Universidad Autónoma de Madrid

Facultad de Ciencias

Clasificación de galaxias

Exploración de las redes neuronales para la clasificación de imágenes

Autor:
Carlos Calles Gómez & Pablo Granizo Cuadrado

21 de abril de 2023

Índice

1. Resumen

En este informe relataremos el proceso que hemos seguido para lograr diseñar una aplicación capaz de clasificar galaxias a través de imágenes. Implementando redes neuronales con MATLAB y otras diseñadas por nuestra cuenta lograremos el objetivo de clasificar distintos tipos de imágenes.

Más adelante, aplicaremos distintos algoritmos con el fin de indagar en el funcionamiento de estas redes neuronales.

A lo largo del informe señalaremos y explicaremos los distintos procesos generales y específicos que hemos utilizado con el fin de lograr nuestro objetivo. Pasando por lo más básico de las redes neuronales hasta llegar al pre-procesado de imágenes implementado directamente en la aplicación.

2. Introducción

La clasificación de objetos estelares es una práctica que se lleva haciendo desde que existe la astronomía. Siempre se ha hecho de forma manual, pero modernas herramientas computacionales nos permiten hacerlo automáticamente y a escala. Específicamente, nos interesaremos en las redes neuronales como herramienta para resolver este problema y en las galaxias como objeto a clasificar. Primero usando los métodos que nos proporciona MATLAB, y luego intentando recrear desde cero los algoritmos subyacentes. Esto además nos permitirá adaptar las redes a cualquier tarea de clasificación de imágenes.

Nuestra motivación surge, por tanto, de un deseo de automatizar el proceso de clasificación y de entender como funcionan las redes neuronales fundamentalmente. Adicionalmente, el *machine learning* es hoy en día una habilidad muy deseada, así que este proyecto nos sirve como introducción al campo. Por lo tanto, formalizamos una serie de objetivos: encontrar una manera de clasificar galaxias a partir de sus imágenes, diseñar una red con la ayuda de MATLAB para que las clasifique, recrear el código detrás de una red neuronal básica, y probar esta implementación en un par de tareas simples de clasificación. Esta también es una cronología aproximada de los hitos conseguidos en el trabajo.

Concretamente, crearemos dos redes para clasificar galaxias, una que las clasifique en función de si son elípticas o espirales, y otra que use la clasificación completa de *Galaxy Zoo* [18] , detallada más adelante.

También crearemos una red simple con dos entras y una salida, que imite una puerta XOR, es decir, toma como entradas [0 o 1, 0 o 1] y solo da como salida 1 si una de las entradas es 0 y la otra 1. En el resto de los casos debe de dar 0. En adición a esta red, crearemos otra con una capa oculta entre las entradas y salidas, cuyo objetivo será clasificar imágenes con dígitos del 0 al 9 escritos a mano. Esto corresponde a un problema clásico de la IA y del *Machine Learning* [9].

Finalmente, para poder usar las redes de una manera fácil y simple, diseñamos una interfaz gráfica con MATLAB mediante *App Designer*.

2.1. Teoría sobre la clasificación de galaxias

Con el fin de poder clasificar las galaxias a través de imágenes, utilizaremos la clasificación de Hubble [12] que data de 1930. La parte más simple de esta clasificación diferencia las galaxias en espirales o elípticas únicamente en función de su forma. Con esto entrenaremos la primera red, la más simple.

Para lograr una clasificación más precisa, utilizaremos la clasificación de *Galaxy Zoo* [18] que se basa en seguir un árbol de decisión (1) en el que se observan cualidades individuales y se van añadiendo. Estas cualidades son, entre otras: si tiene o no tiene anillos, si tiene un borde texturizado, el tipo de patrón que tiene y como se comporta este patrón, si tiene algún artefacto en el centro, etc. Estas son las características principales, que a su vez tiene subcategorías y se pueden combinar entre sí. A continuación se muestra el árbol de decisión detallado que hemos empleado:

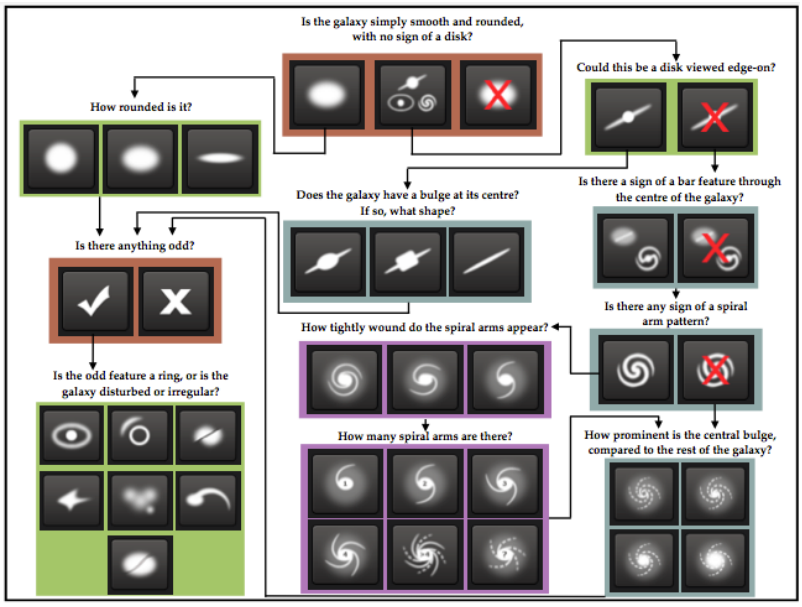


Figura 1: Árbol de decisión para la clasificación de galaxias

El proyecto de *Galaxy Zoo*, además de proporcionar un sistema de clasificación con el que entrenar la red neuronal grande para clasificar galaxias detalladamente, nos da los datos en sí. En este proyecto se dio acceso a montones de imágenes del cielo tomadas por el *Sloan Sky Survey* [18] a través de una aplicación, permitiendo que el usuario las clasificase en distintas categorías. Con el tiempo se consiguió una base de datos con imágenes de galaxias y su clasificación. Es esta base de datos la que usamos para entrenar nuestras redes neuronales [3].

2.2. *Deep Learning* y redes neuronales

El *Deep Learning* es el campo en el que se incluyen los distintos tipos de redes neuronales que existen. Este, a su vez, esta dentro del *Machine Learning*, que estudia varios algoritmos con los que un ordenador puede "aprender" a realizar una tarea, usando grandes cantidades de datos. Y ambos campos son solo una rama del estudio de la inteligencia artificial.

Dentro del *Deep Learning*, vamos a usar redes neuronales convolucionales, dadas por MATLAB, y diseñar redes neuronales profundas, para entender los métodos de este campo.

Una red neuronal simple se caracteriza por emplear una capa de entrada, una de salida, y un número de capas ocultas, todas compuestas por neuronas. Estas reciben un vector de entrada, correspondiente a la capa anterior (o, si es la primera capa, el vector entrada directamente), multiplican cada valor con su peso correspondiente y le suman su sesgo (2), para luego introducir el resultado en su función de activación. El resultado de esta operación, que se llama activación de la neurona (1) , se unirá al resto de activaciones de las neuronas de una misma capa para formar el vector de entradas de la siguiente capa. Una vez se llega a la última capa de la red, la última función de activación producirá una salida, que será la salida final de la red.

Los pesos y sesgos de la red se denominan **parámetros**, y se agrupan por capas en matrices. Cuando se crea una red neuronal, lo primero es decidir el número de capas que tiene, cuantas neuronas tiene cada una y cual es la función de activación de cada capa. Al principio, las matrices que contienen los parámetros de cada capa estarán inicializadas con números aleatorios, así que para una entrada dada, la salida no tendrá sentido.

$$a = f(z) \tag{1}$$

$$z = wx + B \tag{2}$$

Para poder mejorar la red, es decir, conseguir que aprenda, diseñamos una función de coste o pérdida (***loss***) que cuantifica el parentesco entre la salida obtenida y la deseada. Queremos, entonces, minimizar el resultado de la función de coste, para que las salidas obtenidas sean lo más parecido posible a las deseadas. Para minimizar esta función, calculamos su gradiente con respecto a los parámetros de la red (3) (4) y restamos dicho gradiente a cada parámetro. Si realizamos este proceso muchas veces, con muchas entradas y salidas, al final conseguiremos bajar la pérdida y aumentar la precisión de nuestra red neuronal.

$$\frac{\partial C}{\partial w} = \frac{\partial C}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w} \tag{3}$$

$$\frac{\partial C}{\partial b} = \frac{\partial C}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial b} \tag{4}$$

Estos son los fundamentos de las redes neuronales más básicas. Por supuesto que existen otras arquitecturas, ecuaciones y métodos más complejos, pero esta base es común a prácticamente todo el campo del *Deep Learning*.

3. Métodos

3.1. Obtención de los *datasets*

Un *dataset* es un conjunto de datos que han sido tabulados, es decir, son datos que cumplen ciertas condiciones (ya sea de tamaño, calidad, o cualidades) y que poseen un nombre o identificación. La mayoría de *datasets* que contienen imágenes están compuestos por una carpeta de estas y una tabla con las propiedades de cada una.

Algo importante a tener en cuenta es que, si estos *datasets* contienen imágenes todas con una serie de condiciones iguales (por ejemplo, que este centrado el objeto a clasificar), la red entrenada tendrá problemas al intentar clasificar imágenes que no cumplan estas condiciones. Por lo tanto, se deben de proporcionar imágenes a la red lo más parecidas posibles a las que usa para entrenar. Otra solución es conseguir un *dataset* más variado, pero esto requiere de más tiempo de entrenamiento y mejores arquitecturas de red para funcionar.

El *dataset* que hemos empleado para entrenar las redes neuronales de clasificación de galaxias, como mencionamos antes, ha sido obtenido del *Galaxy Zoo* [3], la tabla con la información de cada imagen estaba compuesta con la probabilidad de que cada imagen perteneciese a cierta categoría. Importamos estos datos a MATLAB, leyendo la tabla y cargando las imágenes de una carpeta.

Para la red neuronal que trata de recrear una puerta XOR, el *dataset* lo creamos artificialmente, ya que solo son 4 posibilidades ($[0,0], [1,0], [0,1], [1,1]$), repetidas muchas veces.

Finalmente, para la clasificación de dígitos usaremos la base de datos MNIST [9], que es la usada tradicionalmente para este problema.

3.2. Redes Neuronales mediante MATLAB

MATLAB pone a nuestra disposición una herramienta llamada *Machine Learning Toolbox*, que logra hacer el diseño de redes neuronales más simple a través de una función llamada *Trainnet* [2] , que compone una red neuronal a la que alimentaremos con los datos (imágenes) de nuestro *dataset* y con ciertas categorías que escojamos. Realizará el entrenamiento de la red, modificando los parámetros de cada neurona automáticamente, a la vez nos muestra dos gráficas en las que podemos ver como la precisión y la pérdida varían con el tiempo.

Para conseguir la mejor precisión posible las dos tareas de clasificación de galaxias, cargamos una red ya preentrenada [5] con millones de imágenes y 1000 categorías: *ResNet-50* [6], con 50 capas ocultas. Sustituimos las dos últimas capas: "fc1000", "ClassificationLayerfc1000". Estas capas están hechas para una red de 1000 categorías, así que las remplazamos con dos capas parecidas pero para 2 categorías (espiral y elíptica) y, en el otro caso, 3166 categorías (caminos distintos del árbol de decisión (1)).

3.3. Redes Neuronales desde cero

Con la teoría a nivel cualitativo ya vista en introducción, ahora nos centraremos en los 3 algoritmos fundacionales de las redes neuronales.

3.3.1. *Feed Forward*

El *Feed Forward* es la computación que realiza una red cada vez que se introduce una entrada, y da como salida la predicción de la red (que puede ser, por ejemplo, un vector de dimensión n, con la probabilidad de que la entrada pertenezca a cada una de las n categorías, normalizado para que la suma de todos los elementos sea 1).

En nuestro caso, definimos una clase para las capas y otra clase para los modelos. La clase modelo contiene un vector tipo *cell* con un objeto tipo capa en cada entrada del vector. La clase capa contiene un vector de sesgos, una matriz de pesos y una función anónima, que es la función de activación. De estas últimas uso 3 distintas: *sigmoid*, *lRelu* y *softmax* [13] [8], las dos primeras son simplemente las más comunes en redes neuronales de clasificación, la última es un poco más interesante. La función *softmax* funciona de manera que, dado un vector, su salida es una distribución de probabilidad, con la suma de los elementos igual a 1. Esto se suele aplicar a la última capa, para que la salida de la red sea un vector con las probabilidades de que pertenezca a cada categoría.

Ya hablamos de las operaciones que realiza una neurona, pero en realidad estas operaciones se realizan por capas, ya que es más rápido [7]. Primero pasamos el vector entrada a la primera capa, donde lo multiplicamos por la matriz de pesos y le sumamos el vector de sesgos. Guardamos el vector resultante como paso intermedio z, que pasamos a la función de activación de la capa, obteniendo el vector de activaciones a de la capa. Este vector se pasa como las entradas de la siguiente capa, y se repite el proceso hasta llegar a la última capa, cuyas activaciones son el vector salida.

$$A_{(L)} = f_{(L)}(W_{(L)} * A_{(L-1)} + B_{(L)}) \tag{5}$$

En la ecuación (5) se representa la computación realizada por cada capa. El subíndice indica la capa a la que pertenece cada vector o matriz.

3.3.2. *Backpropagation*

El algoritmo de *Backpropagation* consiste en calcular el gradiente de la función de coste respecto de los parámetros de la red. El nombre proviene de que su cálculo se realiza desde la última capa a la primera, ya que para calcular este gradiente en una capa se necesita el gradiente en la siguiente.

La función de coste no se calcula comparando solo la salida de la red con la esperada, sino que calculamos la función de coste con cada entrada y designamos la función de coste final como la suma de todas las anteriores. Hemos usado las funciones MSE (*Mean Squared Error*) (6) (para la red XOR) y CCE (*Categorical Cross Entropy*) (7) (para la red clasificadora de dígitos)[19] [15] [13].

$$C(Y, Y', n) = \frac{1}{n}\sum_{i=1}^{n}(Y - Y')^2 \tag{6}$$

Donde Y es la matriz con cada fila correspondiente a un vector salida verdadero, e Y' la matriz con cada fila correspondiente a un vector salida predicho por la red. La cantidad de filas es n.

$$C(Y, Y', n, m) = \frac{1}{n}\sum_{i=1}^{n}\big(-\sum_{j=1}^{m} Y log(Y')\big) \tag{7}$$

Aquí las variables son iguales al MSE, añadiendo m, que es la cantidad de categorías a clasificar.

El resultado de este algoritmo son dos vectores, $\frac{\partial C}{\partial W}$ y $\frac{\partial C}{\partial B}$, de tipo *cell* donde cada elemento corresponde al gradiente para una capa:

$$\frac{\partial C}{\partial W^{(L)}} = \begin{pmatrix} \frac{\partial C}{\partial W^{(L)}_{1,1}} & \cdots & \frac{\partial C}{\partial W^{(L)}_{1,n}} \\ \cdots & & \\ \frac{\partial C}{\partial W^{(L)}_{m,1}} & \cdots & \frac{\partial C}{\partial W^{(L)}_{m,n}} \end{pmatrix} \tag{8}$$

$$\frac{\partial C}{\partial B^{(L)}} = \begin{pmatrix} \frac{\partial C}{\partial B^{(L)}_{1}} \\ \cdots \\ \frac{\partial C}{\partial B^{(L)}_{m}} \end{pmatrix} \tag{9}$$

Aquí, n es el número de neuronas de la capa $L-1$ y m el número de neuronas de la capa L.

Antes de realizar el proceso de *Backpropagation*, hacemos un *Feed Forward* con todas las entradas, guardando las activaciones y pasos intermedios de cada capa. Ahora, para cada capa (empezando por la última) calculamos la matriz $\frac{\partial C}{\partial W^{(L)}}$ y el vector $\frac{\partial C}{\partial B^{(L)}}$ [16] . En este caso, denotamos K como la última capa, L como una cualquiera, f^L es la función de activación de la capa L y $\cdot$ un producto escalar.

$$\delta^{(K)} = \nabla_{a^{(K)}} C \cdot f'(z_i)^{(K)} \tag{10}$$

$$\delta^{(L)} = ((W^{(L+1)})^T \delta^{L+1}) \cdot f'(z)^{(L)} \tag{11}$$

$$\frac{\partial C}{\partial W^{(L)}_{i,j}} = a^{(L-1)}_j \delta^{(L)}_i \tag{12}$$

$$\frac{\partial C}{\partial b_i^{(L)}} = \delta_i^{(L)} \tag{13}$$

Por último, especificamos las funciones de activación y sus derivadas.

1. Sigmoide

$$f(x) = \frac{1}{1 + e^(-x)} \tag{14}$$

$$f'(x) = f(x)(1 - f(x)) \tag{15}$$

2. Softmax

$$f(x_i) = \frac{e^{x_i}}{\sum_{i=1}^{n} e^{x_i}} \tag{16}$$

$$f'(x) = f(x)(1 - f(x)) \tag{17}$$

3. lRelu

$$f(x) = max(0.01x, x) \tag{18}$$

$$\begin{cases} f'(x) = 1 & \text{si } x > 0 \\ f'(x) = 0.01x & \text{si } x \leq 0 \end{cases} \tag{19}$$

3.3.3. Bucle de entrenamiento y *Gradient Descent*

Ahora que ya tenemos el *Feed Forward* y el *Backpropagation*, solo falta una pieza del puzzle: el bucle de entrenamiento.

Aplicaremos el algoritmo de *Gradient Descent*, en el vamos iterativamente añadiendo a los parámetros de la red el gradiente de la función de coste. De esta manera, los parámetros siguen un camino que minimiza esta función.

Originalmente, este proceso se realiza con cada entrada de la red, hallando el gradiente del coste respecto a la entrada y sumando este gradiente multiplicado por un hiperparámetro, el *learning rate* a los parámetros originales de la red. Sin embargo, con muchas entradas, este proceso es muy costoso computacionalmente, así que usamos una modificación del algoritmo original: *Stochastic Gradient Descent*.

Es igual al original, solo que ahora hallamos el gradiente con respecto a varias entradas, no solo una. De esta manera, los cálculos del apartado anterior se hacen con matrices de activación, aprovechando el poder de las GPU [7] a la hora de realizar cálculos con matrices. La cantidad de

entradas que usemos en cada iteración de *Backpropagation* se denomina *batch size*, y es nuestro segundo hiperparámetro.

En resumen, el bucle de entrenamiento realiza un número de iteraciones igual al número total de entradas entre mi *batch size*, en cada una modificando los parámetros de la red según lo indicado por el *Backpropagation* para alcanzar valores cada vez más pequeños de la pérdida.

3.4. Preprocesado de imágenes

Antes de alimentar una red neuronal capaz de procesar imágenes, debemos realizar algunos ajustes a estas para que cumplan ciertas condiciones y así la red poder operar con ellas [4]. En nuestro caso, aplicamos los siguientes procesos, dependiendo de la red en cuestión, a las imágenes del *dataset* que usamos para entrenar las redes:

- *Cropping* o recorte: Con este proceso recortamos y centramos la imagen hasta un tamaño de 224 por 224 píxeles, en el caso de las redes entrenadas por MATLAB, o 20 por 20 píxeles, para el reconocimiento de dígitos.

- *Resize* o cambio de tamaño: Aquí aplicamos un algoritmo de MATLAB para lograr reducir el tamaño de las imágenes sin perder apenas la información que contiene, a diferencia del *cropping*. Esto lo usamos en todos los casos, pues así nos aseguramos que el tamaño es exactamente el que debe de introducirse en la red.

- *IM2Gray* o decoloración: Pasamos de escala de colores o *RGB* a una escala de grises, este proceso solo es utilizado en el caso de la red reconocedora de núeros.

Cabe destacar que cualquier imagen que es introducida a la aplicación *App1* sufre los mismos procesos automáticamente para que pueda ser utilizada, aplicándose una combinación de ellos dependiendo de la red que se quiera usar.

4. Resultados

4.1. Clasificación simple de galaxias

Nuestra mejor red de clasificación simple de galaxias consigue clasificar con precisión un 87 % de las imágenes de su *dataset* de evaluación. En este caso y el siguiente, hemos separado las imágenes en un 90 % para entrenamiento y un 10 % para validación (con el que calculamos la precisión).

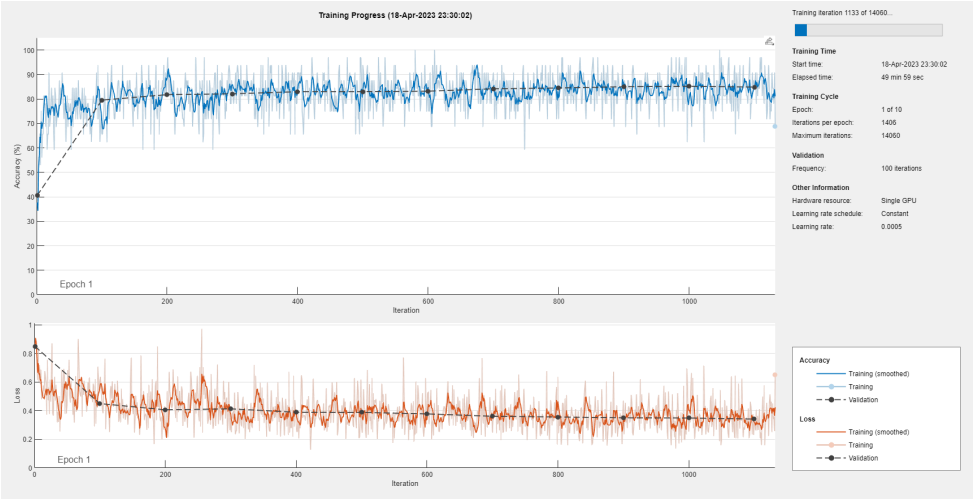


Figura 2: Gráfica de entrenamiento de la red de clasificación simple de galaxias. En azul está representada la precisión, y en naranja la pérdida de la red. En el panel derecho aparecen algunos datos, como la iteración en la que se encuentra o el tiempo que lleva entrenando. Solo se muestra una porción del entrenamiento aquí.

En la figura (2), nos interesa especialmente las líneas negras rayadas, pues describen los mismos datos (precisión y pérdida), pero en las imágenes que usamos de validación. A continuación, vamos a ver la matriz de confusión de esta red, donde seleccionamos una porción de nuestro *dataset* a clasificar y luego comprobamos si coincide con la clasificación correcta.

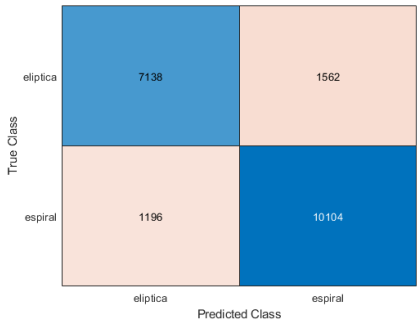


Figura 3: Matriz de confusión de la red de simple clasificación simple de galaxias. En el eje vertical se representa la categoría verdadera obtenida del *dataset* y en el eje horizontal la categoría predicha por la red neuronal. Se escogen 20000 imágenes, y se especifica en cada cuadrante de la matriz la cantidad de imágenes que pertenecen a él.

4.2. Clasificación avanzada de galaxias

Con la red de clasificación avanzada, conseguimos un máximo de 29 % de precisión. Esto, aunque puede parecer baja, es excepcionalmente alta. Esto se debe a que poseemos más de 3000 categorías, resultado de los distintos caminos del árbol de decisión. Además, muchas de estas categorías son muy similares entre sí, diferenciando solo entre número de brazos espirales o el tamaño del bulto central, cosa que muchas veces es difícil de apreciar en la imagen.

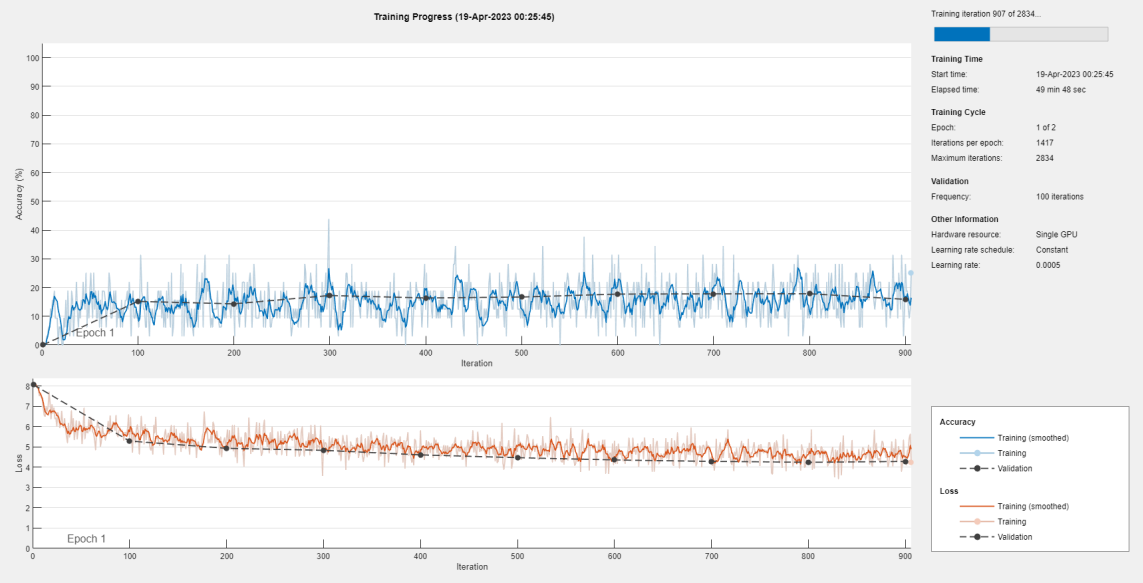
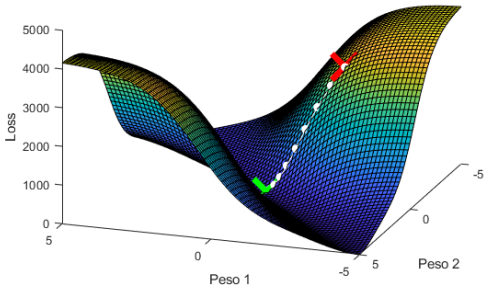


Figura 4: Gráfica de entrenamiento de la red de clasificación avanzada de galaxias. En azul está representada la precisión, y en naranja la pérdida de la red. La gráfica en cuestión no cubre el entrenamiento del modelo usado, sino uno posterior de menor precisión e iteraciones.

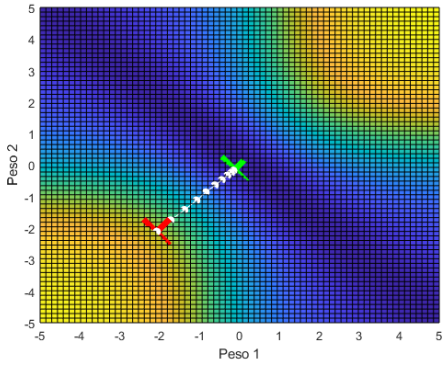
En la gráfica de entrenamiento vemos que rápidamente se alcanza una precisión baja y no aumenta mucho más (aunque la gráfica mostrada no corresponde exactamente al modelo de la *app*). Si quisiéramos mejorar este modelo, se podría recurrir a un tipo distinto de red, que nos diga la probabilidad en cada una de las 37 categorías iniciales de la tabla de datos con las características de la galaxia. De este manera reconstruiríamos después de la predicción el camino por el árbol de decisión, evitando las más de 3000 categorías. Pese a todo, el resultado obtenido es aceptable para el método que hemos usado.

4.3. Fundamentos de las redes neuronales mediante XORNet

Esta red neuronal, que trata de imitar una puerta XOR, no consigue realmente una precisión satisfactoria. Pero esto es a propósito, ya que usando solos dos parámetros la red nunca podrá alcanzar un valor para el que las entradas posibles ([0 o 1, 0 o 1]) siempre produzcan las salidas esperadas (0 si las dos entradas son iguales, 1 en otro caso). Lo que si podemos hacer es representar la función de coste como una función de dos variables. Así, es posible visualizar la pérdida inicial, la pérdida en cada paso del entrenamiento, y la final. Nos sirve entonces como una buena introducción al diseño de redes neuronales desde cero.



(a) Función de coste de la red vista de perfil



(b) Función de pérdida de la red vista desde arriba

Figura 5: Función pérdida de la red. La cruz roja muestra la pérdida inicial y la verde la final. Los puntos blancos son los pasos intermedios del entrenamiento, unidos cronológicamente por líneas. Estos puntos se muestran encima de la superficie de todas las posibles pérdidas de nuestra red, en función de sus dos parámetros.

4.4. Clasificación de números con numNet

Usando ahora los algoritmos explicados en la sección 3.3, tratamos de clasificar imágenes en blanco y negro de tamaño 20x20 con dígitos escritos a mano. Obtenemos varias redes, variando sus hiperparámetros y el número de neuronas de la capa oculta. La máxima precisión conseguida es del 74 % sobre las imágenes de entrenamiento.

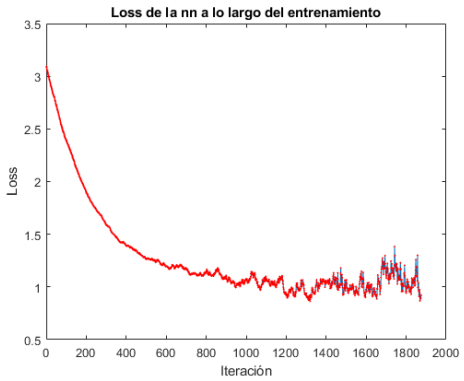


Figura 6: Gráfica de pérdida de la red clasificadora de dígitos. En el eje horizontal representamos la iteración en el bucle de entrenamiento y en el vertical la pérdida de la red con respecto a todas las imágenes en ese punto.

Al final del entrenamiento vemos que la pérdida se vuelve inestable. El motivo exacto de este fenómenos nos es desconocido, pero pensamos que se puede deber a que un mínimo local ya ha sido alcanzando, y la red no puede salir de él.

Usando una matriz de confusión recreada de la misma manera que en nuestra red simple de clasificación de galaxias, podemos observar los dígitos que más suele confundir la red con otros dígitos.

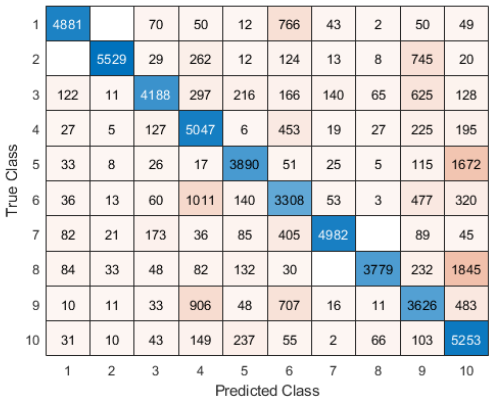
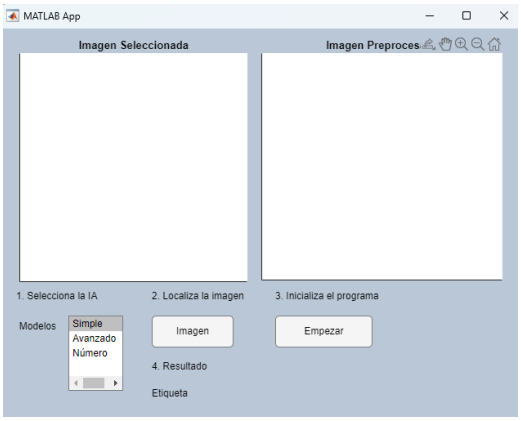


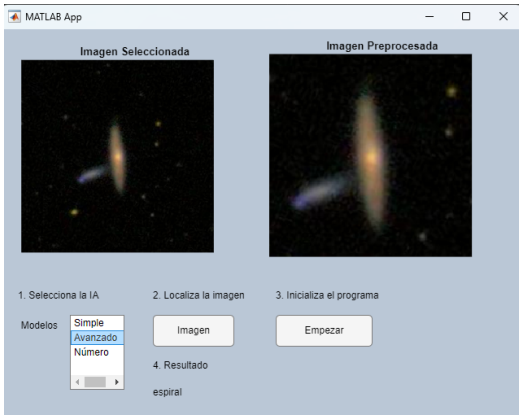
Figura 7: Matriz de confusión de la red clasificadora de números. En el eje vertical se representa la categoría verdadera obtenida del *dataset* y en el eje horizontal la categoría predicha por la red. Aquí usamos las 60000 imágenes disponibles en el *dataset*.

4.5. Aplicación final

Finalmente, la aplicación ha sido diseñada en *MATLAB app designer*, está compuesta por seis componentes principales, como pueden ver en las siguientes imágenes (8), que son: dos lienzos para representar la imagen seleccionada y la imagen procesada, un selector para elegir la red que se va a utilizar, un botón para seleccionar la imagen a utilizar, otro botón para empezar el proceso de clasificación, y un cuadro de texto que exponga la clase predicha.



(a) Aplicación sin entradas



(b) Aplicación en uso, al introducir y clasificar una galaxia con la red simple

Figura 8: Imágenes de la aplicación

5. Referencias

[1] Clasificar una imagen con una red preentrenada - matlab & simulink - mathworks españa.

[2] Crear una red de clasificación de imágenes sencilla - matlab & simulink - mathworks españa.

[3] Galaxy zoo - the galaxy challenge.

[4] Importación, procesamiento y exportación básicos de imágenes - matlab & simulink - mathworks españa.

[5] Introducción a la transferencia del aprendizaje - matlab & simulink - mathworks españa.

[6] Red neuronal convolucional resnet-50 - matlab resnet50 - mathworks españa.

[7] Run matlab functions on a gpu - matlab & simulink - mathworks españa.

[8] The softmax function and its derivative - eli bendersky's website.

[9] Mnist handwritten digit database, yann lecun, corinna cortes and chris burges, 2009.

[10] Tan Pengshi Alvin. Neural networks from scratch: Logistic regression — part 1, 04 2023.

[11] Tyler Elliot Bettilyon. How to classify mnist digits with different neural network architectures, 05 2019.

[12] Maribel Angélica Marín Castro. *Modelo Jerárquico para la clasificación de galaxias.* PhD thesis, Instituto Nacional de Astrofísica, Óptica y Electrónica, 2012.

[13] Paras Dahal. Classification and loss evaluation - softmax and cross entropy loss, 05 2017.

[14] Siddharth Hegde. Read digits and labels from mnist database.

[15] neuralthreads. Categorical cross-entropy loss — the most important loss function, 12 2021.

[16] Michael A Nielsen. Neural networks and deep learning, 2019.

[17] Grant Sanderson. Neural networks - youtube, 2019.

[18] Kyle W. Willett, Chris J. Lintott, Steven P. Bamford, Karen L. Masters, Brooke D. Simmons, Kevin R. V. Casteels, Edward M. Edmondson, Lucy F. Fortson, Sugata Kaviraj, William C. Keel, Thomas Melvin, Robert C. Nichol, M. Jordan Raddick, Kevin Schawinski, Robert J. Simpson, Ramin A. Skibba, Arfon M. Smith, and Daniel Thomas. Galaxy zoo 2: detailed morphological classifications for 304 122 galaxies from the sloan digital sky survey. *Monthly Notices of the Royal Astronomical Society*, 435:2835–2860, 11 2013.

[19] Vishal Yathish. Loss functions and their use in neural networks, 08 2022.

A. Anexo

app1

Este es el *script* que se utiliza para procesar imágenes utilizando redes neuronales. Está compuesto por distintos lienzos y botones que funcionan de la siguiente manera: primero seleccionas la red neuronal que vas a emplear a través de un selector; luego con el primer botón seleccionas la imagen a clasificar que aparecerá en el primer lienzo; por último seleccionas empezar para procesar la imagen y que empiece la clasificación. El resultado aparecerá en el cuadro de texto con la palabra etiqueta.

Layer

Este archivo es un *script* tipo clase que crea objetos. En este archivo creamos un "contenedor" para retener las neuronas, y la información de estas (sus parámetros, sesgo, peso y función de activación de la capa). Este *script* también calcula la derivada de la función de activación, y realiza el *eval* (5), la parte de multiplicación de matrices del *Feed Forward*.

massiveGalaxyModel

A través de este *script* logramos crear y entrenar una red neuronal. Para lograr esto leemos la tabla de datos de cada imagen, obtenida del *dataset*, luego convertimos los datos a una matriz y seleccionamos las columnas necesarias. Aplicamos un bucle *for* y creamos las categorías que implementaremos. Creamos las categorías y las almacenamos en un vector categórico.

A continuación pasamos a formato *imds* y dividimos las imágenes en dos, una parte para entrenar y otra para validar este entrenamiento. Después seleccionamos el tamaño y el número de categorías con el vector categórico anterior, y cargamos una red preentrenada. Modificamos las últimas capas de esta red para que sea capaz de clasificar. Y por último cambiamos los hiperparámetros, entrenamos la red y la guardamos.

matlabGalaxyNetImagePreprocessing

Usamos este *script* para generar una carpeta con las imágenes de las galaxias recortadas a 224 por 224, asegurándonos de recortar lo mismo por todos lados, para que la imagen siga centrada. Realizamos este proceso imagen a imagen en un bucle *for* y las guardamos en una nueva carpeta.

MNISTmodelReview

Esta es una comprobación de la calidad del modelo de clasificación de dígitos. Cargamos las imágenes y el modelo e inicializamos la matriz de confusión. Obtenemos así la cantidad de predicciones correctas y la tendencia a equivocarse en ciertos casos (y que casos son estos).

MNISTnet

El funcionamiento de esta red es bastante parecido a las anteriores, empezamos leyendo las entradas a través de una función externa que los pueda leer, transformamos las salidas para que podamos usarlas para categorizar, creamos una red con 512 neuronas en una capa oculta, especificamos hiperparámetros y entrenamos. Luego, graficamos el cambio de la pérdida en función del tiempo, observamos el resultado de pasar a una sola imagen e imprimimos la pérdida final. Por último, calculamos la precisión global de la red y la guardamos.

Model

Este *script* crea una red neuronal (conjunto de capas). Primero de todo definimos las propiedades globales, siendo estas *Layers* (el *array* de células) y *TrainLoss* (vector de pérdida a lo largo del entrenamiento). A continuación, creamos la función constructora de redes neuronales tomando un *array* de capas y pasándolo al *array celular*, después toma un *input* y lo pasa capa por capa hasta producir un *output*. Cogemos la matriz traspuesta del *input* y la pasamos capa por capa realizando los cálculos a través de la función *eval*.

Ahora aplicamos un *feed forward* a una porción del *training set* guardando a su vez las funciones de activación y los pasos intermedios de cada capa.

Con esto obtenemos una predicción, después realizamos el *backpropagation* para calcular el gradiente. Por último entrenamos la red calculando el gradiente en cada paso (siguiendo los pasos aquí descritos) y calculamos la pérdida con la función coste a través del *Categorical Cross Entropy*.

readMNIST

El funcionamiento de esta función es bastante simple, primero lee los dígitos y los categoriza, luego lee estas categorías, y por último calcula la media de los dígitos y la cuenta. Después cortamos los dígitos y normalizamos los datos. Esta función fue obtenida de [14].

simpleGalaxyModel

Este *script* es muy similar al *massiveGalaxyModel*. El proceso es el siguiente: leemos la tabla de datos, creamos categorías con un bucle *for* y las almacenamos en un vector categórico, seleccionamos

las imágenes y las dividimos en dos grupos (entrenamiento y validación), seleccionamos el tamaño de las imágenes, cargamos una red preentrenada y sustituimos la última capa para que sea capaz de categorizar, especificamos los hiperparámetros, entrenamos la red y la guardamos.

simpleGalaxyModelReview

A continuación comprobaremos la eficacia del *script* anterior cargando la red neuronal, cargando y clasificando las imágenes, calculando las etiquetas reales y generando una matriz de confusión con un bucle *for*.

xorNetLoss

Este *script* es un ejemplo de una red que emula la puerta XOR. Para empezar creamos los valores del *training set*, después con un bucle *for* creamos los vectores de datos X e Y. Creamos dos capas de neuronas (2 de entrada, 1 de salida) y creamos el modelo. Inicializamos los pesos y visualizamos la función de pérdida. Calculamos la precisión de la red sin entrenar, y posteriormente la entrenamos definiendo los hiperparámetros para iniciar el bucle de entrenamiento con la función *feedforward* entre otras. Representamos la gráfica de la función pérdida, calculamos la precisión de la red de entrada y representamos gráficamente la pérdida total de esta.