UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

"CLASIFICACIÓN DE HUEVOS POR TAMAÑO USANDO REDES NEURONALES "

TRABAJO FIN DE GRADO

Julio -2020

AUTOR: Teresa Pomares Palomares

DIRECTOR/ES: Luis Payá Castello

RESUMEN

El objetivo es desarrollar una GUI en Matlab que clasifique los huevos por tamaño utilizando diferentes tipos de redes neuronales. Por lo general, los huevos se clasifican en grupos de calidad según su peso, pero intentamos realizar una evaluación de peso indirecta, en función del tamaño de los huevos. La GUI debe permitir al usuario construir diferentes tipos de redes neuronales, usarlas para la clasificación de los huevos y evaluar la precisión de la clasificación en cada caso.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

1 INTRODUCCIÓN

Actualmente, los robots clasificadores de huevos de la industria tienen un proceso lento y poco eficiente de separación de huevos. Este proceso se utiliza para facilitar su posterior venta al público y se basa en la separación de los huevos por cuatro masas base: S, M, L, XL. Dependiendo del tamaño, el precio de los huevos variará y, por tanto, una correcta separación es clave para la venta del producto. Por otro lado, el mismo proceso en sí es mejorable y optimizar el tiempo y la calidad del proceso puede ahorrar dinero a la empresa.

Hemos basado nuestro proyecto en diversos trabajos anteriores que recopilaban los datos de 120 muestras de huevos, con sus respectivas cualidades.

Basándonos en proyectos anteriores, [1], el método indirecto para la clasificación de los huevos mediante el procesamiento de imágenes incluye los siguientes pasos principales:

- Adquisición de imágenes de huevos utilizando el sistema de visión artificial.
- Análisis de la imagen usando procedimientos apropiados para el cálculo de los parámetros geométricos de cada huevo.
- Métodos estadísticos para determinar la relación entre el peso de los huevos y sus parámetros geométricos.
- Determinar el modelo matemático del peso del huevo utilizando uno de los parámetros geométricos y probando la precisión del enfoque indirecto propuesto para la clasificación del huevo.

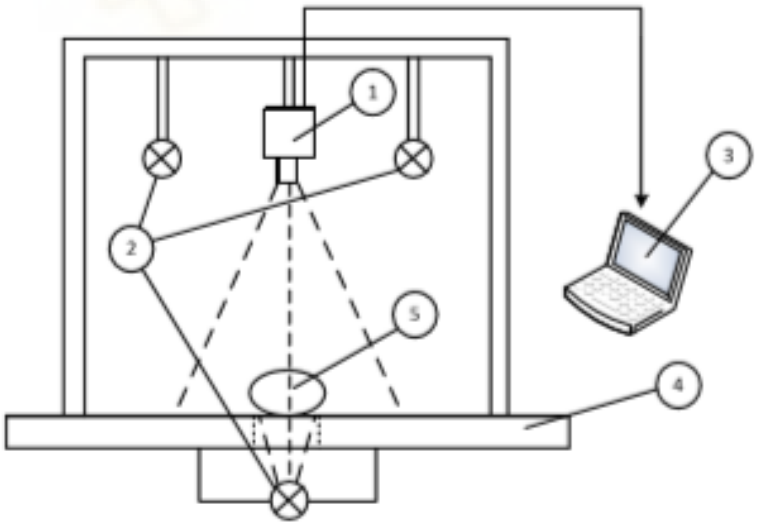

Figura 1: Mecanismo de visión por computador para la medición de los huevos, extraída de [1].

La figura 1 incluye:

1. NI 1772C Smart Camera.
2. Fuentes de luz.

3. Ordenador.
4. Superficie de trabajo.
5. Huevo a estudiar.

El sistema mostrado en la figura 1 proporciona iluminación al objeto en dos direcciones diferentes. Primeramente, se enfoca sobre el objeto para analizar las características del color, y seguidamente se enfoca desde la parte más baja para un mejor estudio de la profundidad y para radiografiar el objeto. La iluminación utilizada es de tipo LED, con luz blanca y una temperatura de 3000K [1].

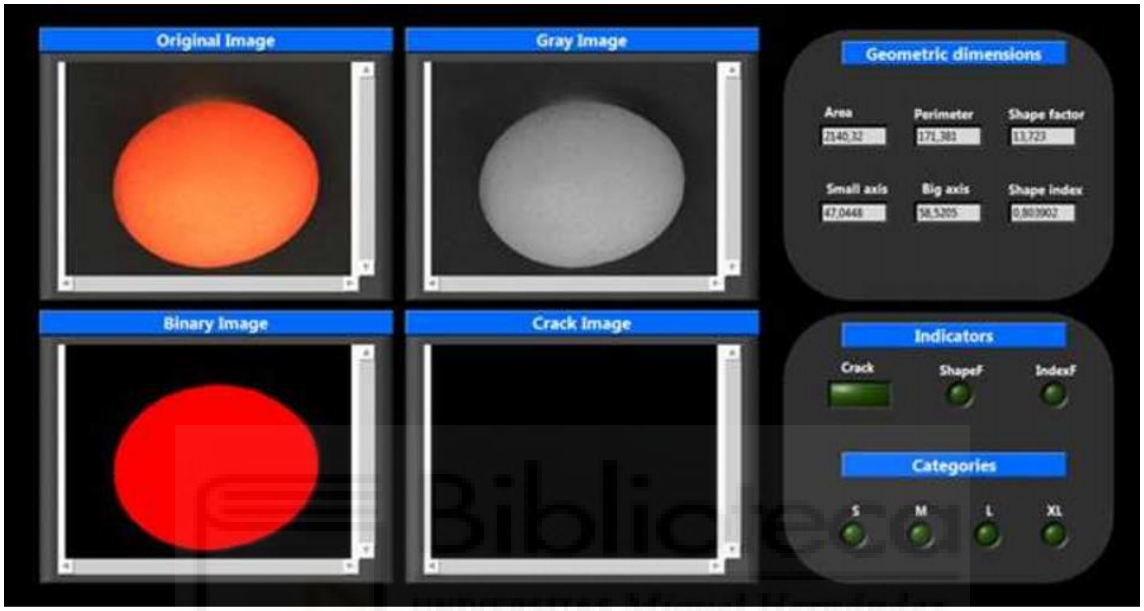

Figura 2: Imagen frontal de un huevo, extraída de [1].

La figura 2 muestra el panel frontal con el que se ha trabajado en proyectos anteriores para obtener las imágenes de los huevos. Mediante un riguroso proceso de selección de variables y ecuaciones, se estudió cada huevo y cada imagen para finalmente poder hacer la clasificación. Los cuatro parámetros esenciales para el análisis de los huevos son: perímetro, área, eje mayor y eje menor.

En trabajos anteriores se ha estudiado que a partir de los parámetros geométricos de los huevos, medidos con visión artificial, se estima la mejor correlación entre la masa y el área, con un coeficiente de correlación igual a 0.9891. Los resultados obtenidos del modelado de masas indican que el error absoluto máximo Δm entre la masa de huevo medida y la masa de huevo, calculada con el modelo para la muestra de prueba, es superior al 5% para la clase S, mientras que es menor para todas las demás clases. El error de clasificación común para la muestra de prueba, estimado indirectamente con el modelo matemático recibido usando la visión por computadora, es 2.5%, mientras que el error común para la muestra de entrenamiento es mayor: 12.5%. El análisis de los resultados obtenidos muestra que el enfoque para la estimación indirecta de la masa de los huevos, usando la visión por computadora, puede usarse para clasificar los huevos por peso en grupos cualitativos, de acuerdo con el estándar especificado [1].

Clase	Tamaño	Masa
XL	Muy grande	73 g
L	Grande	63 g a 73 g
M	Medio	53 g a 63 g
S	Pequeño	Menos de 53 g

Tabla 1: Clasificación de los huevos por masas.

Como se ha citado anteriormente, por lo general, los huevos se clasifican en grupos de calidad según su masa. Como se muestra en la tabla 1, dependiendo de la masa del huevo, se distinguen cuatro grupos de calidad: S, M, L y XL.

Las máquinas de clasificación existentes primero miden el peso de cada huevo en un flujo, luego lo separan en el grupo apropiado. Este es un proceso relativamente lento. El propósito de la investigación es aumentar la productividad del proceso de clasificación. Por esa razón, se propone, en lugar de la medición del peso, medir algunas características geométricas básicas de los huevos y encontrar la relación de estas características con el peso. Esta es la llamada medición indirecta del peso. Las características geométricas se obtienen mediante el sistema de visión por computadora, como se muestra en la figura 1, ya que es mucho más rápido que la medición de peso habitual. Después de eso, para encontrar la relación entre las características geométricas y el peso, se deben usar algunos métodos matemáticos. Uno de estos métodos podría ser redes neuronales.

Hay dos formas de usar redes neuronales para resolver esta tarea en particular. La primera es realizar la llamada “tarea de clasificación”. La red se crea con cuatro neuronas en la capa de entrada, correspondiente a P, S, D y d, donde P es el perímetro del huevo; S el área del huevo; d el eje menor y D el eje mayor, entrenadas con características de huevos de cada uno de los cuatro grupos de calidad (muestra de entrenamiento) y debe proporcionar una respuesta definiendo en ella el grupo al cual pertenece. El peso de los huevos no es una entrada de la red puesto que este no es el objetivo de la investigación: el peso se utiliza en este caso solo para determinar a qué grupo pertenece cada huevo del conjunto de datos. Por tanto, solo utilizamos el peso para determinar a qué grupo pertenece cada huevo. La segunda forma es realizar la llamada “tarea de regresión”. En este caso se crea una red con cuatro entradas (P, S, D y d), la entrada y la red debe dar como respuesta el peso previsto de cada huevo. En este caso nuevamente, no usamos el peso como entrada sino como vector objetivo (salida deseada) durante el entrenamiento de la red.

En este proyecto se ha diseñado una interfaz utilizando Matlab, que es capaz de facilitarle al usuario la creación de una red neuronal personalizada que se ajusta a las necesidades concretas del estudio a realizar y que es capaz de adaptarse a distintas situaciones puesto que es el propio usuario quien define la red desde las bases. Para ello, primeramente, el usuario diseña una red a partir de una matriz de conexión, y define los parámetros a utilizar. Seguidamente se entrena dicha red utilizando los datos introducidos previamente por el usuario, y finalmente la red está operativa para su funcionamiento.

2 MATERIALES Y MÉTODOS

Para el desarrollo de este proyecto hemos usado la red neuronal Toolbox de MATLAB R2018a, a pesar de no ser una herramienta imprescindible para el desarrollo y la implementación de las redes neuronales en MATLAB, sí acelera el proceso y nos ha servido eficientemente para obtener los resultados buscados.

2.1 INTRODUCCIÓN A REDES NEURONALES

Para comprender bien el proceso de este proyecto, es de vital importancia explicar previamente qué es una red neuronal y cómo funciona, pues es la herramienta principal que nos acontece. Una red neuronal se puede definir como el modelo simplificado de un cerebro humano, pues es una copia del modo en que este procesa la información y simultanea un número elevado de unidades de procesamiento interconectadas las cuales podríamos definir como las pseudo neuronas de una pseudo red.

Fue en 1943 cuando se propuso el primer modelo de red neuronal en términos de modelo computacional de la actividad nerviosa. Dicho modelo era binario y cada neurona tenía un umbral previamente fijado. Este modelo sirvió como base para el desarrollo de las redes, pero no ha sido hasta la actualidad cuando las redes neuronales han logrado una madurez muy aceptable y su uso se ha normalizado en todo tipo de aplicaciones, desde el reconocimiento de voz o imagen, hasta aplicaciones para la medicina moderna.

2.1.1 REDES NEURONALES BIOLÓGICAS

El cerebro humano es un órgano altamente complejo, formado por millones de neuronas que se interconectan entre sí por una sinapsis. Cada neurona es capaz de desarrollar sus propios impulsos eléctricos, los cuales son transmitidos por los axones y conectan con otras neuronas a través de sus dendritas, formando una red compleja y amplia de conexiones llamada sinapsis, que es quien se encarga de regular el impulso eléctrico anteriormente citado. Podríamos decir que la información en el cerebro se transmite mediante impulsos eléctricos, que viajan por las neuronas biológicas y por señales químicas que comunican las neuronas con otras neuronas en sus bordes usando unas estructuras llamadas sinapsis.

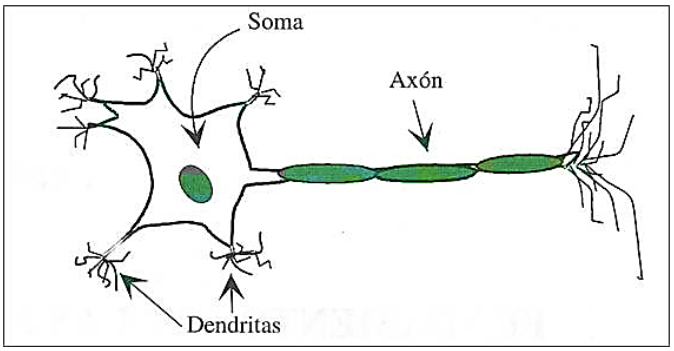

Figura 3: Modelo de neurona biológica, extraído de [2].

En la figura 3 se muestra la estructura básica de una neurona biológica. El cerebro humano está formado por millones de neuronas que se conectan entre sí formando una amplia red neuronal biológica.

2.1.2 REDES NEURONALES ARTIFICIALES

La estructura de una red neuronal artificial está inspirada en el funcionamiento de una red neuronal biológica, es decir, en un cerebro. Una red neuronal artificial está formada por un conjunto de nodos conocidos como neuronas artificiales que están conectadas y transmiten señales entre sí y dichas señales se transmiten desde la entrada hasta la salida generada.

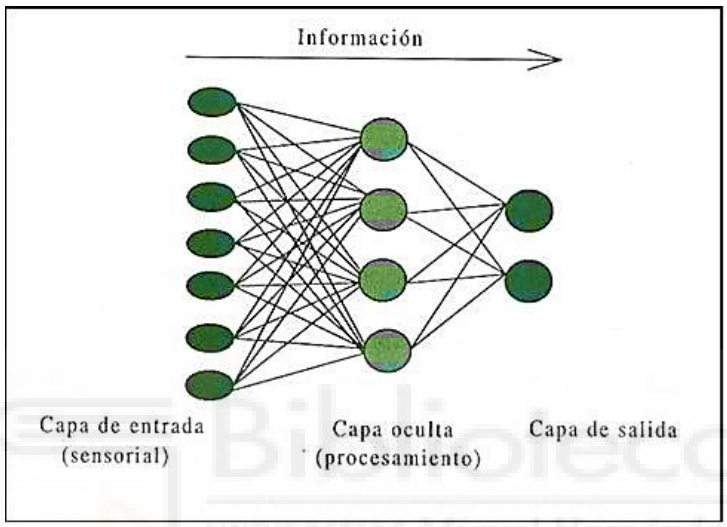

Figura 4: Modelo de red neuronal artificial de conexiones hacia adelante, extraído de [2].

Como se observa en la figura 4, una red neuronal artificial está formada por tres partes principales: la capa de entrada, la capa oculta y la capa de salida.

El objetivo principal de este modelo es aprender modificándose automáticamente a sí mismo de forma que puede llegar a realizar tareas complejas que no podrían ser realizadas mediante la clásica programación basada en reglas. De esta forma se pueden automatizar funciones que en un principio solo podrían ser realizadas por personas.

En este proyecto, el uso de redes neuronales es necesario para crear un modelo eficiente y automatizado que sea capaz de clasificar los huevos de la industria alimenticia de forma rápida y precisa, ahorrando así tiempo y dinero al sector.

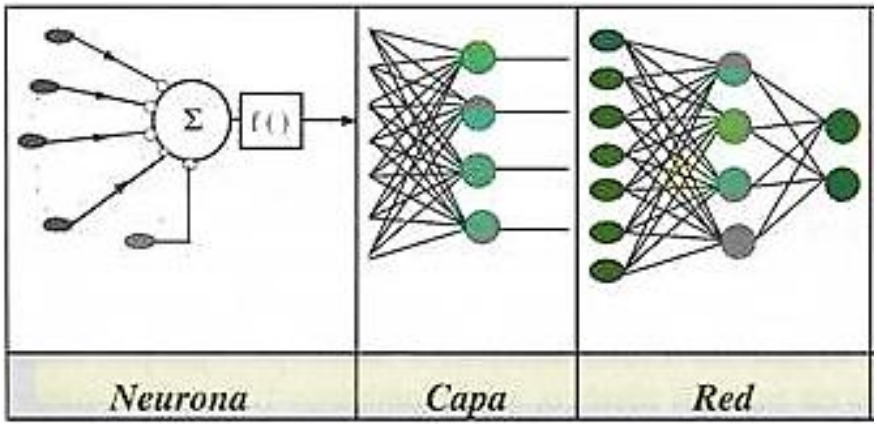

Figura 5: Las partes de una red neuronal, extraído de [2].

La arquitectura de una red neuronal está basada en una serie de conexiones hacia delante y multicapa. En la figura 5 se observan las distintas partes de una red neuronal artificial.

Una arquitectura muy común de las redes neuronales artificiales es *feedforward* y en este modelo la primera capa está conectada a la entrada de la red y cada capa subsiguiente tiene una conexión de la capa anterior. La última capa produce la salida de la red.

Las redes *feedforward* se pueden usar para cualquier tipo de mapeo de entrada a salida. Una red *feedforward* con una capa oculta y suficientes neuronas en las capas ocultas, puede ajustarse a cualquier problema de mapeo finito de entrada y salida [3].

Las versiones especializadas de la red *feedforward* incluyen redes de ajuste, tales como *fitnet*, y de reconocimiento de patrones como *patternnet*. Una variación de la red de avance es la red de avance en cascada *cascadeforwardnet*, que tiene conexiones adicionales de la entrada a cada capa, y de cada capa a todas las capas siguientes [3].

Las redes de cascadeforward son muy parecidas a las redes feedforward, pero incluyen una conexión de la entrada y cada capa anterior a las siguientes. Como ya pasaba con las redes feedforward, las redes de cascadeforward de dos o más capas son capaces de aprender cualquier relación finita de entrada y salida, dadas suficientes neuronas [4].

En redes neuronales artificiales, la capa de entrada está formada por las unidades o cualidades que introducimos como usuarios. Es la capa más subjetiva ya que entrarán los valores que consideremos que son importantes para hacer el análisis y representan los campos de entrada [5].

La capa oculta puede estar formada a su vez por varias capas y no tiene conexión directa con el entorno. La existencia de esta capa es la clave que capacita a una red neuronal a resolver problemas complejos.

La capa de salida es la unidad o unidades que proporcionan la respuesta de la red neuronal.

Las capas están formadas por neuronas, las cuales reciben valores de entrada procedentes del exterior o bien de otra neurona, lo transforman, y lo convierten en un único valor de salida o respuesta. Para hacer este proceso se utilizan ecuaciones concretas, denominadas funciones de activación.

Las neuronas se conectan entre sí formando capas y redes neuronales según una determinada arquitectura y sus conexiones tienen un peso que pondera entre cada neurona. Así pues, la salida de cada neurona se conecta como entrada a la neurona siguiente en una red que como hemos explicado anteriormente, avanza hacia delante. Por tanto, la entrada de una neurona es la suma de todas las conexiones que la preceden, es decir, es la suma de las salidas de las neuronas anteriores que se conectan a ella, multiplicadas por el peso de dicha conexión.

Como se observa en la figura 6, la neurona artificial, pues, viene constituida por tres partes básicas: entradas, pesos y salidas.

Por otro lado, una parte elemental de las redes neuronales son las unidades llamadas *bias*, como parte de cualquiera de las capas ocultas y de la capa de salida. Estas unidades presentan constantemente un nivel de activación de valor 1. Además, esta unidad está conectada a todas las unidades de la capa inmediatamente superior y los pesos asociados a dichas conexiones son ajustables en el proceso de entrenamiento. La utilización de esta unidad tiene un doble objetivo, mejorar las propiedades de convergencia de la red y ofrecer un nuevo efecto umbral sobre la unidad que opera [6].

Las entradas x_i pueden ser tanto digitales como analógicas.

Los pesos sinápticos w_i definen la intensidad con que interactúa cada neurona antes y después de la sinapsis. Dependiendo de si el peso es positivo o negativo dada una entrada positiva, se hablará de sinapsis excitadoras o inhibidoras respectivamente.

Las salidas o respuestas proporcionan el valor del potencial postsináptico de la neurona en función de sus pesos y entradas.

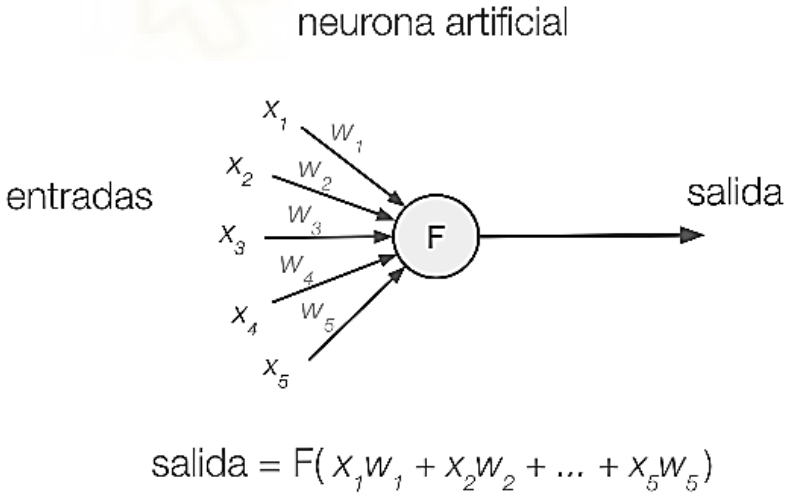

Figura 6: Modelo de neurona artificial, extraído de [2].

Como se observa en la figura 6, la neurona recibe una entrada x_i y un peso w_i y saca una respuesta o salida. Cada entrada se multiplica por su respectivo peso asociado a su conexión y seguidamente se suman y se les aplica la función de activación F, generando así la señal de salida o respuesta de cada neurona.

Las redes neuronales son capaces de modelizar un sistema sin la necesidad de programar dicho sistema específicamente, dependiendo de la entrada recibida, la red devolverá una salida o respuesta en función de lo aprendido, y es por este motivo que las redes neuronales deben ser entrenadas previamente. Por tanto, podemos definir dos procesos elementales: la etapa de entrenamiento y la etapa de funcionamiento.

2.1.2.1 ETAPA DE ENTRENAMIENTO

En la etapa de entrenamiento se usa un conjunto de datos de entrenamiento para determinar los pesos que definirán la red neuronal. Este proceso se desarrolla de manera iterativa, de acuerdo con los valores de entrenamiento, y el objetivo es minimizar el error que se comete entre la salida que devuelve la red y la salida deseada. Por tanto, para entrenar una red neuronal artificial, se presentan un conjunto de datos conocidos, los cuales entrarán en la red como entradas, siendo conocidas también las salidas, y ambas se propagarán por la red para obtener unas salidas concretas. De esta manera se ajusta los pesos siguiendo un determinado criterio para obtener las respuestas deseadas [7].

El entrenamiento puede distinguirse en aprendizaje supervisado y no supervisado.

2.1.2.1.1 APRENDIZAJE SUPERVISADO

El aprendizaje supervisado consiste en introducir en la red neuronal artificial un conjunto de valores conocidos, incluyendo la salida que se desea obtener, y esta, usando información detallada del error que se comete en cada paso, ajustará los pesos de forma iterativa hasta que la salida sea la deseada y la red será capaz de estimar la relación entre la entrada y la salida de forma natural, por tanto, el objetivo de los algoritmos de aprendizaje supervisado es que, dada una cierta entrada, los pesos se ajusten lo más eficientemente posible para que la salida que se genera sea lo más cercana a la real.

Se llama supervisado porque se conocen los patrones de salida, y dichos patrones supervisan de alguna manera a la red.

Dentro de los problemas de aprendizaje supervisado, existen dos tipos: de clasificación y de regresión. Mientras que para el modelo de clasificación el resultado es una clase, que viene dada por un número limitado de clases, en el modelo de regresión el resultado es un número dentro de un conjunto infinito de posibles respuestas.

Las redes neuronales tienen una capacidad de crear modelos de predicción. Permiten separar los datos según un conjunto de características que se extraen de ciertas observaciones de fenómenos observables. A este tipo de algoritmos se le denomina Machine Learning.

La regresión lineal es uno de los ejemplos más simples de machine learning. La regresión lineal es un modelo matemático que aproxima la relación entre las variables dependientes del modelo Y, las variables independientes X y un término adicional.

Las redes neuronales permiten realizar esta tarea de regresión, de forma lineal o no lineal. Las redes neuronales artificiales de regresión permiten predecir las variables de salida como función de las variables de entrada. Las entradas de la red serán variables "independientes", en este caso las características. Las variables dependientes del modelo deben ser numéricas, si fueran binarias sería un clasificador.

En este caso la red se puede utilizar en modo regresión para estimar el peso del huevo según las características de entrada. La eficiencia con la que la red estima los datos de salida se calcula como el coeficiente R, que se estima como la correlación entre los datos de entrada y de salida. Cuanto más próximo a 1.

2.1.2.1.1.1 ALGORITMO BACKPROPAGATION

Dicho algoritmo es muy utilizado dentro del mundo de las redes neuronales y facilita la modificación de los pesos y *bias* para permitir de este modo que las salidas deseadas coincidan con las salidas reales durante el proceso de entrenamiento de la red. Es un algoritmo básico de aprendizaje, que usa un entrenamiento supervisado y que pretende adaptar los parámetros de la red. Con el algoritmo de *backpropagation* se procura, por tanto, minimizar el error entre la salida que la red neuronal ha obtenido y la salida deseada por el usuario, utilizando patrones de entrenamiento [8].

Por lo general, la propagación en una red neuronal artificial se ejecuta haciendo un sumatorio de las diferentes entradas multiplicadas por los respectivos pesos y añadiéndole un valor de sesgo o *bias*.

2.1.2.1.2 APRENDIZAJE NO SUPERVISADO

En el aprendizaje no supervisado los patrones entran en la red sin introducir una salida deseada.

El objetivo principal del aprendizaje no supervisado es que los pesos de la red queden ajustados para que la red sea capaz de encontrar una configuración presente en los datos.

2.1.2.1.2.1 CLUSTERING

Se encarga de agrupar distintos datos, de los cuales no se conocen sus características comunes, en categorías para su posterior análisis. Se pretende descubrir algunas características en común de los distintos grupos por cercanía, para facilitar el estudio de los resultados [9].

2.1.2.2 ETAPA DE FUNCIONAMIENTO

Una vez la red está entrenada y los pesos se han ajustado correctamente, la red está lista para su utilización. Llegados a este punto, los pesos y la estructura quedan fijos y la red recibirá entradas y sacará salidas utilizando los parámetros fijados en la etapa de entrenamiento.

2.1.3 IMPLEMENTACIÓN DE LA GUI DE MATLAB

Hemos utilizado la GUI de Matlab porque nos proporciona las herramientas necesarias para crear una red neuronal personalizada por el usuario que es capaz de abordar exitosamente los objetivos de nuestro proyecto. Esta interfaz gráfica facilita la creación de una red neuronal desde las bases, siendo el usuario quien la desarrolla a su gusto personal basándose en las necesidades particulares de su propio interés [10].

Así mismo, permite desarrollar distintas variantes dentro de una amplia variedad de características, desde gráficos simples a menús complejos, que se adaptan a los distintos objetivos.

Mediante la creación de un script, hemos desarrollado ventanas gráficas que así mismo están formadas de varios elementos. La ventana principal consta de un menú muy básico que facilitará al usuario la creación de más ventanas que a su vez implementarán la red neuronal deseada. Esta ventana tiene conexión directa con el usuario.

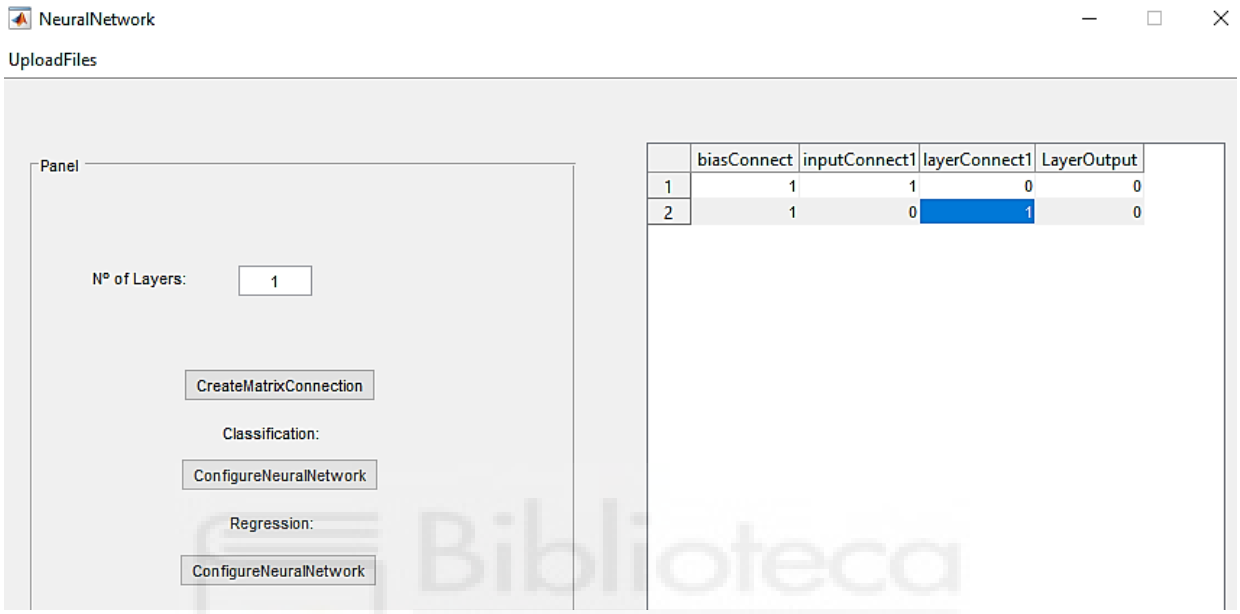

Figura 7: Ventana principal.

Otra manera funcional de crear ventanas con este programa es mediante la herramienta de diseño de GUIs de Matlab llamada GUIDE. En este proyecto no hemos utilizado este modelo concreto, pero es igualmente apto.

El elemento principal es la pantalla, y esta estará a su vez formada por distintas ventanas. A diferencia de las ventanas, solo puede haber una única pantalla y su identificador será siempre cero mientras que los identificadores de la ventana o ventanas siempre serán números enteros.

No puede haber dos ventanas activas simultáneamente, así pues, mientras estén todas las ventanas abiertas solo podremos tener una de ellas en modo activo. Este proceso también es extrapolable a los ejes de dicha ventana. Utilizando el comando *gcf* podremos obtener el identificador de la ventana que tenemos activa.

Algunos comandos que nos pueden ser útiles son:

- *get(id)* devuelve el valor concreto del objeto.
- *set(handle)* devuelve las propiedades de un objeto.
- *get* devuelve el valor de una o alguna propiedad concreta.
- *set* modifica el valor de una propiedad.

Utilizando *uicontrol* se crean todos los controles que hemos diseñado y es un comando que se implementa de la siguiente manera:

ejemplo=uicontrol(id_figura, 'Style', 'tipo de diseño'…

'String', opciones_texto…

'Position', posición_definida…

'CallBack', 'funcion')

- Style: se introduce una cadena concreta de valores y señala el control que se desea.
- Position: es quien define tanto el tamaño como la posición del objeto dentro de la figura. Dicha definición se lleva a cabo a partir de cuatro valores base, dos de ellos definirán las coordenadas del vértice inferior izquierdo y los otros dos definirán el ancho y el alto de dicho control. Vienen dados mediante un vector de cuatro coordenadas.
- String: Define el nombre que se muestra en el control.
- CallBack: Permite hacer la llamada a una función.
- Identificador figura: es la ventana que está activa y señala el objeto del control que se está creando.

Para la implementación de nuestra interfaz hemos utilizado los siguientes controles:

- Texto estático: es cuando la propiedad "style" tiene un valor "text" y su función principal es mostrar mensajes. Estos mensajes no son necesariamente una cadena, sino que también puede tomar valores numéricos. El tipo de texto se debe especificar en la propiedad de "string" y para cambiarlo, como hemos mencionado anteriormente, se debe utilizar el comando *set*.
- Botón: Al pulsar sobre él se ejecuta una acción concreta y vienen dados por objetos dentro de la pantalla.
- Texto editable: este control facilita al usuario que introduzca una cadena que es leída por la función *get.*
- Listas: muestra por pantalla un número determinado de opciones y permite al usuario elegir entre ellas.

2.1.4 DESARROLLO DE LA GUI

2.1.4.1 PARÁMETROS DE ENTRADA DE LA RED

La ventana principal de la GUI creada en Matlab tiene dos parámetros base a elegir: el número de capas y el número de neuronas (por defecto según los datos cargados).

El usuario tendrá, por tanto, el poder de decisión a la hora de elegir qué número de capas y neuronas quiere para la red neuronal que se creará de forma inmediata. Dependiendo de los valores que tome, la red se creara de una forma u otra. Es importante que el usuario tenga un conocimiento básico de redes neuronales artificiales para que su elección sea precisa y eficiente, puesto que la efectividad de los resultados depende de estos valores.

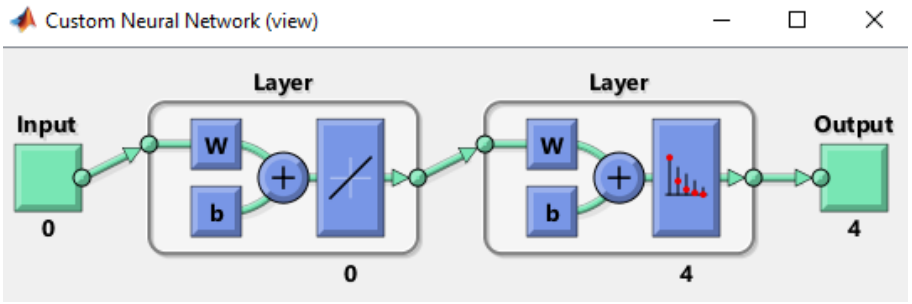

Figura 8: Red neuronal artificial creada con la interfaz.

La figura 8 es un ejemplo de red neuronal artificial que se crea cuando introduces los valores iniciales de la red en la ventana principal, en el modelo de clasificación.

En la ventana principal de la interfaz tendremos que clicar en UploadFiles->TrainData y seleccionar las dos matrices de datos con las que queremos trabajar (Figura Apéndice 1).

Asimismo, en esta misma ventana principal de nuestro proyecto, se crea una matriz de conexión que el usuario debe rellenar para la posterior creación de la red neuronal artificial personalizada [11]. En esta matriz se establecen distintas características (Figura Apéndice 2):

1. Establece en qué capas existen *bias*.
2. Establece las conexiones entre las neuronas de entrada y cada capa.
3. Establece las conexiones entre capas.
4. Establece las conexiones entre las capas y la salida.

Las redes neuronales artificiales que se crean con la interfaz tendrán tantas capas ocultas como se precise en la ventana principal, más una capa de salida que viene dada por defecto y para la cual también pueden modificarse sus conexiones. Esta capa estará en todas las redes neuronales que se creen.

Aunque generalmente las redes neuronales son hacia adelante, como se ha comentado anteriormente, es posible crear redes neuronales con retroalimentación, es decir, que una de las salidas de una capa sea al mismo tiempo una entrada de la misma capa o de una capa anterior. Al establecer las conexiones de la red neuronal, el usuario puede definir qué tipo de red quiere crear y, por tanto, será capaz de limitar o ampliar las conexiones de las capas a su gusto y, si lo considera oportuno, podrá crear retroalimentación entre las capas o conexiones hacia atrás y hacia adelante.
En general, todas las redes neuronales que tienen retroalimentación, tanto si es sobre una misma capa o una capa anterior, necesitan de un *delay*, es decir, este valor debe ser distinto de 0. El *delay* permite que la red tenga una respuesta dinámica finita a los datos entrada (Figura Apéndice 3.1).
La interfaz proporciona la opción de utilizar tanto clasificación como regresión. Los comandos que utiliza son las siguientes: *plotperform, plottrainstate, ploterrhist, plotconfusion, plotroc* (Figura Apéndice 3.2).

- *Plotperform*: es usada tanto en regresión como en clasificación.
- *Plottrainstate*: es usada tanto en regresión como en clasificación.
- *Ploterrhist*: es usada en clasificación.
- *Plotconfusion*: es usada en clasificación.

- *Plotroc*: es usada en clasificación.
- *Plotregression*: es usada para regresión.

Una vez creada la matriz de conexión, se crea automáticamente una red neuronal artificial basándose en los datos introducidos por el usuario. Al mismo tiempo, se creará una interfaz que permite elegir las propiedades de la red y las específicas de cada capa; de esta forma, se le permite al usuario entrenar y testear la red.

La red que se ha creado con anterioridad posee unas características generales, que son:

- La función de inicialización.
- La función de rendimiento.
- La función de entrenamiento.

Para la función de inicialización hemos elegido *intlay*, que inicializa la red de acuerdo con las funciones de capa; para la función de rendimiento se da a elegir entre *mse* (mean sqared error) y *crossentropy*, que es una función que mide el rendimiento de la red según la media de los errores al cuadrado; y finalmente, los pesos de la red se calibran con retropropagación de Levenberg-Marquardt, usando *trainlm* o el algoritmo *scaled conjugate gradient method* usando *trainscg*, que es una función que actualiza los valores de peso y sesgo, y es un algoritmo rápido de retropropagación, aunque necesita más memoria que otros algoritmos [12]. (Figura Apéndice 4.1;4.2;4.3)

$$\mathcal{L}(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij}\log(p_{ij})$$

Figura 9: Ecuación para la función *crossentropy*.

En la figura 9 podemos observar la ecuación de una función *crossentropy,* la cual es una medida de variables categóricas, es simétrica, fácil de interpretar, pero con escala univariante y con difícil diferenciación y convergencia.

$$MSE = \frac{1}{2n}\sum_{i}^{n}(\widehat{y_i} - y_i)^2$$

Figura 10: Ecuación para la función *mse*.

Tanto la función *mse* como la función *crossentropy* son funciones de coste, que tratan de determinar el error entre el valor estimado y el valor real, con el objetivo de que los parámetros de la red neuronal sean más eficientes.

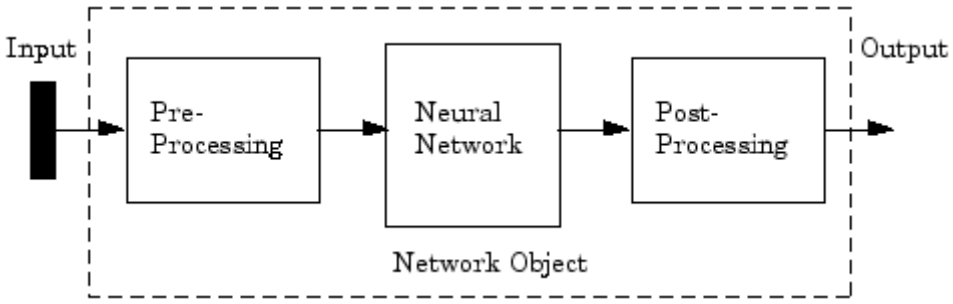

Figura 11: Procesamiento red neuronal, imagen extraída de [12].

A los datos de entrada y salida se les puede aplicar un procesamiento mediante unas funciones que transforman dichos valores proporcionando unos valores que son más adecuados para el funcionamiento de la red. Los más utilizados son *remuveconstantrows* y *mapminmax*. El primero de ellos elimina datos que son constantes mientras que el segundo normaliza las entradas en el rango -1 y 1. Estos parámetros han sido fijados por defecto así en nuestra interfaz.

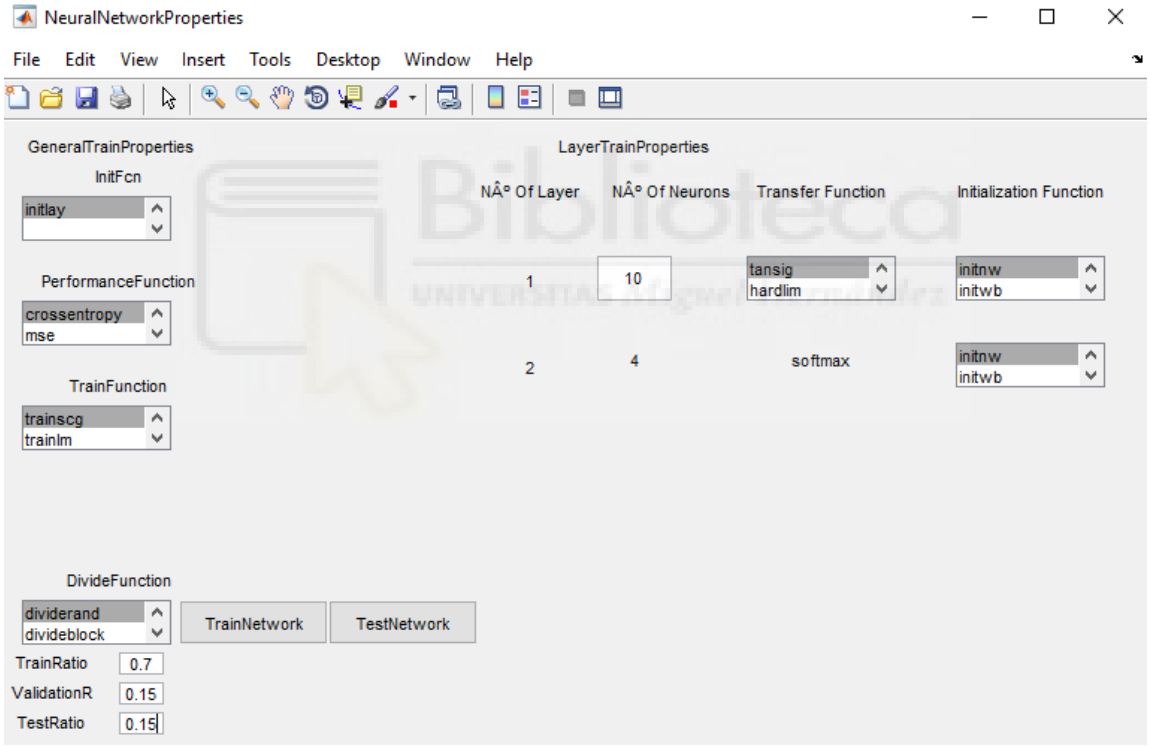

Figura 12: Ventana de configuración para una red de una capa.

En el entrenamiento de redes multicapa, la práctica general es dividir primero los datos en tres subconjuntos. El primer subconjunto es el conjunto de entrenamiento, que se utiliza para calcular el gradiente y actualizar los pesos y sesgos de la red. El segundo subconjunto es el conjunto de validación. El error en el conjunto de validación se controla durante el proceso de entrenamiento. El error de validación normalmente disminuye durante la fase inicial del entrenamiento, al igual que el error del conjunto de entrenamiento. Sin embargo, cuando la red comienza a sobreajustar los datos, el error en el conjunto de validación normalmente comienza a aumentar. Las ponderaciones y los sesgos de la red se guardan en el mínimo del error del conjunto de validación.

El error de la etapa de testeo no se utiliza durante el entrenamiento, pero se usa para comparar diferentes modelos. También es útil para trazar el error de testeo durante el proceso de entrenamiento. Si el error del conjunto de pruebas alcanza un mínimo en un número de iteración significativamente diferente del error del conjunto de validación, esto podría indicar una mala división del conjunto de datos.

Para entrenar la función de división de datos hemos elegido *dividerand* y *divideblock*. Mientras que la primera es una función que divide los datos de forma aleatoria, la segunda divide los datos en bloques contiguo y para establecer los valores de estas funciones se han de cambiar los parámetros de *trainratio, validationratio* y *testratio* [13].

Los datos dentro del aprendizaje supervisado se emparejan con etiquetas y se dividen en tres subconjuntos: *trainratio, validationratio* y *testratio*. Normalmente el porcentaje de entrenamiento es notablemente mayor que el de validación y test. Las proporciones más usadas son las siguientes:

- 70% entrenamiento, 15% validación y 15% test.
- 80% entrenamiento, 10% validación y 10% test.
- 60% entrenamiento, 20% validación y 20% test.

De cada capa se establecen ciertas características tales como el número de neuronas, la función de transferencia de cada capa y la función de inicialización de cada capa.

Para la implementación de nuestra GUI hemos utilizado tres funciones de transferencia básicas:

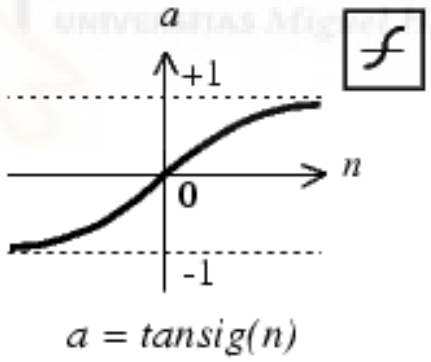

Figura 13: Función de transferencia tangente sigmoidal.

La figura 13 muestra gráfica de una función de transferencia tangente sigmoidal y transforma los valores que entran a una escala que va de -1 a 1, donde los más grandes tienden a 1 y los más pequeños a -1. Está centrada en 0.

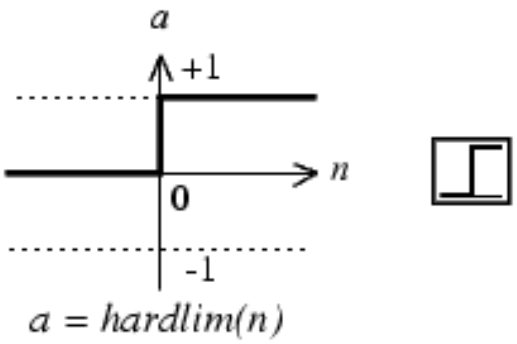

Figura 14: Función de transferencia hard-limit, extraída de [14].

La figura 14 muestra la gráfica de una función de transferencia hard-limit con una salida neuronal 0 siempre que el argumento de entrada n sea menor que 0. La salida neuronal será 1 si el argumento neto de entrada n es igual o mayor a 0.

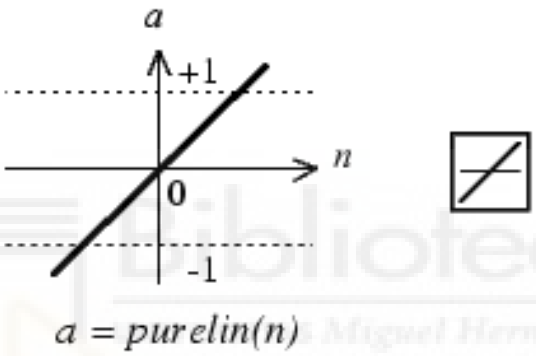

Figura 15: Función de transferencia lineal, extraída de [14].

En la figura 15 se muestra una función de transferencia lineal. Las neuronas de este tipo son generalmente usadas como aproximadores lineales en los filtros lineales.

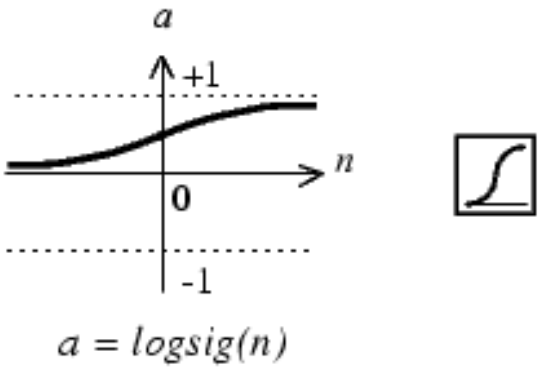

Figura 16: Función de transferencia Log-Sigmoid, extraída de [14].

La figura 16 muestra una función de transferencia sigmoide en la cual los valores de entrada van del 0 al 1 y los valores más grandes van a tender al 1 mientras que los valores más pequeños van a tender al 0. Esta función no está centrada en el 0 como se puede observar en la figura 16. Normalmente se utiliza dicha función para las redes de retropropagación.

Por otro lado, las funciones de inicialización de cada capa que se han utilizado para este proyecto son *initnw* y *initwb*. Mientras que la primera de ellas utiliza el método de Nguyen-Widrow para inicializar los pesos y *bias* de cada capa, la segunda función inicializa cada peso y *bias* con su propia función de inicialización.

Una vez establecidos los parámetros, los botones *trainNetwork* entrenan la red y ofrecen un resultado de la calidad que tiene la red aproximando a la función de predicción.

El botón *testnetwork* ofrece una predicción de nuevos datos. (Figura Apéndice 5)

3 RESULTADOS

En este apartado se va a entrenar y sacar los resultados de nuestros datos con las diferentes configuraciones de red neuronal que nos permite parametrizar nuestra interfaz:

1- Una capa con conexiones hacia adelanta y distinto número de neuronas.
2- Dos capas con conexiones hacia delante con el número de neuronas optimo calculado en el apartado anterior.
3- Dos capas con conexiones hacia atrás.

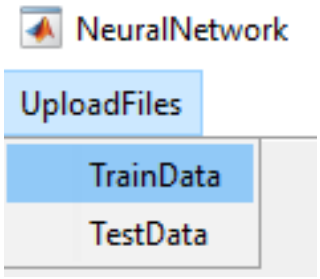

Figura 17: Primer paso para introducir los datos.

En la ventana principal (figura 7), el usuario debe introducir las matrices de los datos mediante *UploadFiles* y seguidamente, *TrainData* como se explicó anteriormente y se muestra en la figura 17. Tenemos dos matrices a introducir, la primera de ellas contiene 120 instancias de huevos con 4 características obtenidas por visión por computador en anteriores proyectos. Estas características son el perímetro, el área, el eje mayor y el eje menor del huevo.

Una vez introducidas las matrices de datos, marcaremos el número de capas que deseamos y seguidamente creamos la matriz de conexiones. Usando esta matriz podremos diseñar las distintas combinaciones para nuestra red dependiendo de las necesidades de la misma.

	biasCo...	inputConnect1	layerConnect1	layerConnect2	LayerOutput
1	1	1	0	0	0
2	1	0	1	0	0
3	1	0	0	1	0

Figura 18: Matriz de conexiones.

Para la figura 18 se ha creado una red neuronal con dos capas ocultas: layerConnect1 y layerConnect2. Para esta configuración en concreto hemos hecho conexiones hacia adelante y hemos conectado cada capa con su siguiente. Los 1 de la primera columna indican que existen *bias* en cada capa.

ImpuConect1 hace referencia a la entrada, y dependiendo de si marcamos un 1 en la fila 1, 2 o 3, esta entrada estará conectada a la capa 1, 2 o 3; siendo fila 1 layerConnect1; siendo la fila 2 layerConnect2 y siendo la fila 3 la capa de salida.

Como observamos en la figura 18, la entrada está conectada únicamente a la primera capa, pues tiene un 1 en la primera fila y en el resto de filas tiene 0. La primera capa oculta está conectada únicamente a la segunda capa oculta, puesto que en la columna de layerConnect1 solo se ha marcado 1 en la segunda fila y el resto son 0. La segunda capa oculta está conectada a la tercera capa, puesto que en la columna de layerConnect2, la única fila que tiene 1 es la tercera mientras que el resto de filas son 0.

LayerOutput es la salida y al no haber retroalimentación hacia atrás con otras capas, mantiene todas sus filas a 0.

Cuando ya hemos diseñado la matriz de conexiones, dependiendo de qué tipo de trabajo queremos que la red haga, elegiremos clasificación o regresión y posteriormente se creará la imagen de nuestra red neuronal. Conviene comprobar que la imagen de la red concuerda con los parámetros metidos para asegurarse de que todos los datos están bien introducidos en la matriz.

3.1 EJEMPLO DE CLASIFICACIÓN

Para este apartado estudiaremos el modelo de clasificación para distintas configuraciones de redes neuronales creada con la interfaz previamente explicada.

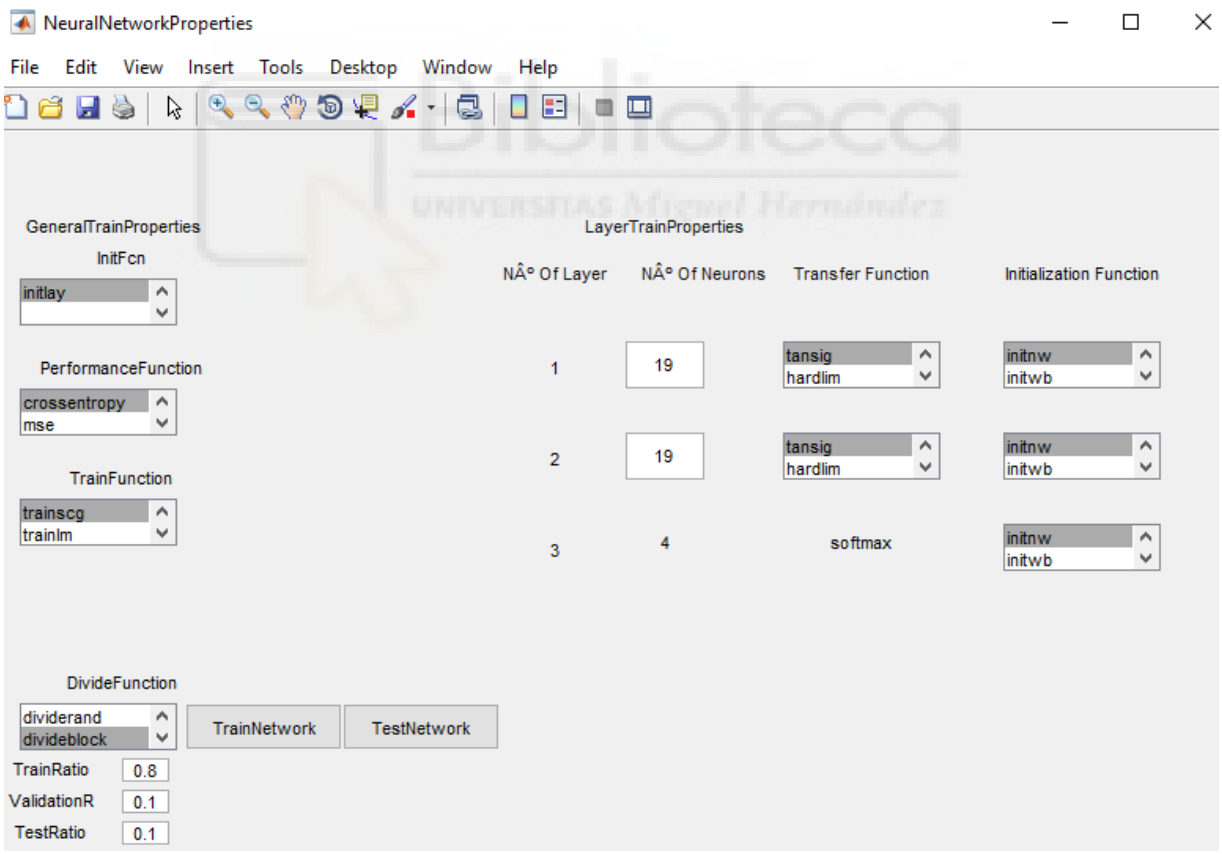

Figura 19: Ventana de configuración para una red de dos capas.

Como se muestra en la figura 19 en la ventana de configuración aparecen las distintas opciones sobre las que el usuario debe clicar, y además, debe elegir el número de neuronas que tiene cada capa. En el caso de la figura 19 hemos elegido 19 neuronas por cada capa oculta.

Los parámetros del problema de clasificación son los siguientes:
InitFcn: initlay.
PerformanceFunction: crossentropy.
TrainFunction: trainscg.
TransferFunction: tansig. Para las capas ocultas.
Softmax. Para la capa de salida.
InitalizationFunction: initinw para todas las capas.

Para reducir la variabilidad el modo de dividir los datos, en vez de utilizar *dividerand* (el más común), hemos utilizado *divideblock*. Antes de inicializar pesos y entrar a la red, establecemos la semilla de aleatoriedad de Matlab a 0 (rng="default").

Los porcentajes que hemos elegido para *TrainRatio*, *ValidationRatio* y *TestRatio* son 80%, 10% y 10% respectivamente, como se muestra en la figura 19.

Para entrenar la red, como se mencionaba anteriormente, se utiliza *divideblock* con los datos separados por bloques con un número proporcional de instancias que representa cada una de las clases. El 80% de estos 120 datos se han utilizado para el entrenamiento, el 10% se ha utilizado para el test y el restante para la validación, parámetros modificables a través de la interfaz.

Finalmente le damos a TrainNetwork y seguidamente clicamos en Confusión para poder ver los resultados.

Núm neuronas	S		M		X		XL	
	Accuracy (%)	Falsos positivos	Accuracy (%)	Falsos positivos	Accuracy (%)	Falsos positivos	Accuracy (%)	Falsos positivos
1	100	0	76,7	23,3	80	20	100	0
3	100	0	83,3	16,7	90	10	100	0
5	96,7	3,3	83,3	16,7	93,3	6,7	100	0
7	100	0	80	20	93,3	6,7	100	0
9	100	0	80	20	90	10	100	0
11	96,7	3,3	83,3	16,7	93,3	6,7	100	0
13	100	0	83,3	16,7	90	10	100	0
15	100	0	83,3	16,7	86,7	13,3	100	0
17	100	0	83,3	16,7	90	10	100	0
19	100	0	83,3	16,7	93,3	6,7	100	0
21	100	0	76,7	23,3	90	10	100	0
23	100	0	76,7	23,3	86,7	13,3	100	0
25	100	0	83,3	16,7	90	10	100	0

Tabla 2.1: Resultados de cada clase para una red neuronal de una capa.

Núm neuronas	TOTAL	
	Accuracy (%)	Falsos positivos
1	**89,2**	**10,8**
3	**93,3**	**6,7**
5	**93,3**	**6,7**
7	**93,3**	**6,7**
9	**92,5**	**7,5**
11	**93,3**	**6,7**
13	**93,3**	**6,7**
15	**92,5**	**7,5**
17	**93,3**	**6,7**
19	**94,2**	**5,8**
21	**91,7**	**8,3**
23	**90,8**	**9,2**
25	**93,3**	**6,7**

Tabla 2.2: Resultados totales para una red neuronal de una capa.

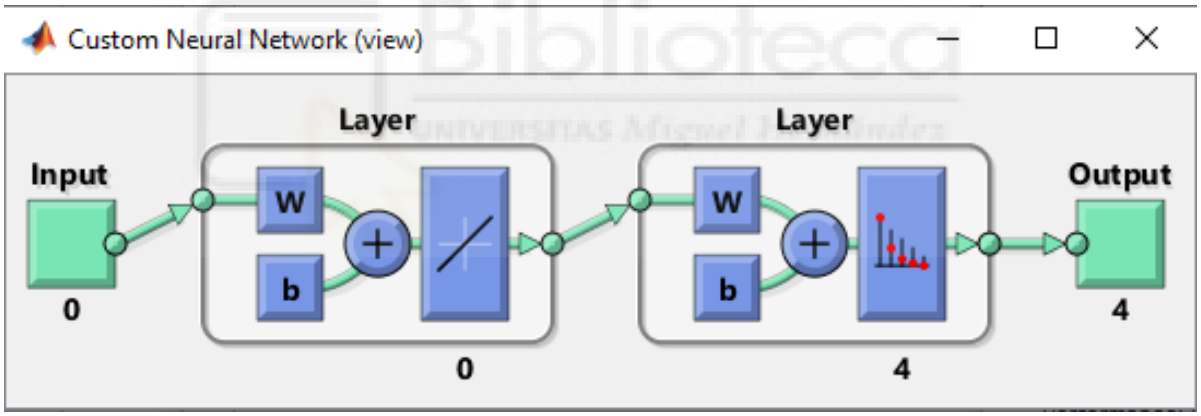

Figura 20: Red neuronal de una capa.

Las tablas 2.1 y 2.2 es la tabla de los resultados obtenidos al introducir en la interfaz una red neuronal de una sola capa con conexiones hacia adelante como se observa en la figura 20, y variando el número de neuronas desde 1 a 25.

El valor más bajo del total, como se observa en la tabla 2.2, se produce en la red de una capa con una neurona. Para este paradigma, como se observa en la tabla 2.1, de media las clases S y XL son las que mejor se diferencian, mientras que las intermedias, por tener dos límites de decisión, su accuracy es menor.

No se ha obtenido una tendencia que pueda relacionar mejores resultados con mayor número de neuronas. Sin embargo, los mejores resultados se han obtenido para 19 neuronas, con un 94,2%, como se muestra en la tabla 2.2.

Núm neuronas		S		M		X		XL	
		Accuracy (%)	Falsos positivos	Accuracy (%)	Falsos positivos	Accuracy (%)	Falsos positivos	Accuracy (%)	Falsos positivos
1	1	100	0	70	30	70	30	100	0
3	3	100	0	80	20	96,7	3,3	100	0
5	5	96,7	3,3	83,3	16,7	90	10	100	0
7	7	100	0	83,3	16,7	93,3	6,7	100	0
9	9	93,3	6,7	73,3	26,7	86,7	13,3	100	0
11	11	100	0	80	20	93,3	6,7	100	0
13	13	96,7	3,3	86,7	13,3	93,3	6,7	100	0
15	15	100	0	83,3	16,7	90	10	100	0
17	17	96,7	3,3	80	20	93,3	6,7	100	0
19	19	100	0	86,7	13,3	93,3	6,7	100	0
21	21	100	0	80	20	93,3	6,7	100	0
23	23	96,7	3,3	80	20	86,7	13,3	100	0
25	25	100	0	80	20	93,3	6,7	100	0

Tabla 3.1: Resultados de cada clase para una red neuronal de dos capas.

Núm neuronas		TOTAL	
		Accuracy (%)	Falsos positivos
1	1	**85**	**15**
3	3	**94,2**	**5,8**
5	5	**92,5**	**7,5**
7	7	**94,2**	**5,8**
9	9	**88,3**	**11,7**
11	11	**93,3**	**6,7**
13	13	**94,2**	**5,8**
15	15	**93,3**	**6,7**
17	17	**92,5**	**7,5**
19	19	**95**	**5**
21	21	**93,3**	**6,7**
23	23	**90,8**	**9,2**
25	25	**93,3**	**6,7**

Tabla 3.2: Resultados totales para una red neuronal de dos capas.

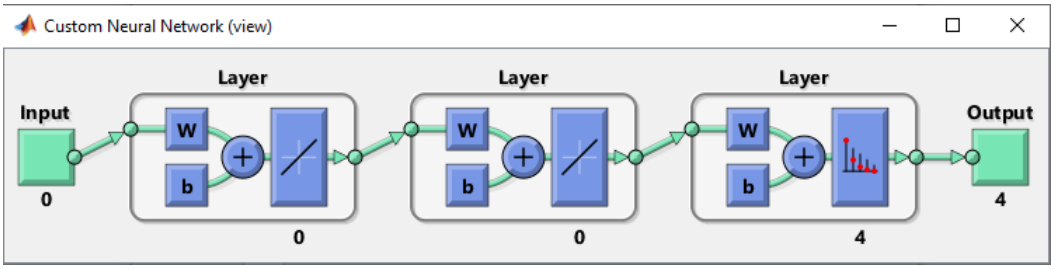

Figura 21: Red neuronal de dos capas.

Las tablas 3.1 y 3.2 es la tabla de los resultados obtenidos al introducir en la interfaz una red neuronal de dos capas con conexiones hacia adelante como se observa en la figura 21, y variando el número de neuronas desde 1 a 25 en ambas capas.

De forma similar a la tabla 2.2, observamos en la tabla 3.2 que los peores resultados se han obtenido para una neurona en cada capa oculta, mientras que los mejores resultados se han obtenido para 19 neuronas en ambas capas con un valor de accuracy de 95%.

Así mismo, hemos realizado algunos experimentos variando las conexiones con 19 neuronas en cada capa para hacer una comparativa de distintas estructuras de redes neuronales. Hemos utilizado redes con dos capas ocultas y una capa de salida, y 19 neuronas por cada capa oculta. Dependiendo de la configuración de la red podemos observar las variaciones de esta.

Para la primera columna de los experimentos que se comentan a continuación se han utilizado las mismas propiedades que las mencionadas anteriormente. Para la segunda columna de los experimentos, la configuración escogida fue la siguiente:

InitFcn: initlay.

PerformanceFunction: mse/crossentropy.

TrainFunction: trainlm.

TransferFunction: logsig. Para las capas ocultas.

Softmax. Para la capa de salida.

InitalizationFunction: initinw para todas las capas.

Se ha probado que trainlm es óptimo para este caso y obtiene mismos resultados con *mse* o *crossentropy*.

		Colum. 1	Colum. 2
TOTAL	**Accuracy (%)**	**91,7**	**95,8**
	Falsos positivos	**8,3**	**4,2**
S	**Accuracy (%)**	100	100
	Falsos positivos	0	0
M	**Accuracy (%)**	76,7	86,7
	Falsos positivos	23,3	13,3
X	**Accuracy (%)**	90	96,7
	Falsos positivos	10	3,3
XL	**Accuracy (%)**	100	100
	Falsos positivos	0	0

Tabla 4: Primer experimento.

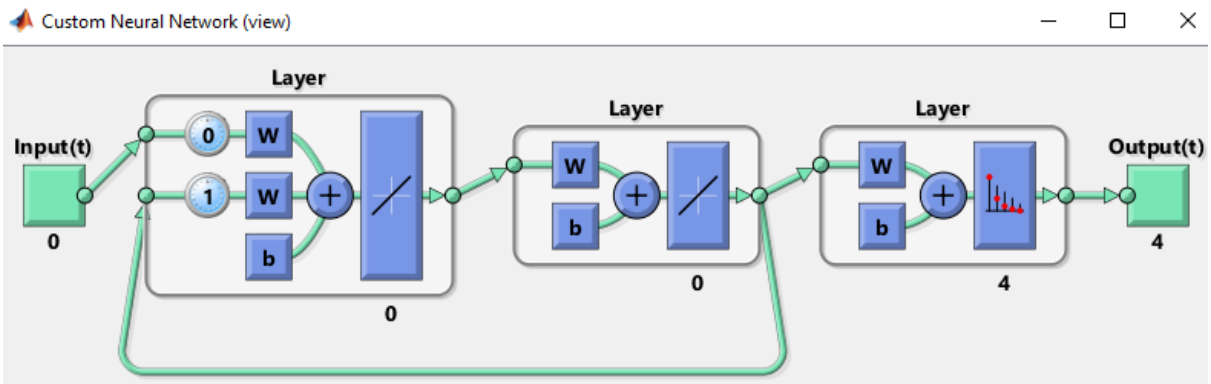

Figura 22: Red neuronal de dos capas del experimento 1.

El primer experimento se basa en una red neuronal de dos capas y 19 neuronas en cada capa, con una retroalimentación de la segunda capa a la primera capa. Aunque generalmente las redes neuronales van hacia adelante como hemos explicado anteriormente, también es útil en ocasiones que las redes neuronales tengan retroalimentaciones entre capas pasadas o entre la misma capa. En la tabla 4 podemos observar los resultados de la red neuronal de la figura 22 y se puede observar la retroalimentación entre capas.

		Colum. 1	Colum. 2
TOTAL	**Accuracy (%)**	**90**	**94,2**
	Falsos positivos	**10**	**5,8**
S	**Accuracy (%)**	90	100
	Falsos positivos	10	0
M	**Accuracy (%)**	86,7	80
	Falsos positivos	13,3	20
X	**Accuracy (%)**	83,3	96,7
	Falsos positivos	16,7	3,3
XL	**Accuracy (%)**	100	100
	Falsos positivos	0	0

Tabla 5: Experimento 2.

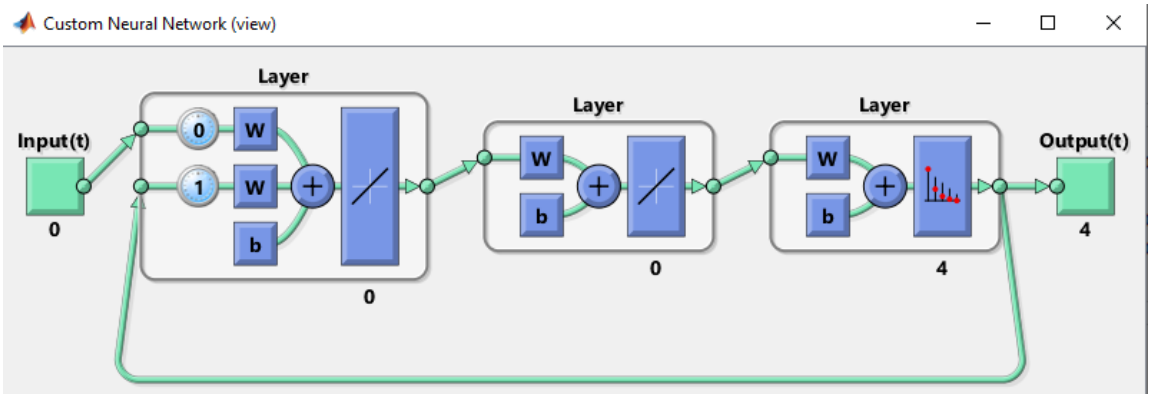

Figura 23: Red neuronal de dos capas del experimento 2.

El segundo experimento está basado en una red neuronal de dos capas que tiene 19 neuronas en cada capa y que además tiene una retroalimentación de la capa de salida a la primera capa. La figura 23 muestra la imagen de dicha red mientras que en la tabla 5 podemos ver los resultados que nos ha proporcionado.

		Colum. 1	Colum. 2
TOTAL	**Accuracy (%)**	**92,5**	**96,7**
	Falsos positivos	**7,5**	**3,3**
S	**Accuracy (%)**	100	100
	Falsos positivos	0	0
M	**Accuracy (%)**	80	90
	Falsos positivos	20	10
X	**Accuracy (%)**	90	96,7
	Falsos positivos	10	3,3
XL	**Accuracy (%)**	100	100
	Falsos positivos	0	0

Tabla 6: Experimento 3.

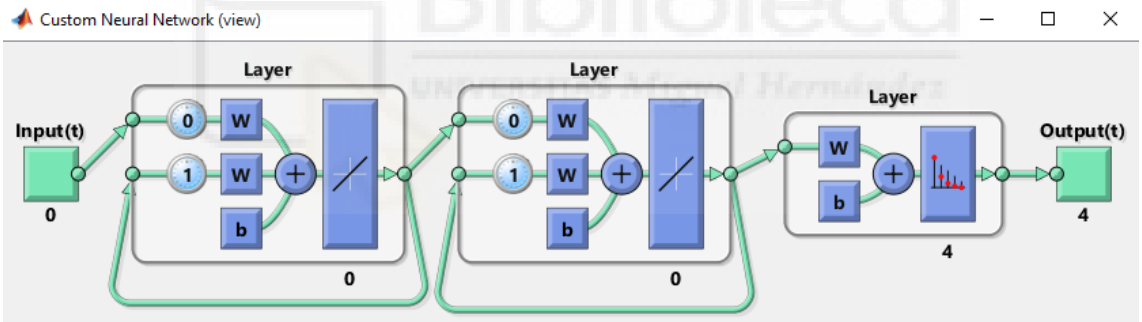

Figura 24: Red neuronal de dos capas del experimento 3.

Y por último, el tercer experimento, como se muestra en la figura 24, es una red neuronal de dos capas que tiene las dos capas ocultas retroalimentadas a sí mismas, es decir, la primera capa está conectada a sí misma mientras que la segunda capa oculta también está conectada a sí misma, y podemos ver los resultados de dicha red en la tabla 6.

Las diferentes configuraciones probadas con retroalimentación han mostrado un rendimiento próximo al 90%.

3.2 EJEMPLO DE REGRESIÓN

En este apartado estudiamos el problema desde el punto de vista de regresión, en base a lo planteado en el apartado anterior se procederá con la misma metodología y los mismos datos, pero bajo el enfoque ya descrito de regresión. Los resultados obtenidos se compararán más adelante con los resultados obtenidos en estudios anteriores [15].

Los datos de entrada para clasificación han sido una matriz de 120x4, a la cual se le realiza la transpuesta. En el problema de regresión la matriz es 120x1, el único valor de salida puede tomar 4 valores del [1,4], dependiendo de la etiqueta que le corresponda al dato de entrada.

Para el problema de regresión que nos concierne la capa de salida debe establecer una relación lineal entre los datos de la capa anterior y los de la capa de salida. Sin embargo, se puede realizar regresión lineal o no lineal dependiendo de la función de transferencia de las capas ocultas. La función más optima será un parámetro a determinar.

Para los parámetros de la red neuronal partimos de la siguiente lista, similar a la del apartado anterior, sin embargo, la función de coste o *performance* debe de ser *mse*:

InitFcn: initlay.

PerformanceFunction: mse.

TrainFunction: trainscg.

TransferFunction: purelin. Para las capas ocultas.

purelin. Para la capa de salida.

InitalizationFunction: initinw para todas las capas.

Se han realizado 3 pruebas más, basadas en las pruebas realizadas en el apartado anterior y se han elegido las mejores configuraciones. La capa de salida tiene por defecto la función *purelin.* Mientras que en las capas ocultas se han probado con funciones de transferencia lineales y no linealess y con dos algoritmos de entrenamiento *trainscg* y *trainlm*. *Trainlm* proporciona mejores resultados para funciones de transferencia no lineales. El mejor resultado en el experimento 4 se dio con la función *logSig* y el algoritmo *trainlm* 0,98. En los siguientes experimentos por tanto se utilizará *trainlm.*

Experimento 4	**R**		**R**		**R**	
	logSig		**tansig**		**purelin**	
TransferFunction	trainscg	trainlm	trainscg	trainlm	trainscg	trainlm
Nº Layers: 1	**0,96**	**0,98**	**0,94**	**0,97**	**0,96**	**0,96**

Tabla 7: Experimento 4.

En la tabla 7 podemos observar los resultados del experimento 4, que está formado por una sola capa oculta y 19 neuronas en dicha capa.

Experimento 5	**R**	**R**
	logSig	**tansig**
Nº Layers: 2	**0,983**	**0,97**

Tabla 8: Experimento 5.

Para dos capas y 19 neuronas en cada capa, como se muestra en la tabla 8, en el experimento 5 se consiguió un 0,983 para ambas capas con la función *logSig*, como se muestra en la figura 27.

Experimento 6	**R**
	logSig
Nº Layers: 2	**0,96**

Tabla 9: Experimento 6.

Por último, en la tabla 9 se muestra el resultado del experimento 6, el cual obtuvo un 0,96 para la mejor configuración determinada en experimentos anteriores. En el experimento 6 se ha utilizado la misma red neuronal que en el experimento 3, como se muestra en la figura 24, pues es la que mejores resultados ha obtenido con retroalimentación. En este caso, como en los anteriores, se han usado 19 neuronas por cada capa y dos capas ocultas.

4 DISCUSIÓN

El objetivo de este proyecto es crear un modelo que, de forma indirecta, mediante parámetros extraídos por visión por computador pudiera clasificar los huevos prediciendo su masa en las categorías S M L XL siendo S el más pequeño y XL el más grande. Esto permite a la compañía vender un producto dependiendo de su peso y optimizar las ventas. Por lo tanto, pretendemos con esto crear una red neuronal que maximice dicho acierto y reduzca los errores maximizando por lo tanto los beneficios de la industria. Este trabajo ha partido de anteriores trabajos en los cuales se han obtenido los datos por visión por computador. Además, también partimos de trabajos anteriores en los que se han probado distintos métodos de predicción como regresión o clasificación (supervisada y no supervisada).

La motivación de este trabajo era facilitar el diseño de una herramienta de predicción basada en redes neuronales a través de una interfaz que pudiera parametrizar el gran conjunto de parámetros que forman una red para ser cambiados por el usuario. Además, se pretendía mejorar los resultados obtenidos anteriormente y hacer una aproximación a los que pudiesen ser los mejores parámetros de una red que con los datos de los huevos de la industria maximizaran el acierto en la clasificación y redujesen los errores de predicción.
En este trabajo se ha obtenido un resultado en la predicción para 4 clases y 120 huevos del 95%, mejorando los resultados de anteriores estudios centrados en esta cuestión.

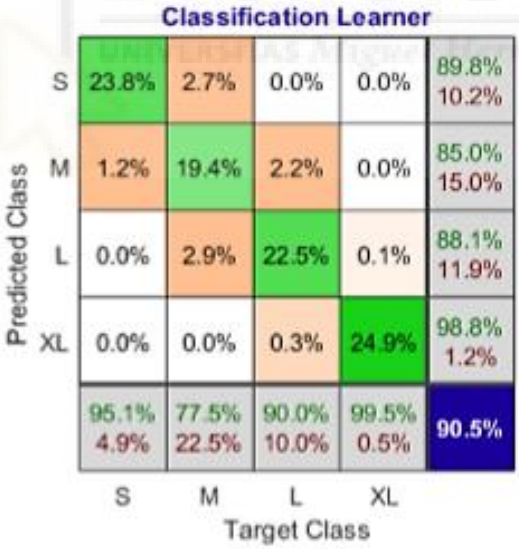

Figura 25: Matriz de confusión en la clasificación de huevos obtenida en anteriores estudios. Imagen extraída de [15].

Como podemos observar en la figura 25, los resultados obtenidos anteriormente, pese a no ser malos, son considerablemente peores que los que se obtienen con nuestra interfaz.

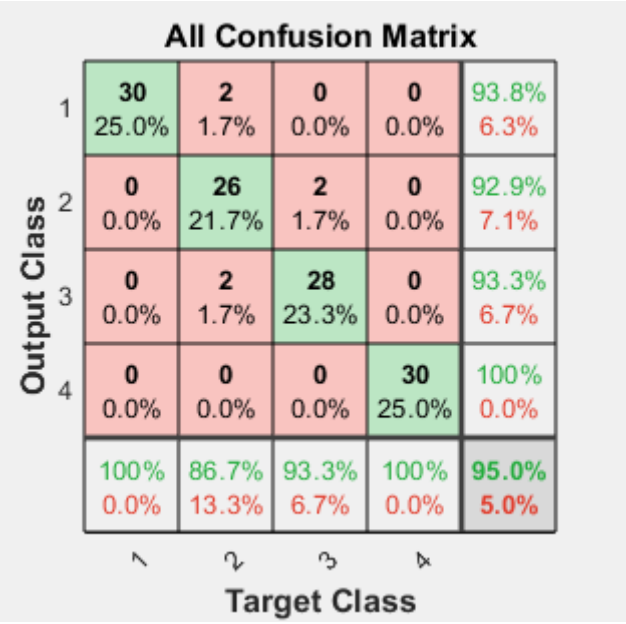

Figura 26: Matriz de confusión de nuestra interfaz para una red neuronal de dos capas ocultas con 19 neuronas en cada capa.

Comparando la figura 25 y la figura 26 nos podemos hacer a la idea de la mejora en el porcentaje total para clasificación que ha tenido nuestra interfaz con respecto a proyectos anteriores. Además, no solo el porcentaje total es más eficiente, sino que también están mejor definidos los límites entre las distintas clases.

A continuación, comentaremos los resultados obtenidos y discutiremos el camino a seguir para mejorar los resultados obtenidos.

En la tabla 2.2, los peores resultados fueron con una neurona, lo que nos puede indicar que cuanto mayor número de neuronas, el patrón de predicción será más complejo. El mejor resultado para una capa se obtuvo con 19 neuronas: 94,2%. Sin embargo, la tendencia no ha sido que cuantas más neuronas mejor resultado, y tampoco se ha llegado a un punto de inflexión claro. En los mejores resultados para las clases intermedias fueron con 7 neuronas del 80% y 96,7% respectivamente.

Como se observa en los experimentos de clasificación, tablas 4, 5 y 6, los resultados de la segunda columna son considerablemente mejores que los de la primera columna. Esto se puede deber a que se han cambiado las funciones de transferencia y los algoritmos de entrenamiento. Al ser redes de retroalimentación, concluimos que el algoritmo *trainlm* (96,7%) funciona mejor que el *trainscg* (92,5). El mejor resultado obtenido en estos experimentos ha sido de 96,7% del total y 90% y 96,7% para las capas intermedias M y X respectivamente tal y como se muestra en el experimento 3.

Teniendo en cuenta la cantidad limitada de datos que tenemos, las metodologías de *deeplearning* o aprendizaje profundo no son las más adecuadas para este tipo de *datasets*. Y, por tanto, las conclusiones podrían estar sesgadas.

A pesar de esto, los mejores resultados se han obtenido con dos capas y 19 neuronas por capa, tanto con una estructura hacia adelante como con retroalimentación. Así como en las tablas 2.1 y 2.2 para una sola capa, los peores resultados son con una neurona en la

capa oculta, para una red neuronal de dos capas ocultas los peores resultados vuelven a ser con una sola neurona en cada capa.

No obstante, somos conscientes de que las redes neuronales en sí tienen una alta variabilidad en los resultados debido a ciertas funciones que introducen aleatoriedad para mejorar su comportamiento. Para ello, decir que esta mejor configuración se ha obtenido estableciendo el parámetro de aleatoriedad a un valor fijo, por lo tanto, estos resultados aseguran cierta reproducibilidad en la práctica y por tanto en la industria en un modelo de negocio.

En cuanto a los resultados obtenidos en regresión, concluimos que la función óptima es *trainlm* cuando la función de transferencia de la capa oculta es no lineal. Los mejores resultados se han obtenido para la función de transferencia *logSig*. Como se observa en el experimento 5, los mejores resultados son para una red neuronal de dos capas y 19 neuronas cada una, con la función de transferencia *logSig* para las capas ocultas y el algoritmo de entrenamiento *trainlm.*

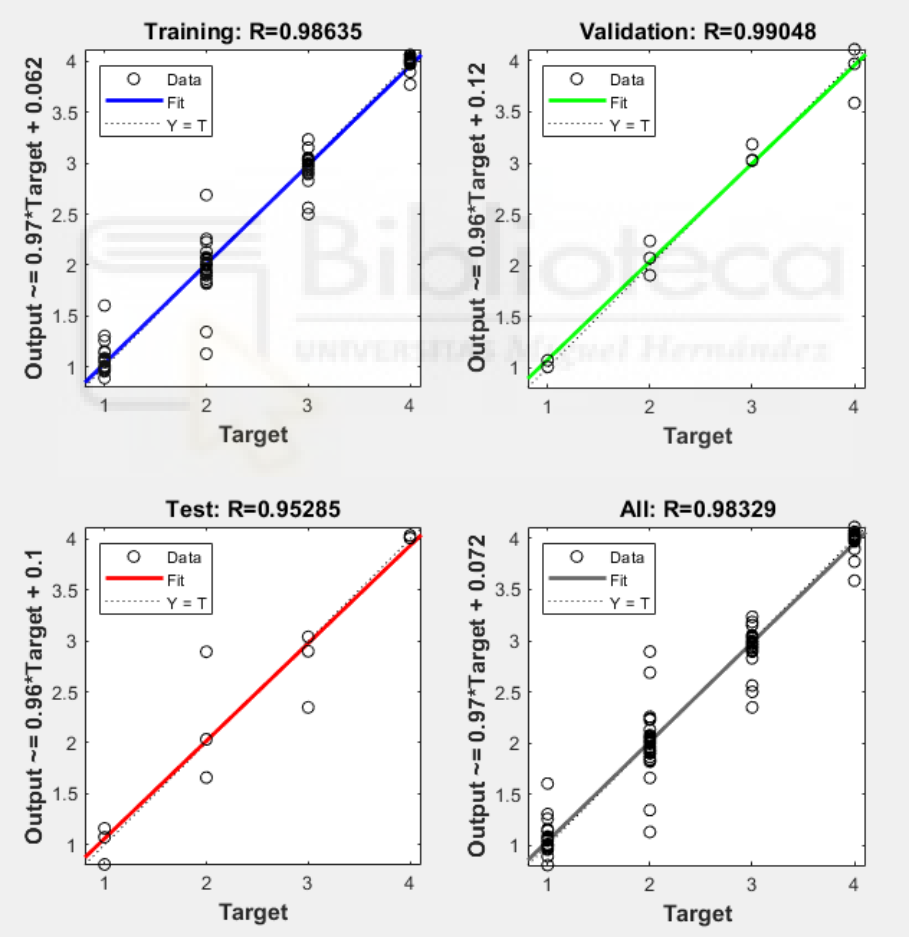

Figura 27: Resultados óptimos de regresión.

Para mejorar todos estos resultados que conciernen a técnicas de *deeplearning*, se podrían utilizar metodologías para aumentar el número de datos sin generar nuevas mediciones, si esto supusiera algún tipo de problema. Algunas de estas metodologías podrían ser: introducir ruido en los datos para generar nuevos datos que varíen de los originales e interpolar entre datos de una misma clase.

5 CONCLUSIONES

En este trabajo se ha desarrollado una interfaz que consigue diseñar al gusto del usuario una red neuronal personalizada para el problema de clasificación en el que se ha centrado este trabajo y también en el de regresión. La interfaz permite entrenar datos etiquetados y testear nuevos datos.

Basándonos en la bibliografía existente, hemos procurado incluir todos los parámetros que pudieran ser de interés para el usuario. Son configurables el número de capas, las conexiones entre estas y propiedades de cada capa, como el número de neuronas, la función de transferencia, la función de inicialización de pesos… Además de las propiedades más generales de la red como la función de ajuste de pesos general o la cantidad de datos que se utilizan para cada una de las fases de entrenamiento de la red.

Una de las partes más críticas en el diseño de esta interfaz puede ser la dificultad para utilizarla por un usuario inexperto o que no esté familiarizado con el uso de redes neuronales. Bajo nuestro punto de vista, una de las partes que se podría mejorar es la tabla para la conexión de las capas.

Esta interfaz permite, cambiando alguno de los parámetros, comprobar cuál es la configuración más apta para nuestro conjunto de datos, lo que puede ser de gran utilidad en la industria como ya se ha comentado anteriormente.

6 BIBLIOGRAFÍA

[1] Alikhano, J., Miroslavov Penchev, S., Dimitrova Georgieva, T., Moldazhanov, A., Shynybay, Z. Daskalov, P. An Indirect Approach for Egg Weight Sorting using Image processing. *University of Ruse "Angel Kanchev", Ruse, Bulgaria.* Director: Plamen Daskalo.

[2] "Conceptos básicos de redes neuronales". http://grupo.us.es/gtocoma/pid/pid10/RedesNeuronales.htm (consultado el 01/06/2020).

[3] "Feedforwardnet" https://es.mathworks.com/help/deeplearning/ref/feedforwardnet.html (consultado el 01/06/2020).

[4] "Cascadeforwardnet" https://es.mathworks.com/help/deeplearning/ref/cascadeforwardnet.html (consultado el 01/06/2020).

[5] Matich, D. J. (2001). Redes Neuronales: Conceptos básicos y aplicaciones. *Universidad Tecnológica Nacional, México*.

[6] Olabe, X. B. (1998). Redes neuronales artificiales y sus aplicaciones. *Publicaciones de la Escuela de Ingenieros*.

[7] Izaurieta, F., & Saavedra, C. (2000). Redes neuronales artificiales. *Departamento de Física, Universidad de Concepción Chile*.

[8] Aldabas-Rubira, E. (2002). Introducción al reconocimiento de patrones mediante redes neuronales. *IX Jornades de Conferències d'Enginyeria Electrònica del Campus de Terrassa, Terrassa, España, del 9 al 16 de Diciembre del 2002*.

[9] Tineo, R. J. M., Sandoval, E. A. P., Becerra, C. I. V., Vargas, E. P., Apaza, G. M., & Salinas, E. A. (2011). Modelo de clustering basado en redes neuronales para identificar el perfil de los alumnos por segmento enfocado a los servicios de tecnologías de información de la universidad peruana unión. *Revista de Investigación Business Intelligence*, *1*(2).

[10] Moreno Rodríguez, Alfonso. Desarrollo de una interfaz gráfica de redes neuronales usando Matlab. Proyecto de fin de carrera, Universidad Carlos III, Madrid, España. Director: Isabel González Farias.

[11] "Edit Shallow Neural Network Properties ". Matlab. https://es.mathworks.com/help/deeplearning/ug/create-and-train-custom-neural-network-architectures.html (consultado el 01/06/2020).

[12] "Choose Neural Network Input-Output Processing Functions". Matlab. https://es.mathworks.com/help/deeplearning/ug/choose-neural-network-input-output-processing-functions.html (consultado el 01/06/2020).

[13] "Divide Data for Optimal Neural Network Training". Matlab. https://es.mathworks.com/help/deeplearning/ug/divide-data-for-optimal-neural-network-training.html;jsessionid=e449559d8f04b068e5bd60e7e5c5 (consultado el 01/06/2020).

[14] (1994-2005) "Transfer Functions". The MathWorks, Inc. http://matlab.izmiran.ru/help/toolbox/nnet/model23.html

[15] Schürer-Waldheim, S., Penchev, S. Matlab Classification Learner app – an alternative to neural networks (ANN) for egg sorting problem. Comparison of supervised learning methods for small data amounts. *Department for Biomedical, Health and sports Engineering, UAS Technikum Wien, Austria. Department for Automatics and Mechatronics, University of Ruse "Angel Kanchev", Ruse, Bulgaria.*

ANEXOS

```
%Update data for Train
% --------------------------------------------------------------------
function UpdateTrainData(hObject, eventdata, handles)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%We display in console the usser commands for update the train data
disp('Seleccionar los datos para entrenar la red')
[dataTrain,path] = uigetfile('*.mat');
handles.dataTrain=load([pwd '\' dataTrain]);
disp('Seleccionar los etiquetas para entrenar la red')
[labelTrain,path] = uigetfile('*.mat');
handles.labelTrain=load([pwd  '\' labelTrain]);
handles.dataTest=handles.dataTrain;
guidata(hObject, handles);

%Update data for Test
% --------------------------------------------------------------------
function UpdateTestData(hObject, eventdata, handles)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%We display in console the usser commands for update the label data
disp('Seleccionar los datos para testear la red')
[dataTest,path] = uigetfile('*.mat');
handles.dataTest=load([pwd  '\' dataTest]);
```

Figura Apéndice 1

Se utiliza la función uigetfile para especificar el formato del archivo a cargar y la función load para cargarlo del path activo.

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%f =figure('Name','layerConnect','NumberTitle','off');

    %We associate the network with the handle of the actual figure
    handles.net=network;
    %Input neural network
    handles.net.numInputs = str2num(string(handles.edit1.String));
    %The number of layers +1 that is the output layer
    handles.net.numLayers = str2num(string(handles.edit3.String))+1;
    %The table is created for edit. We allocate rows as the number of
    %layers and the colums as the connection bias, the input, the
    %connection between layers and the output
    handles.uit = uitable('Data',zeros(str2num(string(handles.edit3.String))+1,str2num(string(handles.edit1.String))...
        +str2num(string(handles.edit3.String))+1+1),'Position', [450 0 400 550]);
    set(handles.uit,'ColumnEditable',true(1,str2num(string(handles.edit1.String))+str2num(string(handles.edit3.String))+1+1));
    %handles.net.outputs{1}.size=4;

    %We create a struct with the of each column
    struct={};
    struct(1)={'biasConnect'};
    for i=1:str2num(string(handles.edit1.String))
        struct(1+i) ={'inputConnect'+string(i)};
    end
    for i=1:str2num(string(handles.edit3.String))
        struct(1+str2num(string(handles.edit1.String))+i) ={'layerConnect'+string(i)};
    end
    struct(str2num(string(handles.edit1.String))+str2num(string(handles.edit3.String))+2)={'LayerOutput'};
    handles.uit.ColumnName=struct;
    %We update the GUI
    guidata(hObject, handles);
```

Figura Apéndice 2

La función uitable permite crear una tabla con la propiedad ColumnEditable; se especifican que todas sean editables para poder ser modificadas por el usuario.

```
% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%Because we call the callback function we cant use the handle for the
%figure, insted of this we  put the value of the handle to a global
%variable
global handlesNet
global handlesData
%We define by code the type of network
handles.problem='classification';
tableData = get(handles.uit, 'data');
%The connections in the table are equal to the connections in the network
handles.net.inputs{1}.size=size(handles.dataTrain.train1,2);
handles.net.biasConnect=tableData(:,1);
handles.net.inputConnect=tableData(:,2:(handles.net.numInputs+2)-1);
handles.net.layerConnect=tableData(:,(handles.net.numInputs+2):((handles.net.numInputs+2)+handles.net.numLayers)-1);

%%%%%
%%The backward connection need a value delay. For dafult 1
for i=1:size(handles.net.layerConnect,2)
    handles.net.layerConnect(i,find(handles.net.layerConnect(1:i,i)==1));
    columnas=find(handles.net.layerConnect(1:i,i)==1);
    for j=1:length(columnas)
       handles.net.layerWeights{columnas(j),i}.delays = [1];
    end
end
%%%%%
```

Figura Apéndice 3.1

Existe un análogo para regresión, con las variaciones especificadas en el apartado 2.1.4.1 PARAMETROS DE ENTRENAMIENTO DE LA RED .

```
%%%%%
handles.net.inputs{1}.processFcns =  {'removeconstantrows','mapminmax'};

%Handles=global variables
handlesData.dataTrain=handles.dataTrain;
handlesData.labelTrain=handles.labelTrain;
handlesData.dataTest=handles.dataTest;

%We put the output connections by default
outputConnection=[];
for oC=1:handles.net.numLayers
    if oC==handles.net.numLayers
        outputConnection=[outputConnection;1];
    else
        outputConnection=[outputConnection;0];
    end
end
handles.net.outputConnect=[outputConnection'];

%LayerOutputStandar
handles.net.layers{handles.net.numLayers}.size = size(handles.dataTrain.train1,2);
handles.net.layers{handles.net.numLayers}.transferFcn='softmax';

%The user has a guide for view the net
view(handles.net);
%Update the GUI
guidata(hObject, handles);

%Plot functions for classification. By default
handles.net.plotFcns = {'plotperform','plottrainstate','ploterrhist','plotconfusion','plotroc'};

NNP=figure('Name','NeuralNetworkProperties','NumberTitle','off');
%Allocate the net
handlesNet=handles.net;
```

Figura Apéndice 3.2

```
%% These are the text and list use for the interface
%Test showing a propertie
GTrainP = uicontrol(NNP,'Style','text',...
    'String','GeneralTrainProperties',...
    'Position',[10 380 130 30]);
%InitFcn
    GTrainP = uicontrol(NNP,'Style','text',...
        'String','InitFcn',...
        'Position',[15 360 130 30]);
        %Listblox with the posibilities
        InitFcn={'initlay'};
        uicontrol('Style', 'listbox','Position',[15 340 100 30],...
        'string',InitFcn,'Callback',{@InitGeneral})

%PerformanceFcn
    GTrainP = uicontrol(NNP,'Style','text',...
    'String','PerformanceFunction',...
    'Position',[15 290 130 30]);
        %Listblox with the posibilities
        InitFcn={'crossentropy','mse'};
        uicontrol('Style', 'listbox','Position',[15 270 100 30],...
        'string',InitFcn,'Callback',{@PerformanceGeneral})
```

Figura Apéndice 4.1

Ejemplos de declaración de texto visual y de las listas desplegables para dos de las funciones: la de inicialización de los pesos general y la de performance o función de coste.

```
%Init callback Fcn
function Initfunction(hObj,event,layer,handles)
global handlesNet
v=get(hObj,'value');
%Assing a property to the network depend of the usser choose
handlesNet.layers{layer}.initFcn=event.Source.String{v};
```

Figura Apéndice 4.2

Llamada a la función de callback de inicialización cuando el usuario decide cambiar el valor seleccionando uno entre los de la lista.

```
%Mse error performance
function PerformanceGeneral(hObj,event,handles)
global handlesNet
v=get(hObj,'value');
%Assing a property to the network depend of the usser choose
handlesNet.performFcn=event.Source.String{v};
```

Figura Apéndice 4.3

Llamada a la función de callback de Performance cuando el usuario decide cambiar de valor seleccionando uno entre los de la lista.

```
%TrainCallback function
function TrainNetwork(hObj,event,handles)
global handlesNet
global handlesData
%handles.dataTrain
%handlesData.dataTrain.train1;
%handlesData.labelTrain.etiquetas
%preTrainedTeresa=handlesNet;
handlesNet.inputs{1}.size
%view(handlesNet)
rng('default')
handlesNet = init(handlesNet);
% handlesNet.numInputs
% size(handlesData.dataTrain.train1)
% size(handlesData.labelTrain.etiquetas)

%TrainMethod
handlesNet = train(handlesNet,handlesData.dataTrain.train1',handlesData.labelTrain.etiquetas');
%postTrainedTeresa=handlesNet;

%Test CallBack function
function TestNetwork(hObj,event,handles)
global handlesNet
global handlesData
%Simulate the network
Y = sim(handlesNet,handlesData.dataTest.test1);
%We save the output in a file .mat
save('output.mat','Y')
```

Figura Apéndice 5

Funciones para entrenar y testear la red. Con el comando save se guarda el archivo generado con las predicciones de la red testeada.