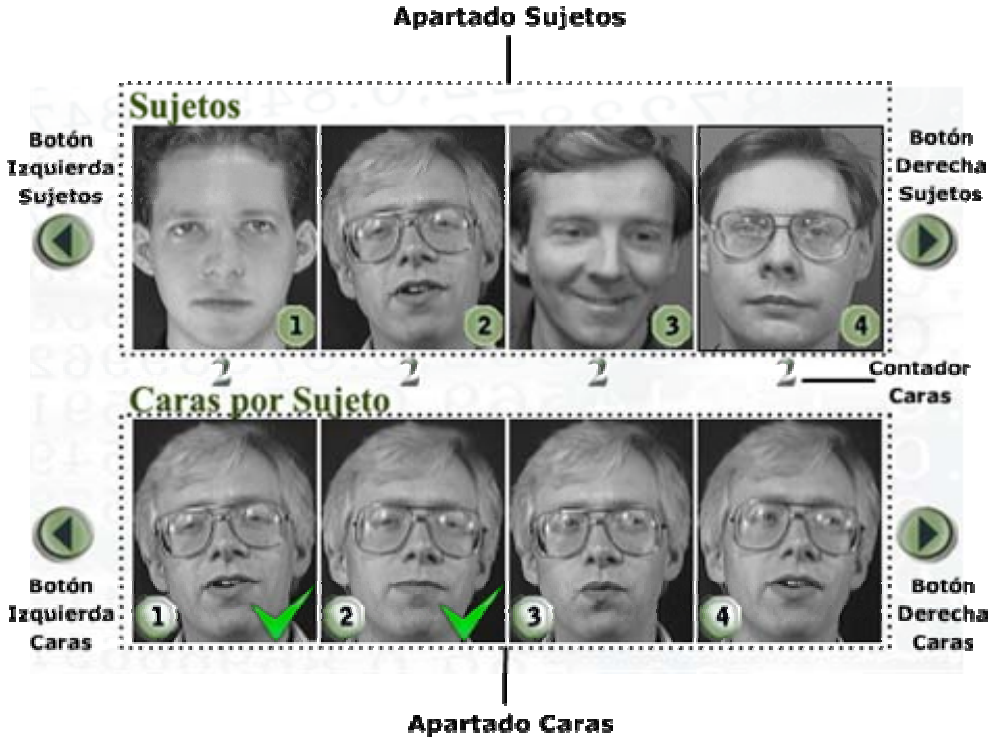

Figura 5.4. *Selección de rostros para el entrenamiento. Fuente: Propia.*

También se puede observar en la figura 5.4 que cada apartado posee dos botones a sus extremos, estos botones tienen como función permitir desplazarse por el conjunto de rostros y de caras utilizadas por la aplicación.

En cuanto a la configuración de los parámetros de las redes neuronales artificiales, en la figura 5.5 se muestra los diferentes cambios que se pueden realizar, en cuanto al perceptrón multicapa se puede indicar el número de capas que se desea utilizar, luego el número de neuronas por cada capa y el factor de aprendizaje. Para la red de base radial, se puede indicar el número de neuronas ocultas y el factor de aprendizaje.

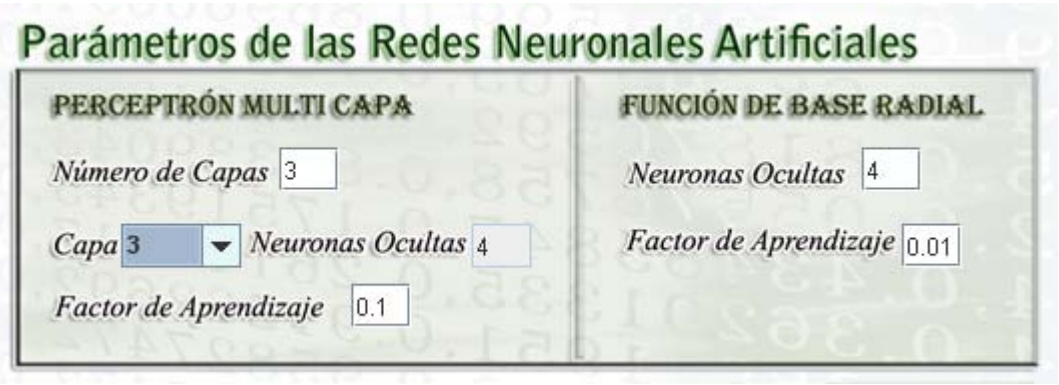

Figura 5.5. *Parámetros de las redes neuronales artificiales. Fuente: Propia.*

Por último, como se muestra en la figura 5.6, el botón aprender es el que se encarga de ordenar el aprendizaje de todos los rostros seleccionados, cabe destacar que para realizar un entrenamiento se debe seleccionar más de un rostro.

Figura 5.6. *Botón aprender. Fuente: Propia.*

Como se puede observar en la figura 5.7, se muestra la interfaz gráfica de usuario correspondiente a la fase de reconocimiento de la aplicación. Al igual que en la figura 5.1, ésta también posee los botones de aprender, reconocer y configurar. Cabe destacar que la pantalla mostrada en la figura 5.7 aparece cuando se presiona sobre el botón reconocer.

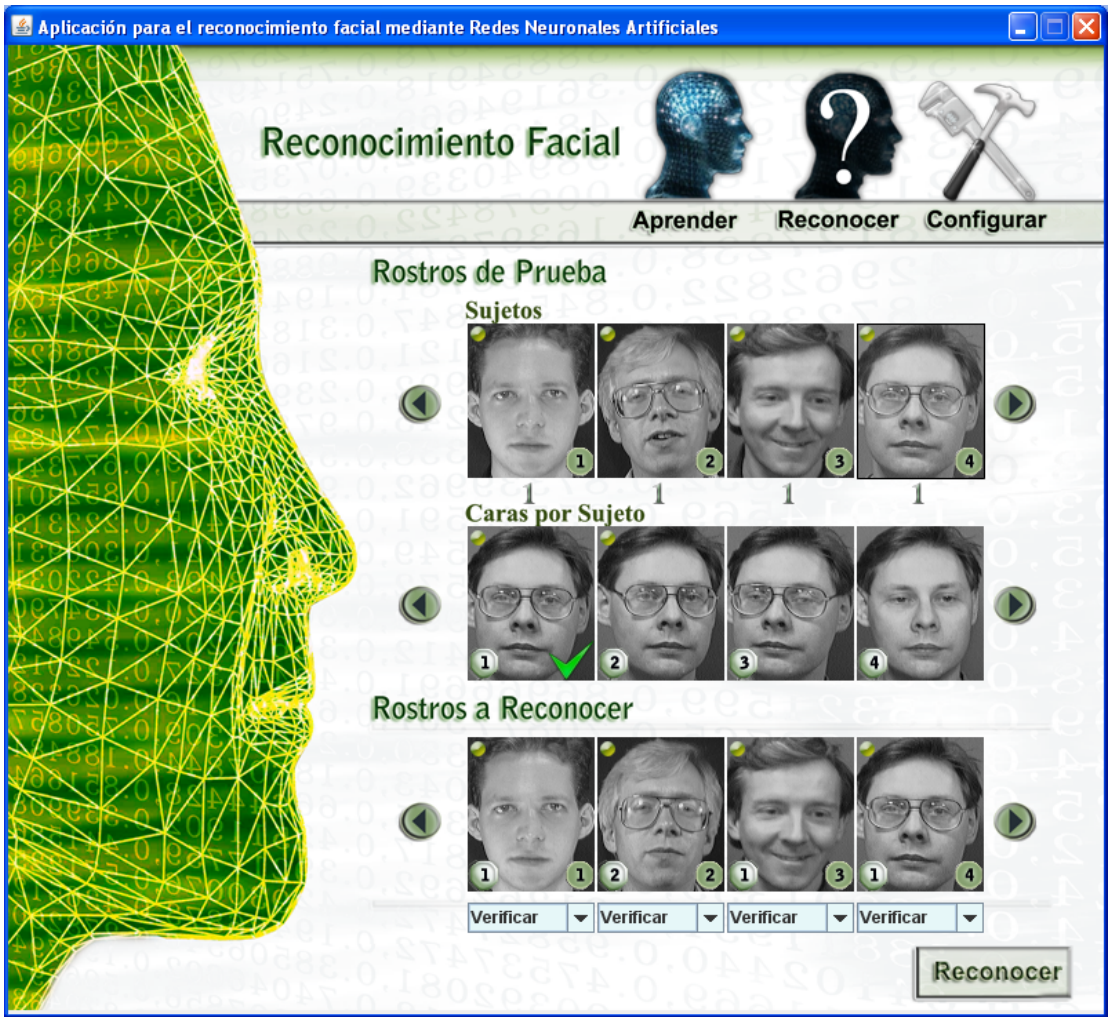

Figura 5.7. *Interfaz gráfica de usuario – fase de reconocimiento. Fuente: Propia.*

En la interfaz que se muestra en la figura 5.7 es donde el usuario indica que sujetos desea que la aplicación reconozca. Para indicar los sujetos que se desean identificar o verificar se deben seleccionar las caras de los sujetos, esto se realiza de igual forma que para la selección de los rostros de entrenamiento, como se muestra en la figura 5.4. Una vez seleccionados los rostros que se desean reconocer, como se muestra en la figura 5.8, se hace uso del combo box para verificar si un sujeto corresponde a otro en particular. En caso de no indicar ningún otro sujeto en el

combo box, entonces, se realizará una identificación del mismo. Para desplazarse entre los sujetos a reconocer se debe hacer uso de los botones a sus lados.

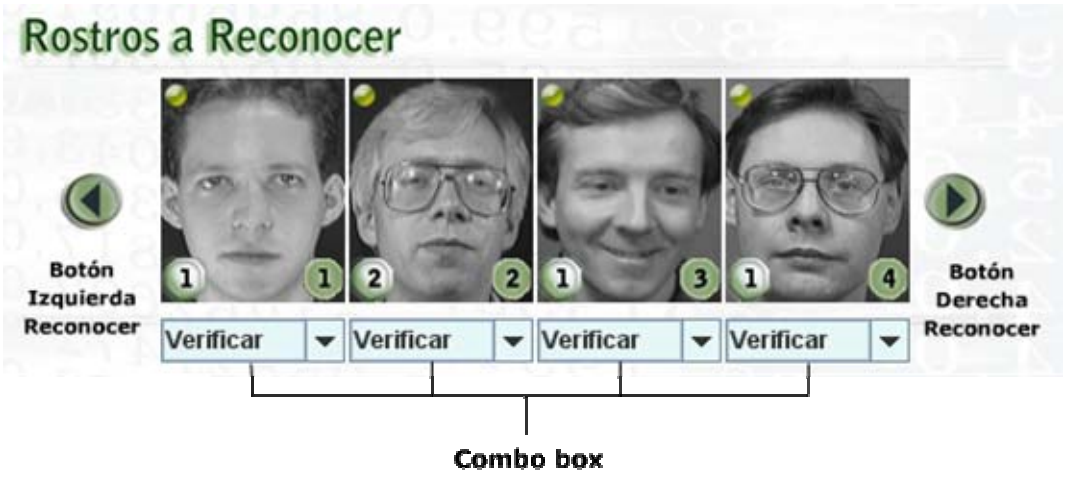

Figura 5.8. *Conjunto de rostros a reconocer. Fuente: Propia.*

Una vez indicados los rostros a reconocer para iniciar el proceso de reconocimiento se debe presionar el botón reconocer (figura 5.9).

Figura 5.9. *Botón reconocer. Fuente: Propia.*

Al finalizar el proceso de reconocimiento, se muestra una pantalla con el conjunto de rostros a reconocer y los resultados obtenidos del reconocimiento, como se muestra en la figura 5.10.

En la interfaz mostrada en la figura 5.10, se muestra inicialmente en la parte superior los rostros a reconocer, estos presentan una numeración, el número de la izquierda corresponde al número cara y el número de la derecha al número de sujeto, podría decirse que representa su identificador o nombre, ya que todas las caras con este número corresponden al mismo sujeto. Al igual que en los casos anteriores, para

desplazarse por el conjunto de rostros a reconocer se hace uso de los botones a sus lados.

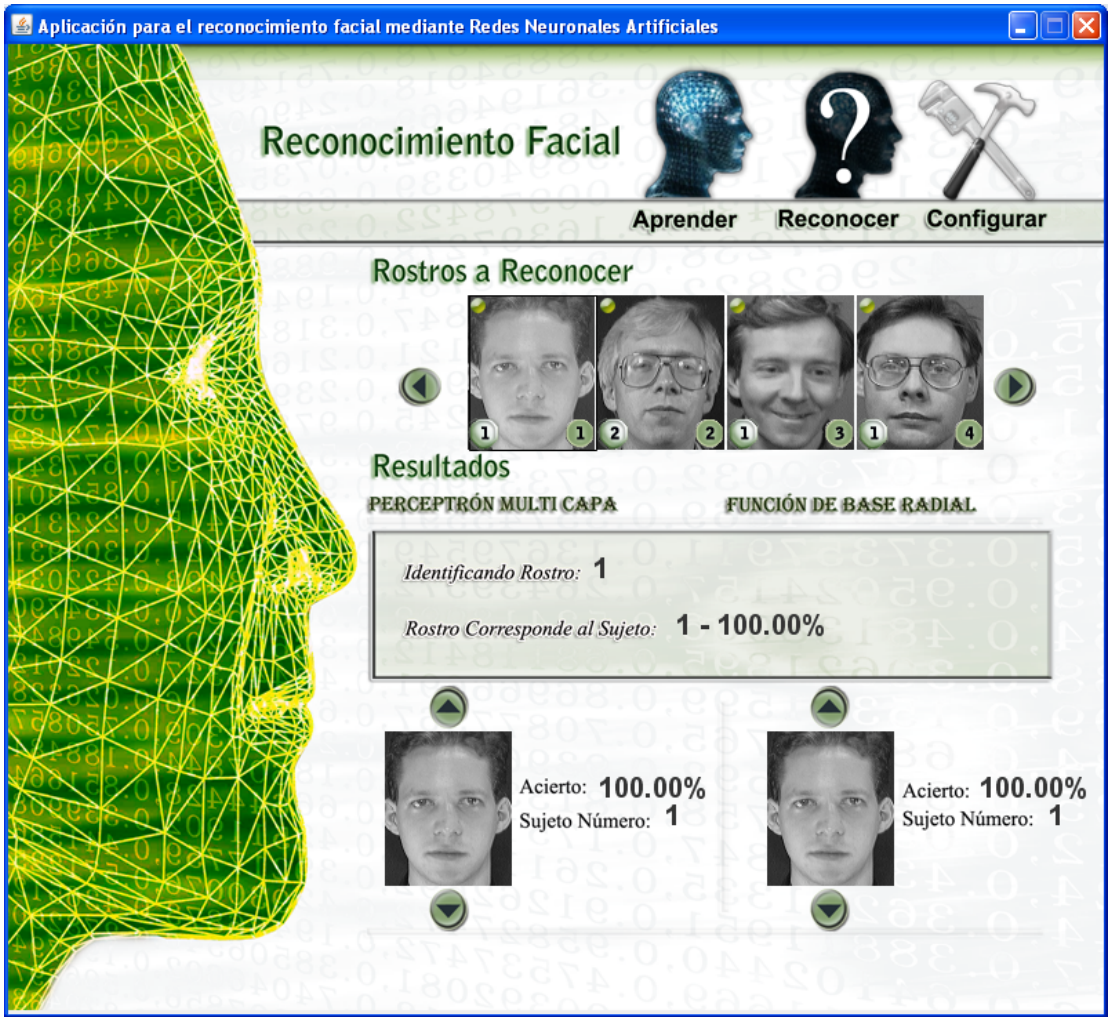

Figura 5.10. *Interfaz de usuario – reconociendo. Fuente: Propia.*

Para conocer los resultados arrojados por la aplicación, más específicamente por cada red neuronal, se debe presionar sobre un rostro. Como se puede observar en la figura 5.10 el primer rostro a reconocer se encuentra seleccionado y los resultados correspondientes se muestran en pantalla, en este caso tanto la red perceptrón multicapa como la red de neuronas de base radial han determinado que el sujeto

seleccionado corresponde al sujeto número 1 con un acierto del 100%. Este proceso se realiza de igual forma para todos los rostros a ser reconocidos.

Para finalizar con las interfaces de usuarios, se muestra a continuación (figura 5.11) la interfaz de usuario que se encarga de la configuración de algunos parámetros, como son: el número de sujetos que la aplicación será capaz de manejar y el número de caras por sujeto. También brinda alguna información referente a la ubicación donde deben estar almacenadas las imágenes y la resolución de éstas.

Figura 5.11. *Interfaz gráfica de usuario – Configuración. Fuente: Propia.*

5.4. Pruebas y resultados computacionales

En este apartado se realizan las pruebas necesarias para conseguir implementar la aplicación de reconocimiento facial, logrando el mejor rendimiento. Para esto es necesario realizar pruebas para determinar: el número óptimo de neuronas en las capas ocultas, funciones de activación, parámetros importantes como, el momento, el error mínimo, factor de aprendizaje, así como también las heurísticas empleadas para la determinación de los centros y amplitudes (para la red de neuronas de base radial), la modificación de los umbrales con Backpropagation (para la red perceptrón multicapa) y la inicialización de los pesos. Todos estos son factores que juegan papel muy importante a la hora de implementar la aplicación, por lo tanto, se desea en este apartado mostrar los resultados haciendo uso de las diferentes posibilidades, para la obtención de redes neuronales capaces del reconocimiento facial.

También se mostrará en este apartado los resultados finales obtenidos para la red perceptrón multicapa y la red de neuronas de base radial, así como comparaciones entre ellas, para medir velocidad y desempeño.

Todas las pruebas se realizarán en un ordenador con las siguientes características: Pentium 4 a 3000MHZ con 512MB de RAM y Windows XP sp2 como sistema operativo.

En caso de no indicar lo contrario, los parámetros que utilizarán las redes neuronas artificiales para realizar las pruebas serán los indicados en la tabla 5.1.

Tabla 5.1. *Parámetros iniciales de las redes neuronales artificiales. (1/2).*

Parámetros	Red perceptrón multicapa	Red de base radial
Nº de neuronas	20	Nº de clases
Función de activación	Potencia	Gausiana
Factor de aprendizaje	0.1	0.01

Fuente: Propia.

Tabla 5.1. ***Parámetros iniciales de las redes neuronales artificiales. (2/2).***

Error mínimo	0.1	0.000001
Factor momento	0.9	0.7
Pesos	[-0.05, 0.05]	[-0.05,0.05]
Heurísticas	No aplica	Media Geométrica

Fuente: Propia.

Para todas las pruebas realizadas en este apartado se toman en cuenta diversas características como: el número de rostros a memorizar por la red, el tiempo y el número de iteraciones durante el proceso de aprendizaje, y como factor fundamental, el porcentaje de generalización, que permite determinar que tan buenos son los factores evaluados, por lo tanto, el porcentaje de generalización tendrá el mayor peso a la hora de tomar una decisión.

Para la obtención de los resultados se realizaron 30 pruebas separadas en 3 grupos de 10, se promediaron los resultados de cada grupo y el de mayor promedio se presentó en la primera fila de cada prueba y en la segunda fila se especificó el resultado más alto durante las 30 pruebas, con lo que se desea determinar que número de neuronas, función de activación, factor de aprendizaje, momento, pesos y heurísticas conviene utilizar durante el proceso de aprendizaje.

Cabe destacar que a medida que se obtengan los parámetros que mejor funcionan con las redes, estos se irán incorporando en su estructura para realizar cálculos cada vez mejores.

Prueba 1. Número de neuronas. En esta prueba se busca el número de neuronas que mejor responde ante el reconocimiento de un conjunto de rostros, para esto se varía el número de neuronas, de acuerdo al número de sujetos a memorizar, es decir, para 10 x 8 = 80 (10 sujetos, 8 caras, 80 rostros totales), el número de neuronas a utilizar podría ser de dos neuronas por sujeto, o una por sujeto o una neurona por

cada dos sujetos, en otras palabras sería 10x2, 10 y 10/2, con lo que se tendría 20, 10 y 5 neuronas.

Tabla 5.2. *Búsqueda del mejor número de neuronas para el perceptrón multicapa.*

Nº neuronas	Nº Rostros	Nº iteraciones	Tiempo	% de generalización
20	10 x 8 = 80	15584	0.431s	97.80%
		14560	0.375s	98.64%
10	10 x 8 = 80	17224	0.272s	97.86%
		16240	0.234s	99.14%
5	10 x 8 = 80	30024	0.262s	97.32%
		58080	0.500s	98.64%
40	20 x 4 = 80	99032	6.884s	88.80%
		105600	7.469s	91.96%
20	20 x 4 = 80	116928	3.946s	86.40%
		160640	4.938s	88.63%
10	20 x 4 = 80	169233	3.472s	85.32%
		203600	4.000s	88.34%
60	30 x 4 = 120	160960	21.083s	81.05%
		137280	19.328s	82.58%
30	30 x 4 = 120	194280	13.098s	79.31%
		202800	14.031s	80.68%
15	30 x 4 = 120	384680	14.484s	75.13%
		494040	17.890s	76.38%

Fuente: Propia.

Como se puede observar en la tabla 5.2, para el primer caso donde el número de rostros es 10 x 8 = 80, se emplea dos neuronas por sujeto, con lo que se dispone de 20 neuronas para el aprendizaje, en estos casos en particular, donde se utiliza dos neuronas por sujeto se obtienen buenos resultados en comparación a los casos donde se utiliza una neurona por cada dos sujetos, sin embargo, para los casos donde se emplea una neurona por sujeto se obtienen también buenos resultados, con la ventaja de realizarlo en un tiempo mucho menor, por lo tanto, para esta aplicación se optó por definir un número fijo de 20 neuronas, capaces de enfrentarse a diferentes números de

rostros y obtener buenos resultados en cuanto a tiempo y calidad, sin embargo, para obtener mayor calidad en lo que respecta a generalización, se puede sacrificar el tiempo y emplear dos neuronas por sujeto obteniendo en su mayoría los mejores resultados.

Como se puede observar a medida que el número de neuronas es mayor para un mismo problema, el número de iteraciones disminuye pero el tiempo de procesamiento aumenta, a primera vista podría parecer que debido a que el número de iteraciones es menor, debería resolver el problema en menor tiempo, pero en realidad el tiempo es mayor, debido al gran número de operaciones realizadas entre todos los nodos de las neuronas.

En cuanto a la elección del número de neuronas apropiadas para la red de neuronas de base radial, se utilizó el mismo criterio que para el perceptrón multicapa, con la única diferencia que se agregó un nuevo caso empleando el número de rostros como el número de neuronas de la red, en la tabla 5.3 se observan los resultados obtenidos.

Tabla 5.3. *Búsqueda del mejor número de neuronas para la red de base radial. (1/2)*

Nº neuronas	Nº Rostros	Nº iteraciones	Tiempo	% de generalización
80	10 x 8 = 80	5626	15.252s	99.00%
		5842	15.453s	99.00%
20	10 x 8 = 80	17745	14.847s	96.80%
		9758	7.891s	99.00%
10	10 x 8 = 80	4566	2.526s	78.88%
		2277	1.109s	90.56%
5	10 x 8 = 80	1646	0.571s	46.53%
		1322	0.453s	57.01%

Fuente: Propia.

Tabla 5.3. ***Búsqueda del mejor número de neuronas para la red de base radial. (2/2)***

80	20 x 4 = 80	12561	67.343s	81.95%
		12204	69.906s	82.25%
40	20 x 4 = 80	53033	16.348s	89.83%
		23713	68.672s	94.00%
20	20 x 4 = 80	9761	16.348s	69.42%
		7349	12.032s	73.25%
10	20 x 4 = 80	5006	4.782s	30.40%
		4063	4.078	35.76%
120	30 x 4 = 120	10206	204.018s	81.96%
		10345	202.922s	82.33%
60	30 x 4 = 120	34452	332.315s	80.89%
		36282	345.687s	83.85%
30	30 x 4 = 120	11766	61.453s	55.55%
		11696	61.469s	58.00%
15	30 x 4 = 120	3987	11.854s	24.88%
		3742	11.297s	26.33%
320	40 x 8 = 320	6752	1926.343s	97.37%
		6910	1930.375s	97.50%
80	40 x 8 = 320	9068	483.859s	77.89%
		8793	449.359s	82.09%

Fuente: Propia.

Para el caso de la red de base radial resulta difícil determinar un valor fijo de neuronas, ya que, este tipo de red realiza aproximaciones locales de los patrones de aprendizaje, por lo tanto, si el número de neuronas es menor al número de sujetos, lo más probable es que el porcentaje de generalización disminuya notablemente, como se puede observar en la tabla 5.3 en los casos donde el número de neuronas es menor al número de sujetos, por el contrario, los mejores resultados se pueden observar en los casos donde el número de neuronas es igual al número de rostros y cuando el numero de neuronas es el doble del número de sujetos.

Para los casos donde el número de rostros no es muy grande, no existe gran diferencia en usar el número de neuronas igual al número de rostros o en usar dos

neuronas por sujeto, sin embargo, en los casos donde el número de rostros aumenta, se puede notar que dos neuronas por sujeto da como resultado tiempos de convergencias mayores y porcentajes menores, como se puede observar en el caso donde se empleo 60 neuronas para resolver el problema de 30 x 4, notándose que el número de iteraciones y el tiempo de convergencia es mayor que al usar 120 neuronas, esto se puede observar con mayor claridad en el caso donde se emplea 320 neuronas para resolver el problema de 40x8, demostrando de esta manera que resulta más apropiado utilizar un número de neuronas igual al número de rostros a entrenar, para garantizar de esta manera una alta capacidad de generalización.

La selección del número de neuronas para esta red es mucho más complicada que para el perceptrón multicapa, ya que, una mala elección podría ocasionar tiempos de convergencias muy grandes o bajos porcentajes de generalización, ya que, el número de neuronas esta estrechamente ligado al conjunto de rostros que se desea aprender.

Cabe destacar que estas pruebas tienen como fin encontrar el número de neuronas que esté más acorde con las redes neuronales, no significa que el número de neuronas que se determine en estas pruebas sea definitivamente el mejor, ya que, sólo se trata de hacer una aproximación del número de neuronas que se pueden utilizar para conseguir un aprendizaje rápido y con un alto grado de generalización, es importante señalar que el deseo de encontrar un número de neuronas que disminuya el tiempo de aprendizaje, tal vez, no sea de gran importancia, sin embargo, para este estudio, lo es, debido a la facilidad de análisis en cuanto a las estructuras de las redes y sus componentes.

Prueba 2. Función de activación. En esta prueba se busca la función de activación que tenga el mejor comportamiento y arroje los mejores resultados durante el reconocimiento facial, tanto para la red perceptrón multicapa como para la red de

neuronas de base radial. En las tablas 5.4 y 5.5 se muestran los resultados obtenidos para diferentes funciones de activación.

Tabla 5.4. *Búsqueda de la mejor función de activación para el perceptrón multicapa.*

Función activación	Nº Rostros	Nº iteraciones	Tiempo	% generalización
Potencia	10 x 8 = 80	15416	0.400s	97.86%
		15600	0.391s	99.14%
	10 x 1 = 10	3704	0.068s	61.01%
		5080	0.078s	66.74%
Sigmoidal	10 x 8 = 80	14160	0.429s	97.94%
		13600	0.407s	99.50%
	10 x 1 = 10	2872	0.062s	64.90%
		2950	0.047s	67.00%
Tangente hiperbólica	10 x 8 = 80	22960	0.721s	97.20%
		22560	0.735s	98.00%
	10 x 1 = 10	1180	0.029s	65.60%
		1220	0.032s	69.02%
Seno	10 x 8 = 80	201056	4.675s	97.31%
		203200	4.907s	99.00%
	10 x 1 = 10	62556	1.260s	51.30%
		54550	1.093s	60.00%

Fuente: Propia.

Como se puede observar en la tabla 5.4, se realizó para cada función de activación dos pruebas variando el número de rostros, en una se utilizó un número de caras elevado (8) y en la otra el número mínimo de caras por sujeto (1), con el objeto de evaluar el comportamiento de las funciones de activación en cuanto a la generalización del problema, como se muestra en el caso de la función sigmoidal, ésta obtiene los mejores resultados cuando se trata de un número elevado de caras por sujeto, sin embargo, no presenta los mejores resultados para un número menor de caras, ya que, los mejores resultados fueron arrojados por la función tangente hiperbólica, a pesar de esto, se decidió utilizar la función sigmoidal como la función

de activación predeterminada para la aplicación, por su mayor velocidad y menor número de iteraciones con respecto a la función hiperbólica, lo que trae por consiguiente un mayor control y estabilidad al enfrentarse con problemas de mayor magnitud.

Las pruebas realizadas para la red de neuronas de base radial, que se muestran en la tabla 5.5, siguen el mismo criterio de las pruebas descritas en la tabla 5.4, con lo que se desea determinar la función de activación de mejor comportamiento en cuanto a tiempo de procesado y calidad.

Tabla 5.5. *Búsqueda de la mejor función de activación para la red de base radial.*

Función activación	Nº Rostros	Nº iteraciones	Tiempo	% generalización
Gausiana	10 x 8 = 80	5626	15.252s	99.00%
		5842	15.453s	99.00%
	10 x 1 = 10	98223	9.887s	76.01%
		33359	3.187s	76.02%
Inversa cuadrática	10 x 8 = 80	5906	4.520s	92.73%
		5460	4.187s	96.00%
	10 x 1 = 10	146348	14.296s	76.00%
		134058	12.687s	76.00%
Inversa multicuadrática	10 x 8 = 80	3594	2.781s	93.81%
		2770	2.125s	99.00%
	10 x 1 = 10	425963	43.850s	74.00%
		28246	2.922s	74.00%

Fuente: Propia.

Como se puede observar en la tabla 5.5, la función gausiana tiene el mejor comportamiento tanto con 80 como con 10 rostros, aunque las diferencias sean pequeñas, se podría destacar su tiempo de procesamiento y su menor número de iteraciones, en el caso de 10 rostros, sin embargo, como se puede ver, la función cuadrática y la multicuadrática presentan un pequeño número de iteraciones al presentarle 80 rostros, a diferencia de la gausiana, a pesar de esto, se considera que

de todas las funciones la que mayor estabilidad ofrece, en cuanto a tiempo de procesamiento y rendimiento, es la función gausiana por arrojar los mejores resultados en relación a las demás funciones.

La elección de la función de activación muchas veces está estrechamente ligada al problema que se desea resolver, pero, se ha notado que durante las pruebas de funciones de activación, todas han tenido un comportamiento aceptable, por lo tanto, decidir que función de activación será empleada para la aplicación de reconocimiento facial no resulta problemático.

Para la selección de las funciones de activación se tomó en cuenta como factor fundamental el tiempo de convergencia y la capacidad de generalización, con el fin de obtener redes neuronales que permitan realizar estudios, en cuanto a los diferentes parámetros que intervienen en su estructura y funcionamiento, facilitando y agilizando la investigación por las diferentes posibilidades que ofrece una red neuronal, a demás de proveer una solución elegante al problema de reconocimiento facial.

Prueba 3. Factor de Aprendizaje. En esta prueba se busca el mejor factor de aprendizaje que permita una buena generalización del problema. Los resultados para la red perceptrón multicapa y la red de neuronas de base radial se muestran en las tablas 5.6 y 5.7 respectivamente.

Tabla 5.6. *Búsqueda del mejor factor de aprendizaje para el perceptrón multicapa. (1/2)*

Factor aprendizaje	Nº iteraciones	Tiempo	% de generalización
0.01	93320	2.146s	95.68%
	97680	2.110s	97.16%

Fuente: Propia.

Tabla 5.6. ***Búsqueda del mejor factor de aprendizaje para el perceptrón multicapa. (2/2)***

0.02	42600	0.999s	95.71%
	39760	0.906s	97.29%
0.03	31880	0.931s	95.91%
	28720	0.734s	98.18%
0.04	30008	0.788s	96.21%
	44640	1.140s	97.83%
0.05	29704	0.896s	96.01%
	29600	0.875s	98.62%
0.06	25064	0.698s	96.83%
	24480	0.704s	99.14%
0.07	21872	0.626s	97.24%
	20400	0.688s	99.00%
0.08	18272	0.518s	96.85%
	21040	0.593s	98.14%
0.09	16216	0.462s	97.61%
	19040	0.500s	99.50%
0.1	15560	0.450s	97.73%
	14880	0.422s	99.00%
0.2	42520	0.869s	95.97%
	38640	0.812s	97.43%
0.3	28848	0.622s	96.57%
	21520	0.453s	97.64%

Fuente: Propia.

Como se puede observar en la tabla 5.6, se realizan una serie de pruebas para determinar cual factor está más acorde con la estructura de la red perceptrón multicapa, se puede notar que a medida que el factor de aprendizaje aumenta, el porcentaje de generalización también, deteniendo su aumento en el factor 0.1, con lo que se optó por utilizar este factor como el predeterminado de la aplicación.

A continuación en la tabla 5.7 se presentan los resultados obtenidos al variar un conjunto de factores en la red de neuronas de base radial.

Tabla 5.7. *Búsqueda del mejor factor de aprendizaje para la red de base radial.*

Factor aprendizaje	Nº iteraciones	Tiempo	% de generalización
0.01	11635	9.271s	95.91%
	8427	6.438s	99.00%
0.02	10442	8.323s	90.39%
	5721	4.531s	99.00%
0.03	5251	4.415s	91.96%
	4516	3.813s	99.00%
0.04	4172	3.532s	91.87%
	2546	2.156s	98.00%
0.05	2668	2.431s	81.45%
	1672	1.468s	99.00%
0.06	2739	2.332s	81.44%
	1544	1.328s	97.00%
0.07	1810	1.621s	84.07%
	3843	3.250s	95.42%
0.08	3216	2.720s	70.53%
	2176	2.032s	85.39%
0.09	1873	1.598s	67.95%
	1281	1.093s	81.73%
0.1	1923	1.662s	65.53%
	2139	1.828s	81.30%
0.2	7460	5.650s	34.00%
	2681	2.015s	51.53%
0.3	25477	19.281s	11.95%
	58876	47.797s	19.75%

Fuente: Propia.

Como se puede observar en la tabla 5.7, a diferencia del perceptrón multicapa, la red de neuronas de base radial trabaja muy bien con factores de aprendizaje pequeños, como es el caso con el factor 0.01, y como se puede ver a medida que este factor aumente el porcentaje de generalización disminuye y con factores muy grandes es notable su mala generalización, por lo tanto, se decidió emplear un factor de aprendizaje de 0.01, por presentar los mejores resultados.

Cabe destacar que durante la realización de estas pruebas, factores de aprendizajes muy grandes, entre 0.4 y 0.9 provocaban inestabilidad en las redes, evitando el aprendizaje y por consiguiente el reconocimiento.

Prueba 4. Error mínimo. En esta prueba se desea encontrar el error mínimo (denominado error cuadrático medio) empleado como condición de parada en la red perceptrón multicapa y en la red de base radial, el objetivo es determinar cual es el error que se les permitirá tener a las redes entre las salidas deseadas y las obtenidas. Para esto se probaron diversos errores como se puede observar en la tabla 5.8 y 5.9, variando los errores estratégicamente para determinar la influencia de éste durante el proceso de aprendizaje.

Tabla 5.8. *Búsqueda del mejor error para el perceptrón multicapa.*

Error	Nº Rostros	Nº iteraciones	Tiempo	% de generalización
0.0001	10 x 8 = 80	5525840	153.810s	98.90%
		4606320	137.906s	99.00%
0.001	10 x 8 = 80	713456	18.931s	98.70%
		778160	21.547s	99.00%
0.01	10 x 8 = 80	228210	5.029s	98.60%
		118880	3.234s	99.00%
0.1	10 x 8 = 80	14464	0.379s	97.36%
		13840	0.375s	99.00%
0.2	10 x 8 = 80	12296	0.338s	97.00%
		11440	0.328s	98.00%
0.3	10 x 8 = 80	10520	0.306s	91.71%
		10160	0.297s	93.68%

Fuente: Propia.

Como se puede observar en la tabla 5.8, a medida que el error aumenta el porcentaje de generalización disminuye, con errores muy pequeños (0.0001), el porcentaje de generalización obtenido es muy bueno, pero, el tiempo de procesamiento o convergencia es bastante elevado con relación a los demás tiempos, esto ocurre también con el número de iteraciones. Ahora bien, para determinar cual de todos los errores conviene utilizar, se tomó en cuenta el tiempo y el porcentaje,

determinando que un factor de 0.1 resulta bastante bueno y emplea tiempos extremadamente pequeños durante el aprendizaje, por lo tanto, se optó por utilizar este factor.

Para estudios donde el tiempo de convergencia no sea relevante, se recomienda el uso de factores pequeños, ya que, estos no provocan en la red perceptrón multicapa, un sobre entrenamiento, como se pudo observar en las pruebas realizadas, sin embargo, se recomienda comparar los resultados al emplear diferentes errores durante el aprendizaje para garantizar el mejor funcionamiento.

Tabla 5.9. *Búsqueda del mejor error para la red de base radial. (1/2).*

Error	Nº Rostros	Nº iteraciones	Tiempo	% de generalización
0.000001	10 x 8 = 80	5984	16.125s	99.00%
		6062	18.047s	99.00%
	20 x 4 = 80	12057	70.190s	82.00%
		11935	66.031s	82.50%
0.00001	10 x 8 = 80	3160	8.656s	99.00%
		2804	7.781s	99.00%
	20 x 4 = 80	6477	37.643s	82.09%
		6097	34.031s	82.50%
0.0001	10 x 8 = 80	1570	4.568s	99.00%
		1606	4.516s	99.00%
	20 x 4 = 80	2742	15.093s	81.80%
		2890	16.172s	82.50%

Fuente: Propia.

Tabla 5.9. ***Búsqueda del mejor error para la red de base radial. (2/2).***

0.001	10 x 8 = 80	617	1.753s	99.00%
		623	1.906s	99.00%
	20 x 4 = 80	920	5.175s	81.50%
		883	4.844s	82.25%
0.01	10 x 8 = 80	111	0.406s	99.00%
		109	0.422s	99.00%
	20 x 4 = 80	225	1.331s	82.85%
		221	1.297s	82.50%
0.1	10 x 8 = 80	10	0.131s	93.48%
		12	0.156s	96.00%
	20 x 4 = 80	24	0.246s	68.90%
		24	0.234s	71.00%

Fuente: Propia.

Como se puede observar en la tabla 5.9, a medida que el error aumenta de tamaño, el número de iteraciones y el tiempo de convergencia disminuyen notablemente, sin embargo, los porcentajes de generalización no se ven afectados drásticamente, si no, hasta que alcanza el error con valor 0.1. Sobre la base de estos resultados, se determinó que el error que se le permitirá tener a la red durante el proceso de aprendizaje será de 0.01, por presentar el menor número de iteraciones, mejor tiempo de convergencia y por mantener el porcentaje de generalización.

Prueba 5. Factor momento. En esta prueba se desea obtener el factor de momento que ofrezca mayor rendimiento a la red perceptrón multicapa y la red de neuronas de base radial, para esto se realizaron varias pruebas con un rango de valores entre [0, 1], donde, cero indica la ausencia del factor momento en la red. Los resultados se muestran en las tablas 5.10 y 5.11.

Tabla 5.10. *Influencia del factor momento en la red perceptrón multicapa. (1/3)*

Momento	Nº Rostros	Nº iteraciones	Tiempo	% de generalización
0.0	5 x 8 = 40	20280	0.568s	98.00%
		19560	0.563s	98.00%
0.01	5 x 8 = 40	20600	0.578s	98.00%
		19800	0.562s	98.00%
0.02	5 x 8 = 40	20693	0.557s	98.00%
		21920	0.578s	98.00%
0.03	5 x 8 = 40	20946	0.562s	98.00%
		20960	0.563s	98.00%
0.04	5 x 8 = 40	21000	0.573s	98.00%
		22120	0.594s	98.00%
0.05	5 x 8 = 40	21506	0.614s	98.00%
		21200	0.656s	98.00%
0.06	5 x 8 = 40	20506	0.614s	98.00%
		20720	0.641s	98.00%
0.07	5 x 8 = 40	20133	0.578s	98.00%
		19400	0.532s	98.00%
0.08	5 x 8 = 40	20080	0.588s	98.00%
		20640	0.656s	98.00%
0.09	5 x 8 = 40	19386	0.573s	98.00%
		19880	0.625s	98.00%
0.1	5 x 8 = 40	20146	0.609s	98.66%
		22440	0.656s	100.0%
0.2	5 x 8 = 40	15813	0.422s	98.00%
		15800	0.422s	98.00%
0.3	5 x 8 = 40	14866	0.432s	98.00%
		14760	0.391s	98.00%
0.4	5 x 8 = 40	12066	0.359s	98.00%
		12120	0.328s	98.00%
0.5	5 x 8 = 40	10066	0.280s	98.00%
		9360	0.250s	98.00%

Fuente: Propia.

Tabla 5.10. ***Influencia del factor momento en la red perceptrón multicapa. (2/3)***

0.6	5 x 8 = 40	8493	0.245s	98.00%
		8200	0.235s	98.00%
0.7	5 x 8 = 40	6813	0.198s	98.00%
		6960	0.230s	98.00%
0.8	5 x 8 = 40	4866	0.135s	98.00%
		4760	0.141s	98.00%
0.9	5 x 8 = 40	3573	0.135s	98.00%
		3600	0.140s	98.00%
0.0	10 x 8 = 80	45200	1.161s	97.66%
		45040	1.156s	98.00%
0.01	10 x 8 = 80	47360	1.192s	97.33%
		49200	1.281s	98.00%
0.02	10 x 8 = 80	45066	1.130s	97.33%
		41040	1.016s	98.00%
0.03	10 x 8 = 80	47360	1.255s	97.33%
		48400	1.203s	98.00%
0.04	10 x 8 = 80	45920	1.224s	97.84%
		48000	1.171s	98.00%
0.05	10 x 8 = 80	44000	1.286s	97.66%
		43920	1.109s	98.00%
0.06	10 x 8 = 80	44080	1.125s	98.00%
		44960	1.125s	98.00%
0.07	10 x 8 = 80	43253	1.166s	98.00%
		44000	1.109s	98.00%
0.08	10 x 8 = 80	41786	1.083s	97.66%
		39360	1.047s	98.00%
0.09	10 x 8 = 80	42560	1.083s	97.66%
		39360	1.047s	98.00%
0.1	10 x 8 = 80	40160	0.995s	97.66%
		39280	0.969s	98.00%
0.2	10 x 8 = 80	34986	0.864s	97.66%
		36000	0.875s	98.00%

Fuente: Propia.

Tabla 5.10. ***Influencia del factor momento en la red perceptrón multicapa. (3/3)***

0.3	10 x 8 = 80	33333	0.953s	97.66%
		30720	0.797s	98.00%
0.4	10 x 8 = 80	27626	0.713s	97.66%
		28080	0.750s	98.00%
0.5	10 x 8 = 80	23493	0.620s	97.33%
		18640	0.453	97.00%
0.6	10 x 8 = 80	18133	0.557s	97.00%
		18640	0.453s	97.00%
0.7	10 x 8 = 80	14720	0.375s	97.66%
		15600	0.422s	98.00%
0.8	10 x 8 = 80	11066	0.317s	97.33%
		11040	0.343s	98.00%
0.9	10 x 8 = 80	7493	0.202s	97.66%
		7600	0.203s	99.00%

Fuente: Propia.

Como se puede observar en la tabla 5.10, en las pruebas realizadas el porcentaje de generalización presenta ligeros cambios a medida que se cambia varía el momento, es decir, podría decirse que la capacidad de generalización para el perceptrón multicapa no depende en gran manera del factor momento, sin embargo, el tiempo de convergencia si se ve afectado, para factores pequeños se obtienen tiempos grandes y para factores grandes tiempos pequeños, con lo que se deduce, que el factor momento acelera el aprendizaje, sin disminuir su calidad.

Basándose en estas pruebas se determinó que el factor 0.9 es el más apropiado para que la red perceptrón multicapa ofrezca un alto rendimiento, ya que, como puede observarse, es el que menor número de iteraciones presenta, en el menor tiempo de convergencia y el mayor porcentaje de generalización.

A continuación en la tabla 5.11 se presentan las pruebas realizadas para la red de neuronas de base radial, es importante mencionar que con estas pruebas se desea

obtener un factor que satisfaga tanto tiempo de convergencia como capacidad de generalización.

Tabla 5.11. *Influencia del factor momento en la red de base radial. (1/3).*

Momento	Nº Rostros	Nº iteraciones	Tiempo	% de generalización
0.0	5 x 8 = 40	324	0.166s	100.0%
		365	0.187s	100.0%
0.01	5 x 8 = 40	397	0.182s	100.0%
		397	0.188s	100.0%
0.02	5 x 8 = 40	380	0.176s	100.0%
		356	0.172s	100.0%
0.03	5 x 8 = 40	332	0.166s	99.71%
		401	0.187s	100.0%
0.04	5 x 8 = 40	329	0.156s	100.0%
		357	0.172s	100.0%
0.05	5 x 8 = 40	341	0.182s	100.0%
		373	0.188s	100.0%
0.06	5 x 8 = 40	331	0.172s	100.0%
		330	0.171s	100.0%
0.07	5 x 8 = 40	356	0.171s	100.0%
		371	0.187s	100.0%
0.08	5 x 8 = 40	306	0.156s	100.0%
		307	0.172s	100.0%
0.09	5 x 8 = 40	329	0.166s	100.0%
		277	0.156s	100.0%
0.1	5 x 8 = 40	350	0.172s	100.0%
		361	0.172s	100.0%
0.2	5 x 8 = 40	259	0.130s	100.0%
		244	0.125s	100.0%
0.3	5 x 8 = 40	249	0.150s	100.0%
		263	0.156s	100.0%

Fuente: Propia.

Tabla 5.11. Influencia del factor momento en la red de base radial. (2/3).

0.4	5 x 8 = 40	206	0.130s	100.0%
		209	0.110s	100.0%
0.5	5 x 8 = 40	181	0.104s	100.0%
		200	0.110s	100.0%
0.6	5 x 8 = 40	134	0.088s	100.0%
		126	0.078s	100.0%
0.7	5 x 8 = 40	99	0.078s	100.0%
		100	0.078s	100.0%
0.8	5 x 8 = 40	70	0.067s	100.0%
		73	0.062s	100.0%
0.9	5 x 8 = 40	21058	13.166s	23.06%
		21053	8.047s	25.88%
0.0	10 x 8 = 80	373	1.266s	99.00%
		360	1.109s	99.00%
0.01	10 x 8 = 80	393	1.219s	99.00%
		368	1.156s	99.00%
0.02	10 x 8 = 80	369	1.103s	99.00%
		374	1.109s	99.00%
0.03	10 x 8 = 80	352	1.261s	99.00%
		338	1.516s	99.00%
0.04	10 x 8 = 80	384	1.172s	99.00%
		387	1.203s	99.00%
0.05	10 x 8 = 80	358	1.266s	99.00%
		364	1.172s	99.00%
0.06	10 x 8 = 80	348	1.146s	99.00%
		361	1.203s	99.00%
0.07	10 x 8 = 80	367	1.244s	98.66%
		370	1.203s	99.00%
0.08	10 x 8 = 80	353	1.094s	99.00%
		367	1.188s	99.00%
0.09	10 x 8 = 80	343	1.088s	99.00%
		358	1.157s	99.00%

Fuente: Propia.

Tabla 5.11. Influencia del factor momento en la red de base radial. (3/3).

0.1	10 x 8 = 80	330	1.015s	99.00%
		316	0.969s	99.00%
0.2	10 x 8 = 80	289	0.958s	99.00%
		316	0.969s	99.00%
0.3	10 x 8 = 80	247	0.807s	99.00%
		232	0.782s	99.00%
0.4	10 x 8 = 80	223	0.724s	98.66%
		220	0.703s	99.00%
0.5	10 x 8 = 80	192	0.651s	98.66%
		200	0.656s	99.00%
0.6	10 x 8 = 80	152	0.812s	99.00%
		145	0.500s	99.00%
0.7	10 x 8 = 80	110	0.411s	99.00%
		109	0.390s	99.00%
0.8	10 x 8 = 80	74	0.396s	98.66%
		72	0.360s	99.00%
0.9	10 x 8 = 80	25472	73.026s	8.05%
		25459	71.672s	10.49%

Fuente: Propia.

Como se puede observar en la tabla 5.11, los tiempos de convergencia disminuyen notablemente a medida que se aumenta el factor de momento, demostrando nuevamente que el factor momento actúa como un acelerador del aprendizaje. También se puede notar que los porcentajes de generalización se mantienen casi estables a medida que cambian los factores de momento, el número de iteraciones disminuye notablemente al aumentar el factor y consigo disminuye el tiempo de convergencia. Los mejores resultados fueron obtenidos por los factores 0.7 y 0.8, en cuanto a números de iteraciones y tiempos de convergencia, sin embargo, se optó por emplear 0.7 como el factor momento predeterminado de la aplicación por presentar mejor promedio (99%) que el factor 0.8 (98.66%). Cabe destacar, que factores mayores o igual a 0.9 ocasionan un mal comportamiento en la red de

neuronas de base radial, como se puede observar en la tabla 5.11 en los casos donde se emplea el factor 0.9.

Prueba 6. Umbrales. Con esta prueba se desea determinar si la corrección de los umbrales mediante el algoritmo Backpropagation en la red perceptrón multicapa mejora su desempeño, para esto se realizan pruebas sin corregir los pesos y corrigiéndolos. En la tabla 5.12 se muestran los resultados sin la corrección de los umbrales y en la tabla 5.13 se muestran los resultados haciendo uso del algoritmo Backpropagation para su corrección.

Tabla 5.12. *Resultados sin corrección de umbrales.*

Nº Rostros	Nº iteraciones	Tiempo	% de generalización
10 x 8 = 80	10586	0.302s	98.00%
	9954	0.297s	98.00%
20 x 4 = 80	27386	1.041s	85.57%
	28160	1.047s	86.57%

Fuente: Propia.

Tabla 5.13. *Resultados con corrección de umbrales.*

Nº Rostros	Nº iteraciones	Tiempo	% de generalización
10 x 8 = 80	14464	0.379s	97.36%
	13840	0.375s	99.00%
20 x 4 = 80	32640	1.281s	86.11%
	32960	1.265s	88.04%

Fuente: Propia.

Como se puede observar en las tablas 5.12 y 5.13 la capacidad de generalización, el número de iteraciones y el tiempo de convergencia, son mayores cuando se realizan correcciones de los umbrales y menores cuando estos se mantienen constantes e iguales a 1. Basándose en estas pruebas se determinó que la corrección de los umbrales aumenta el rendimiento de la red.

Prueba 7. Pesos. En esta prueba se desea obtener el rango en el cual los pesos deben ser inicializados. Para esto se prueban diversos rangos para la red perceptrón multicapa y para la red de neuronas de base radial, como se muestra en las tablas 5.14 y 5.15 respectivamente.

Tabla 5.14. *Búsqueda del rango de pesos para la red perceptrón multicapa.*

Pesos	Nº Rostros	Nº iteraciones	Tiempo	% de generalización
[-0.005, 0.005]	5 x 8 = 40	7712	0.281s	97.49%
		7640	0.281s	98.00%
[-0.05, 0.05]	5 x 8 = 40	5572	0.184s	97.74%
		5440	0.219s	98.00%
[-0.5, 0.5]	5 x 8 = 40	3148	0.095s	98.00%
		2920	0.109s	98.00%
[-0.09, 0.09]	5 x 8 = 40	12416	0.345s	97.51%
		13680	0.344s	98.48%
[-0.9,0.9]	5 x 8 = 40	2520	0.078s	98.00%
		2560	0.094s	98.00%
[-0.05, 0.05]	10 x 4 = 40	9272	0.293s	89.63%
		8600	0.281s	91.00%
[-0.5,0.5]	10 x 4 = 40	4704	0.160s	90.85%
		4560	0.219s	93.00%
[-0.009,0.009]	10 x 4 = 40	11772	0.390s	88.49%
		11840	0.391s	90.58%
[-0.09,0.09]	10 x 4 = 40	8188	0.259s	90.12%
		8600	0.281s	92.00%
[-0.9,0.9]	10 x 4 = 40	3640	0.106s	90.36%
		4000	0.109s	92.53%

Fuente: Propia.

Como se puede observar en la tabla 5.14, los mejores resultados se obtienen en los rangos [-0.5, 0.5] y [-0.9, 0.9], sin embargo, el rango [-0.9, 0.9] presenta menores tiempos de convergencia y menor número de iteraciones. También se puede observar que a medida que los rangos disminuyen, el número de iteraciones aumenta, como en

el caso de [-0.009, 0.009] y a medida que los rangos son más grandes, el número de iteraciones disminuye.

Para la red perceptrón multicapa utilizar rangos muy grandes provocaría una inestabilidad, evitando el entrenamiento, por lo tanto, los rangos aceptables para la inicialización de los pesos debe estar entre [-1, 1] y como máximo entre [-2, 2]. Basándose en los resultados obtenidos se optó por establecer el rango [-0.5, 0.5], como el rango de inicialización predeterminado de la aplicación. Es importante mencionar que no existe un método determinista para la inicialización de los pesos, por lo cual, se realiza un conjunto de pruebas para la obtención de un rango apropiado para la inicialización de lo pesos.

Tabla 5.15. *Búsqueda del rango de pesos para la red de base radial. (1/2).*

Pesos	Nº Rostros	Nº iteraciones	Tiempo	% de generalización
[-0.005, 0.005]	5 x 8 = 40	594	0.244s	100.0%
		594	0.266s	100.0%
[-0.05, 0.05]	5 x 8 = 40	593	0.237s	100.0%
		595	0.219s	100.0%
[-0.5, 0.5]	5 x 8 = 40	623	0.246s	100.0%
		676	0.266s	100.0%
[-0.009, 0.009]	5 x 8 = 40	593	0.337s	100.0%
		594	0.250s	100.0%
[-0.09, 0.09]	5 x 8 = 40	588	0.234s	100.0%
		587	0.235s	100.0%
[-0.9, 0.9]	5 x 8 = 40	767	0.303s	100.0%
		883	0.344s	100.0%

Fuente: Propia.

Tabla 5.15. Búsqueda del rango de pesos para la red de base radial. (2/2).

[-0.005, 0.005]	10 x 4 = 40	1548	1.275s	98.00%
		1548	1.312s	98.00%
[-0.05, 0.05]	10 x 4 = 40	1547	1.187s	98.00%
		1553	1.281s	98.00%
[-0.5, 0.5]	10 x 4 = 40	1587	1.384s	98.40%
		1642	1.344s	99.00%
[-0.009, 0.009]	10 x 4 = 40	1547	1.240s	98.00%
		1548	1.266s	98.00%
[-0.09, 0.09]	10 x 4 = 40	1545	1.168s	98.00%
		1542	1.250s	98.00%
[-0.9, 0.9]	10 x 4 = 40	1675	1.171s	98.40%
		1737	1.219s	99.00%

Fuente: Propia.

Como se puede observar en la tabla 5.15, los mejores resultados, se obtienen en el rangos [-0.5, 0.5]. Cabe destacar que la red de neuronas de base radial no presenta problemas cuando se inicializan sus pesos en rangos grandes, como sucede con la red perceptrón multicapa. También se puede notar que en la mayoría de los casos, a medida que el rango de inicialización aumenta, el número de iteraciones también aumenta. En base a estos resultados, se decidió emplear un rango entre [-0.5, 0.5] como el predeterminado para la aplicación.

Prueba 8. Heurísticas. En esta prueba se desea determinar que heurística es mejor para la determinación de los centros y amplitudes de la red de neuronas de base radial. Los resultados se muestran en la tabla 5.16.

Tabla 5.16. *Búsqueda de la mejor heurística para la red de base radial.*

Heurísticas	Nº Rostros	Nº iteraciones	Tiempo	% de generalización
Media Uniforme	5 x 8 = 40	645	0.262s	100%
		626	0.266s	100%
Media Geométrica	5 x 8 = 40	101	0.084s	100%
		107	0.093s	100%

Fuente: Propia.

Como se puede ver en la tabla 5.16, se muestran los resultados de aplicar dos heurísticas, media uniforme y media geométrica. Mediante el uso de la media uniforme la red de base radial alcanzó al igual que con la media geométrica una capacidad de generalización máxima, sin embargo, lo realizó en un tiempo de convergencia mayor al empleado por la heurística de media geométrica, razón por la cual, se optó por emplear la heurística con menor tiempo de convergencia, para compensar la red en cuanto a tiempo y calidad.

Prueba 9. Número de autovectores. En esta prueba se desea mostrar que el número de autovectores utilizados en el análisis de componentes principales para obtener los mejores resultados, durante el proceso de aprendizaje, no debe ser necesariamente el 100% de los autovectores. Cabe destacar que el número de autovectores utilizados para las pruebas anteriores fue de un 33.33% del número total, y cuando este resultaba en un número de componentes menor a 15 se empleó el número total de autovectores menos uno, a continuación se presentan diversas pruebas variando el número de autovectores, desde un 100% de estos hasta un 10%, como se muestra en la tabla 5.17.

Tabla 5.17. *Variación del número de autovectores para el perceptrón multicapa. (1/2)*

% autovectores	Nº Rostros	Nº iteraciones	Tiempo	% generalización
10%	20 x 3 = 60	113628	2.955s	73.89%
		91500	2.454s	76.80%
20%	20 x 3 = 60	37800	0.978s	78.96%
		39300	1.031s	80.27%
30%	20 x 3 = 60	23988	0.681s	79.82%
		23040	0.656s	82.87%
40%	20 x 3 = 60	15396	0.512s	77.84%
		14880	0.484s	81.06%
50%	20 x 3 = 60	14004	0.512s	77.93%
		13740	0.500s	80.85%

Fuente: Propia.

Tabla 5.17. Variación del número de autovectores para el perceptrón multicapa. (2/2).

60%	20 x 3 = 60	13020	0.519s	77.35%
		11820	0.516s	80.02%
70%	20 x 3 = 60	12324	0.524s	77.74%
		13020	0.547s	79.78%
80%	20 x 3 = 60	12108	0.559s	77.64%
		11520	0.531s	81.52%
90%	20 x 3 = 60	12540	0.615s	77.68%
		12360	0.609s	79.54%
100%	20 x 3 = 60	12012	0.628s	78.42%
		11580	0.609s	82.53%
10%	30 x 3 = 90	156798	7.084s	69.95%
		179190	7.922s	72.45%
20%	30 x 3 = 90	75906	3.690s	74.92%
		84870	4.094s	78.60%
30%	30 x 3 = 90	50022	2.615s	75.99%
		49500	2.625s	79.34%
40%	30 x 3 = 90	38268	2.272s	75.79%
		37800	2.250s	79.38%
50%	30 x 3 = 90	34578	2.303s	75.10%
		34110	2.312s	78.51%
60%	30 x 3 = 90	31500	2.290s	75.64%
		34020	2.344s	78.35%
70%	30 x 3 = 90	31500	2.646s	76.06%
		30690	2.438s	79.00%
80%	30 x 3 = 90	31014	2.280s	75.82%
		32400	2.359s	78.19%
90%	30 x 3 = 90	32220	2.506s	75.82%
		31320	2.500s	78.33%
100%	30 x 3 = 90	32220	2.959s	75.58%
		31590	2.953s	78.18%

Fuente: Propia.

En la tabla 5.17 se muestran una serie de pruebas utilizando 60 y 90 rostros, en ambas pruebas el número de autovectores no afecto significativamente en el porcentaje de generalización.

Los resultados más bajos en cuanto a capacidad de generalización se obtuvieron al emplear 10% y 20% de los autovectores, notándose también un aumento en los tiempos de convergencia, por otro lado, los mejores resultados para 60 rostros se presentaron al utilizar 30% y 100% de los autovectores, y para 90 rostros, 40% y 70%.

Tabla 5.18. *Variación del número de autovectores para la red de base radial. (1/2)*

% autovectores	Nº Rostros	Nº iteraciones	Tiempo	% generalización
10%	20 x 3 = 60	279	0.853s	62.09%
		278	0.859s	63.73%
20%	20 x 3 = 60	254	0.781s	65.30%
		269	0.813s	66.74%
30%	20 x 3 = 60	286	0.906s	66.19%
		288	0.906s	66.73%
40%	20 x 3 = 60	257	0.821s	69.05%
		269	0.859s	70.25%
50%	20 x 3 = 60	247	0.800s	70.60%
		239	0.781s	72.00%
60%	20 x 3 = 60	245	0.812s	73.05%
		258	0.844s	74.50%
70%	20 x 3 = 60	252	0.840s	73.85%
		240	0.797s	75.00%
80%	20 x 3 = 60	266	0.899s	74.40%
		278	0.953s	75.50%
90%	20 x 3 = 60	269	0.924s	75.20%
		257	0.875s	76.50%
100%	20 x 3 = 60	258	0.918s	77.20%
		246	0.859s	79.50%
10%	30 x 3 = 90	309	3.546s	65.42%
		313	3.563s	66.16%
20%	30 x 3 = 90	331	4.106s	68.05%
		339	4.297s	68.49%
30%	30 x 3 = 90	332	3.862s	69.89%
		327	3.859s	71.00%
40%	30 x 3 = 90	321	3.847s	70.53%
		328	3.969s	71.00%

Fuente: Propia.

Tabla 5.18. Variación del número de autovectores para la red de base radial. (2/2)

50%	30 x 3 = 90	320	3.796s	72.19%
		314	3.703s	72.66%
60%	30 x 3 = 90	324	4.093s	72.93%
		311	3.985s	74.00%
70%	30 x 3 = 90	328	4.025s	73.19%
		327	4.328s	73.66%
80%	30 x 3 = 90	341	3.902s	73.92%
		356	3.921s	74.33%
90%	30 x 3 = 90	357	3.952s	74.06%
		362	3.968s	74.66%
100%	30 x 3 = 90	362	4.534s	74.06%
		365	4.563s	74.33%

Fuente: Propia.

Como se puede observar en la tabla 5.18, la red de neuronas de base radial presenta resultados cada vez mejores, a medida que se aumenta el número de autovectores, manteniendo los tiempos de convergencia uniformes durante las variaciones de los autovectores, por lo tanto, los resultados más bajos se encuentran al realizar las pruebas con 10%, 20% y así sucesivamente, lográndose notar a ciertos porcentajes que el aumento de la capacidad de generalización no es significativa, por lo tanto, se podría hacer uso de un menor número de autovectores sin afectar la capacidad de generalización de esta red.

Mediante estas pruebas se demuestra que el análisis de componentes principales permite una reducción de la dimensión real del problema a dimensiones mucho más pequeñas, sin afectar significativamente el porcentaje de generalización.

Una vez finalizadas las pruebas para determinar los valores de los parámetros que aumentan el desempeño y mejoran el funcionamiento de las redes neuronales para el reconocimiento facial, se procede a realizar comparaciones entre la red perceptrón multicapa y la red de base radial. Los valores de los parámetros que mejores resultados presentaron a lo largo de las pruebas se muestran en la tabla 5.19, los cuales serán empleados para pruebas finales.

Tabla 5.19. *Parámetros finales de las redes neuronales artificiales.*

Parámetros	Red perceptrón multicapa	Red de base radial
Nº de neuronas	20	Nº de Rostros
Función de activación	Sigmoidal	Gausiana
Factor de aprendizaje	0.1	0.01
Error mínimo	0.1	0.01
Factor momento	0.9	0.7
Pesos	[-0.5,0.5]	[-0.5,0.5]
Heurísticas	No aplica	Media Geométrica

Fuente: Propia.

Prueba 10. Comparación de las redes neuronales. En esta prueba se comparan los resultados obtenidos por la red perceptrón multicapa y de base radial, para determinar que red se comporta de mejor manera. En las tablas 5.20 y 5.21 se muestran los resultados del perceptrón multicapa y de la red de base radial respectivamente, haciendo uso de los parámetros descritos en la tabla 5.19, y un 33% del total de los autovectores y para casos donde el 33% de autovectores es menor a 15, se empleó el 100%.

Tabla 5.20. *Resultados de la red perceptrón multicapa. (1/2).*

Nº Rostros	Nº iteraciones	Tiempo	% de generalización
10 x 3 = 30	4416	0.128s	82.90%
	3990	0.125s	84.00%
10 x 8 = 80	7424	0.215s	97.99%
	7280	0.218s	99.42%
20 x 3 = 60	20100	0.743s	79.47%
	20340	1.093s	81.35%
20 x 8 = 160	46272	2.356s	98.06%
	48480	2.563s	98.50%
30 x 3 = 90	45920	2.199s	97.80%
	48000	2.234s	98.28%

Fuente: Propia.

Tabla 5.20. Resultados de la red perceptrón multicapa. (2/2).

30 x 8 = 240	143591	11.827s	98.10%
	127626	9.672s	98.33%
40 x 3 = 120	62568	14.869s	71.53%
	66720	15.985s	73.27%
40 x 8 = 320	181440	67.481s	97.40%
	214080	77.593s	97.75%

Fuente: Propia.

Como se puede observar en la tabla 5.20, la red perceptrón multicapa presenta muy buenos resultados cuando se le presenta 8 caras por sujetos, en promedio, se encuentra entre 97.40% y 99.42%, por otro lado, cuando se le presentan tres caras por sujetos los resultados disminuyen un poco ubicándose entre 71.53% y 97.80%, lo cual, podría considerarse bastante bueno debido al bajo número de rostros que forman parte del conjunto de entrenamiento.

Tabla 5.21. *Resultados de la red de base radial.*

Nº Rostros	Nº iteraciones	Tiempo	% de generalización
10 x 3 = 30	171	0.093s	93.00%
	165	0.093s	94.00%
10 x 8 = 80	115	0.421s	99.00%
	117	0.422s	99.00%
20 x 3 = 60	266	0.840s	70.00%
	261	0.828s	70.00%
20 x 8 = 160	208	5.881s	98.00%
	205	5.828s	98.00%
30 x 3 = 90	325	3.578s	70.19%
	311	3.453s	71.00%
30 x 8 = 240	311	32.837s	97.99%
	309	32.469s	98.33%
40 x 3 = 120	415	11.196s	68.45%
	437	11.547s	68.87%
40 x 8 = 320	631	155.343s	98.00%
	629	153.922s	98.00%

Fuente: Propia.

Como se puede apreciar en la tabla 5.21, la red de neuronas de base radial presenta muy buenos resultados cuando se le presentan 8 caras por sujetos, alcanzando un rango de generalización entre 97.99% y 99.00%, en cambio, cuando se le presenta 3 caras por sujetos, se obtienen resultados entre 68.45% y 93.00%. Cabe destacar que estos resultados pueden mejorarse haciendo uso de un error (condición de parada del algoritmo) mucho más pequeño, con la única desventaja de aumentar los tiempos de convergencia, y como se puedo notar en la tabla 5.21, los tiempos son bastante aceptables.

A continuación en la tabla 5.22, se realiza una comparación entre los porcentajes de generalización de la tabla 5.20 y 5.21:

Tabla 5.22. *Comparación entre perceptrón multicapa y función de base radial.*

Nº Rostros	% de generalización	
	Perceptrón Multicapa	Función de Base Radial
10 x 3 = 30	82.90%	93.00%
	84.00%	94.00%
10 x 8 = 80	97.99%	99.00%
	99.42%	99.00%
20 x 3 = 60	79.47%	70.00%
	81.35%	70.00%
20 x 8 = 160	98.06%	98.00%
	98.50%	98.00%
30 x 3 = 90	97.80%	70.19%
	98.28%	71.00%
30 x 8 = 240	98.10%	97.99%
	98.33%	98.33%
40 x 3 = 120	71.53%	68.45%
	73.27%	68.87%
40 x 8 = 320	97.40%	98.00%
	97.75%	98.00%

Fuente: Propia.

En la tabla 5.22, se puede apreciar que ambas redes tienen buena capacidad de generalización, aunque cada red presenta un comportamiento particular para cada caso de prueba.

En la tabla 5.23 y 5.24 se presentan los resultados obtenidos al hacer uso de un 90% de los autovectores.

Tabla 5.23. *Resultados de la red perceptrón multicapa con 90% de autovectores.*

Nº Rostros	Nº iteraciones	Tiempo	% de generalización
10 x 3 = 30	4240	0.120s	82.66%
	4200	0.125s	85.00%
10 x 8 = 80	7360	0.348s	97.33%
	7280	0.344s	98.00%
20 x 3 = 60	12620	0.661s	78.18%
	12720	0.657s	79.50%
20 x 8 = 160	42400	4.125s	97.85%
	48160	4.500s	98.50%
30 x 3 = 90	31020	2.661s	75.14%
	33930	2.844s	76.23%
30 x 8 = 240	134160	19.692s	98.33%
	123840	18.078s	99.00%
40 x 3 = 120	68680	7.849s	72.79%
	65280	7.610s	74.62%
40 x 8 = 320	337386	64.430s	97.25%
	378240	71.594s	98.25%

Fuente: Propia.

Como se puede observar en la tabla 5.23, la diferencia de los resultados obtenidos con respecto a la tabla 5.22 no es muy significativa, a pesar de emplear un 90% de los autovectores.

Tabla 5.24. *Resultados de la red de base radial con 90% de autovectores.*

Nº Rostros	Nº iteraciones	Tiempo	% de generalización
10 x 3 = 30	167	0.078s	93.66%
	158	0.078s	95.00%
10 x 8 = 80	132	0.635s	99.00%
	132	0.642s	99.00%
20 x 3 = 60	263	0.979s	75.16%
	281	1.031s	77.50%
20 x 8 = 160	231	8.624s	98.00%
	231	8.593s	98.00%
30 x 3 = 90	354	4.437s	73.66%
	359	4.350s	74.00%
30 x 8 = 240	476	57.265s	97.55%
	462	55.250s	97.66%
40 x 3 = 120	666	19.682s	71.41%
	679	20.000s	72.25%
40 x 8 = 320	1225	342.000s	97.50%
	1156	314.859s	97.50%

Fuente: Propia.

Como se puede apreciar en la tabla 5.24, la red de neuronas de base radial, presenta para la mayoría de los casos, mejores resultados que los presentados en la tabla 5.22.

Tanto la red perceptrón multicapa como la red de función de base radial presentan buenos resultados cuando se enfrentan a problemas de pocos sujetos y pocas caras, por el contrario, cuando se enfrenta a problemas con muchos sujetos y pocas caras, su capacidad de generalización se ve afectada levemente, sin embargo, la calidad de los resultados aumenta cuando se le presentan un mayor número de caras por sujetos.

Entre estas dos redes estudiadas, determinar cual es la mejor o cual presenta mejor rendimiento, ante cualquier problema, sería incierto, debido a que ambas redes se desempeñan de forma diferente ante un mismo problema, por lo tanto, podría decirse que la red perceptrón multicapa es mejor para algunos problemas y la red de neuronas de base radial para otros problemas, sin embargo, se podría decir que ambas redes se complementan y que podrían alcanzar altos porcentajes de aciertos trabajando juntas para la resolución de un mismo problema.

A continuación presentamos los resultados obtenidos para el caso 3 x 2 = 6, una vez calculado el porcentaje de acierto provisto por ambas redes, para esto, se le presentó a cada red, una vez entrenada, todos los rostros a reconocer.

En la tabla 5.25 se presentan los resultados retornados por cada red al presentarle los 30 rostros correspondientes al conjunto de pruebas (3 sujetos, 10 caras por sujeto).

Tabla 5.25. *Porcentajes de aciertos de las redes neuronales. (1/2).*

Sujeto - Cara	**% de aciertos**		
	Perceptrón Multicapa	Función de Base Radial	Ambas Redes Neuronales
1 – 1	1=100%	1=100%	1=100%
1 – 2	1=100%	1=100%	1=100%
1 – 3	1=100%	1=60%, 2=40%	1=80%,2=20%
1 – 4	1=100%	1=100%	1=100%
1 – 5	1=100%	1=100%	1=100%
1 – 6	1=100%	1=100%	1=100%
1 – 7	1=100%	1=100%	1=100%
1 – 8	1=100%	1=100%	1=100%
1 – 9	1=100%	1=100%	1=100%
1 – 10	1=100%	1=100%	1=100%
2 – 1	2=100%	2=100%	2=100%

Fuente: Propia.

***Tabla 5.25**. Porcentajes de aciertos de las redes neuronales. (2/2).*

2 – 2	2=100%	2=100%	2=100%
2 – 3	2=100%	2=100%	2=100%
2 – 4	2=100%	2=100%	2=100%
2 – 5	2=100%	2=100%	2=100%
2 – 6	2=100%	2=100%	2=100%
2 – 7	2=100%	2=100%	2=100%
2 – 8	2=100%	2=100%	2=100%
2 – 9	2=100%	2=52%, 1=48%	2=76%,1=24%
2 – 10	2=100%	2=100%	2=100%
3 – 1	3=100%	3=100%	3=100%
3 – 2	3=100%	3=100%	3=100%
3 – 3	3=100%	3=100%	3=100%
3 – 4	1=51%, 3=49%	1=53%, 3=47%	1=52%, 3=48%
3 – 5	1=100%	1=100%	1=100%
3 – 6	3=54%,1=46%	3=100%	3=77%,1=23%
3 – 7	3=54%,1=46%	3=100%	3=77%,1=23%
3 – 8	3=52%,1=48%	3=100%	3=76%,1=24%
3 – 9	3=59%,1=41%	3=100%	3=80%,1=20%
3 – 10	3=59%,1=41%	3=100%	3=80%,1=20%
Nº Aciertos:	28 Aciertos = 93.3%	28 Aciertos = 93.3%	28 Aciertos=93.3%

Fuente: Propia.

El porcentaje de generalización obtenido por las redes perceptrón multicapa y de base radial para la elaboración de la tabla 5.21 fueron de 87.56% y 91.96% respectivamente. Como se puede observar los porcentajes de acierto de cada red son más altos que los de generalización, y puede darse el caso donde el porcentaje de acierto entre ambas redes sea aun mayor.

CONCLUSIONES

1. Las redes neuronales artificiales perceptrón multicapa y función de base radial representan una excelente opción para el reconocimiento facial, complementándose entre sí y abarcando gran parte del funcionamiento de las redes neuronales.

2. El número de neuronas empleadas para la capa oculta de la red perceptrón multicapa no afecta de manera radical su funcionamiento, en cambio, la red de neuronas de base radial presenta los mejores resultados cuando su número de neuronas es igual al número de rostros a entrenar, es decir, el número de neuronas debe estar acorde con la dimensión del problema que se le presente a la red.

3. Las funciones de activación Potencia, Sigmoidal, Tangente Hiperbólica y Seno, presentan un comportamiento bastante uniforme en cuanto a capacidad de generalización y pequeñas diferencias en lo que a tiempos de convergencias se refiere.

4. El factor de aprendizaje determina la magnitud de las correcciones de los pesos, estableciendo la velocidad de aprendizaje de la red. Sobre la base de los experimentos realizados en este trabajo, este factor debe ubicarse entre 0.01 y 0.3 para evitar oscilaciones sobre la superficie del error.

5. El factor momento acelera el proceso de aprendizaje, manteniendo estable la capacidad de generalización y evitando la saturación de las funciones de activación.

6. Debido a las características particulares del perceptrón multicapa y la red de neuronas de base radial, los valores de los parámetros iniciales como los pesos, umbrales, factores de aprendizaje y momento, también deben ser particulares para cada red, para alcanzar buenos resultados.

7. Cuando en la red función de base radial las amplitudes se obtienen mediante la media geométrica, los centros se activan de mejor manera, aportando beneficios al entrenamiento de la red.

8. La red de neuronas de base radial es mucho más sensible, al número de autovectores empleados para la obtención de las entradas reducidas, que el perceptrón multicapa, cumpliéndose que a mayor número de autovectores, mayor es la capacidad de generalización y a menor número de autovectores, menor es la capacidad de generalización, mientras que el perceptrón multicapa, con pocos autovectores alcanza niveles de generalización muy buenos.

9. La red perceptrón multicapa alcanza tiempos de convergencia mucho más pequeños que la red de función de base radial, y capacidades de generalización bastantes similares.

10. El perceptrón multicapa y la red de base radial se pueden emplear conjuntamente para aumentar el porcentaje de acierto y garantizar altos niveles de reconocimiento.

11. Esta aplicación además de reconocer rostros, es capaz de reconocer gran variedad de patrones, ya sean, números, letras, figuras, huellas dactilares, etc.

RECOMENDACIONES

1. Implementar una red neuronal producto de la fusión del perceptrón multicapa y la red de neuronas de base radial, la cual, sea multicapa y disponga de una fase no supervisada, para el cálculo de los centros y amplitudes, y para la corrección de los pesos emplear el algoritmo de Backpropagation.

2. Realizar pruebas utilizando funciones de activación no exploradas en esta investigación, en busca de mejores resultados en cuanto a generalización del problema.

3. Disminuir el tamaño del error cuadrático, a valores próximos a cero, con la finalidad de obtener mejores resultados.

4. Emplear algún filtro para obtener a partir de un rostro, un conjunto de rostros con diferentes tamaños y orientaciones (filtro de Gabor).

5. Utilizar imágenes a color para comparar los resultados obtenidos con las imágenes a escala de grises.

BIBLIOGRAFÍA

[1] De Abajo, N. Gómez, A. (2001). **Introducción a la inteligencia artificial: Sistemas expertos, redes neuronales artificiales y computación evolutiva,** Universidad de Oviedo.

[2] Grau, M. Noguera, M. (2001). **Cálculo numérico: Teoría y Práctica**, Edicions UPC.

[3] Pérez, M. Caparrini, F. (2003). **Máquinas moleculares basadas en ADN**, Universidad de Sevilla.

[4] Serrano, R. (2003). **Introducción al análisis de datos experimentales: Tratamiento de datos en Bioensayos**, Universitat Jaume I.

[5] Viñuela, P. Galván, I. (2004). **Redes de Neuronas Artificiales, Un Enfoque Práctico**, España, Pearson Prentice Hall.

[6] Gordon, A. (2005). **Fundamentos de inversiones. Teoría y práctica. 3b0 edición**, Pearson Educación.

[7] Guisande, C. (2006). **Tratamiento de datos,** Ediciones Díaz de Santos.

[8] Hernández, L. (2006). **Predicción y optimización de emisores y consumo mediante redes neuronales en motores diesel**, Reverte.

[9] Vásquez, M. Ramfrez, G. (2006). **Aspectos Teóricos del Algebra Matricial con Aplicaciones Estadísticas**, CDCH UCV.