industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Desarrollo de un sistema de inteligencia artificial para el reconocimiento de calificaciones manuscritas en exámenes.

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA

Autor: Tomás León Morales
Director: Dr. Miguel Almonacid Kroeger
Codirector: Pablo Alejandro Martínez Ruiz

Cartagena, a 14 de marzo de 2021

Universidad Politécnica de Cartagena

Agradecimientos

Ahora que termina este largo trabajo y una etapa de mi vida, quiero dar las gracias a mi familia por haberme dado siempre todo su apoyo.

Agradecer a mis profesores de universidad por sus enseñanzas y dedicación a lo largo de mis años de estudio, en especial a mi director y codirector, Dr. Miguel Almonacid Kroeger y Pablo Alejandro Martínez Ruiz por haberme introducido en el campo de la visión por computador y despertar mi interés en este campo.

Además agradecer a mis amigos, Pedro, Francisco, Pablo Martínez, Pablo López y Daniel por ser un pilar fundamental en mi vida.

RESUMEN

El presente trabajo fin de estudios pretende servir al alumno como medio para llegar a la comprensión de los fundamentos teóricos de la inteligencia artificial y sus posibles aplicaciones. Sus principales objetivos son la obtención de conocimiento en este campo así como en el campo de visión por computador y, en última instancia, se pretende elaborar un sistema software basado en el uso de redes neuronales junto con técnicas de visión por computador para la digitalización de forma automática de calificaciones en pruebas evaluativas escritas. De esta forma se busca facilitar y reducir la carga de trabajo al personal docente, sobre todo de educación primaria, secundaria obligatoria y bachiller en la mayoría de lo posible, minimizando la interacción humana en el desarrollo de esta tarea común hoy en día.

Índice

Figuras.

Tablas.

1 Introducción.

Este capítulo sirve de introducción para este trabajo fin de grado. En él se expone la motivación de su realización, objetivos a cumplir para su desarrollo, el esquema general del sistema software elaborado y un resumen de la composición completa del trabajo.

1.1 Motivación.

Como estudiante de Ingeniería electrónica y automática, tengo como uno de mis objetivos reducir la carga de trabajo de las personas mediante la automatización de tareas y si es posible, la mejora de la eficiencia en su realización.

El personal docente, a todos los niveles, desenvuelve un papel crucial, casi diría que el de mayor importancia en una sociedad moderna. Las funciones que se exigen al personal docente comprenden un abanico mucho más amplio que simplemente impartir la docencia. Se extiende desde por ejemplo, la planificación del curso, reuniones de departamento o tutorías con los tutores de los alumnos, desarrollo de material docente, elaboración de pruebas de evaluación de los conocimientos impartidos y su corrección, etc. En la inmensa mayoría de los casos, ya sea por normativa o por requisitos del centro educativo, se le exige al docente la digitalización y almacenamiento de la información referente al alumno, incluidas las pruebas evaluativas y sus calificaciones. La tarea de digitalizar las calificaciones de los alumnos en su correspondiente, formato, hoja de cálculo y/o base de datos puede llegar a ser repetitiva y exasperante si se realiza a mano, como se da en la mayoría de los casos. Además, no requiere cualificación alguna y consume una cantidad considerable de tiempo, sobre todo cuando el docente está a cargo de varios grupos de alumnos como suele ser habitual. De modo que la finalidad última de este proyecto consiste en la automatización de esta tarea mediante técnicas de inteligencia artificial combinadas con visión por computador.

La tarea de digitalizar las calificaciones de los alumnos se ha vuelto aún más tediosa desde la reforma educativa en la que se cambió la forma de evaluación en forma de estándares. Esta forma de evaluación para la educación primaria, secundaria obligatoria y bachiller está recogida por el ministerio de educación en la que se evaluará la materia en forma de estándares, como se muestra un ejemplo en la Figura 1 , recogidos en el Real Decreto 126/2014, de 28 de febrero, por el que se establece el currículo básico de la Educación Primaria [1], y en el Real Decreto 1105/2014, de 26 de diciembre, por el que se establece el currículo básico de la Educación Secundaria Obligatoria y del Bachillerato [2].

Geología. 2º Bachillerato

Contenidos	Criterios evaluación	Estándares de aprendizaje evaluables
Bloque 1. El planeta tierra y su estudio		
Perspectiva general de la Geología, sus objetos de estudio, métodos de trabajo y su utilidad científica y social: Definición de Geología. El trabajo de los geólogos. Especialidades de la Geología. La metodología científica y la Geología. El tiempo geológico y los principios fundamentales de la Geología. La Tierra como planeta dinámico y en evolución. La Tectónica de Placas como teoría global de la Tierra. La evolución geológica de la Tierra en el marco del Sistema Solar. Geoplanetología. La Geología en la vida cotidiana. Problemas medioambientales y geológicos globales.	1. Definir la ciencia de la Geología y sus principales especialidades y comprender el trabajo realizado por los geólogos. 2. Aplicar las estrategias propias del trabajo científico en la resolución de problemas relacionados con la geología. 3. Entender el concepto de tiempo geológico y los principios fundamentales de la geología, como los de horizontalidad, superposición, actualismo y uniformismo. 4. Analizar el dinamismo terrestre explicado según la teoría global de la Tectónica de Placas. 5. Analizar la evolución geológica de la Luna y de otros planetas del Sistema Solar, comparándolas con la de la Tierra 6. Observar las manifestaciones de la Geología en el entorno diario e identificar algunas implicaciones en la economía, política, desarrollo sostenible y medio ambiente.	1.1. Comprende la importancia de la Geología en la sociedad y conoce y valora el trabajo de los geólogos en distintitos ámbitos sociales. 2.1. Selecciona información, analiza datos, formula preguntas pertinentes y busca respuestas para un pequeño proyecto relacionado con la geología. 3.1. Comprende el significado de tiempo geológico y utiliza principios fundamentales de la geología como: horizontalidad, superposición, actualismo y uniformismo. 4.1. Interpreta algunas manifestaciones del dinamismo terrestre como consecuencia de la Tectónica de Placas. 5.1. Analiza información geológica de la Luna y de otros planetas del Sistema Solar y la compara con la evolución geológica de la Tierra. 6.1. Identifica distintas manifestaciones de la Geología en el entorno diario, conociendo algunos de los usos y aplicaciones de esta ciencia en la economía, política, desarrollo sostenible y en la protección del medio ambiente.
Bloque 2. Minerales, los componentes de las rocas		
Materia mineral y concepto de mineral. Relación entre estructura cristalina, composición química y propiedades de los minerales. Clasificación químico-estructural de los minerales. Formación, evolución y transformación de los minerales. Estabilidad e inestabilidad mineral. Procesos geológicos formadores de minerales y rocas: procesos magmáticos, metamórficos, hidrotermales, supergénicos y sedimentarios	1. Describir las propiedades que caracterizan a la materia mineral. Comprender su variación como una función de la estructura y la composición química de los minerales. Reconocer la utilidad de los minerales por sus propiedades. 2. Conocer los grupos de minerales más importantes según una clasificación químico-estructural. Nombrar y distinguir de visu, diferentes especies minerales. 3. Analizar las distintas condiciones físico-químicas en la formación de los minerales. Comprender las causas de la evolución, inestabilidad y transformación mineral utilizando diagramas de fases sencillos. 4. Conocer los principales ambientes y procesos geológicos formadores de minerales y rocas. Identificar algunos minerales con su origen más común: magmático, metamórfico, hidrotermal, supergénico y sedimentario.	1.1. Identifica las características que determinan la materia mineral, por medio de actividades prácticas con ejemplos de minerales con propiedades contrastadas, relacionando la utilización de algunos minerales con sus propiedades. 2.1. Reconoce los diferentes grupos minerales, identificándolos por sus características físico-químicas. Reconoce por medio de una práctica *de visu* algunos de los minerales más comunes. 3.1. Compara las situaciones en las que se originan los minerales, elaborando tablas según sus condiciones físico-químicas de estabilidad. Conoce algunos ejemplos de evolución y transformación mineral por medio de diagramas de fases. 4.1. Compara los diferentes ambientes y procesos geológicos en los que se forman los minerales y las rocas. Identifica algunos minerales como característicos de cada uno de los procesos geológicos de formación.

Figura 1. Estándares de Geología de 2°Bachiller.

Como se puede observar en Figura 1, cada bloque de una materia tendrá asignado un número de estándares que debe ser evaluado por el docente. La forma de evaluación de estos estándares puede ser libre o venir especificada según la normativa. Suele ser común evaluar varios estándares en una prueba escrita, reduciendo así la carga de trabajo del docente, de modo que mientras que anteriormente se tenía una sola calificación por bloques, tendremos tantas calificaciones como estándares se evalúen en cada prueba escrita.

1.2 Objetivos.

El objetivo final del proyecto consiste en la elaboración de un sistema de digitalización de calificaciones en exámenes a una hoja de cálculo, eliminando la mayor interacción humana posible. Se deberá proporcionar un documento de tipo PDF con los exámenes correspondientes a un grupo y bloque para su correspondiente digitalización. Para lograr este objetivo será necesario cumplir con una serie de subobjetivos descritos a continuación:

- Adquirir los conocimientos referentes a la inteligencia artificial, concretamente a la rama de redes neuronales. Entender el principio teórico de funcionamiento de la red neuronal artificial, métodos de aprendizaje y topología.
- Adquirir los conocimientos referentes a técnicas de visión por computador, tratamiento y adquisición de imágenes. Técnicas de procesamiento de imágenes, filtrado, binarización, transformaciones morfológicas, algoritmos de detección y clasificación en imágenes.
- Estudio de la red neuronal artificial aplicada junto con una optimización y análisis de los hyperparametros de la red, apoyado en una serie de experimentos.
- Elaboración de un algoritmo de detección de las calificaciones y su estándar correspondiente. Detección de celdas y segmentación de calificaciones en sus dígitos correspondientes, así como de signos de puntuación de puntos y comas.
- Digitalización y exportación de los resultados obtenidos por la red a una hoja de cálculo.

1.3 Estructura del sistema software.

En la Figura 2 se muestra el esquema general del sistema diseñado. Durante el desarrollo del sistema software se decidió que tomará como entrada un documento tipo PDF con el scan de un grupo de alumnos. En primer lugar se convertirá cada página del documento PDF a un formato tipo JPG. Mediante técnicas de visión artificial se detectarán las calificaciones de cada examen y se le pasará a una red neuronal programada y entrenada como entrada las imágenes correspondientes a posibles dígitos dentro de las celdas para que se lleve a cabo su clasificación. Una vez clasificados todos los dígitos de cada celda se elabora, se agrupan según la celda a la que pertenezcan y se elabora una tabla a exportar a un documento tipo hoja de cálculo con todas las calificaciones del grupo.

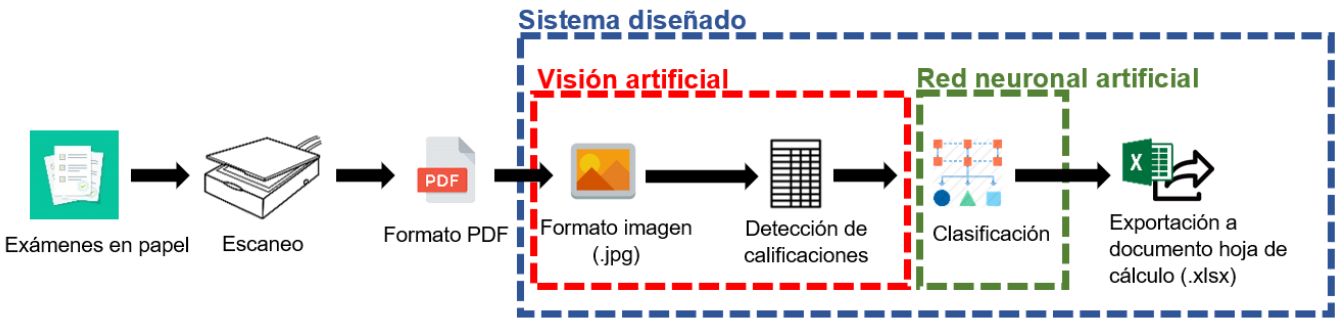


Figura 2. Estructura global del sistema software diseñado.

1.4 Resumen de capítulos.

El presente trabajo fin de estudios cuenta con un total de 5 capítulos en los que se divide el mismo.

El primer capítulo sirve de introducción al trabajo y en él se exponen los distintos objetivos a lograr.

En el segundo capítulo se exponen distintos conocimientos relacionados con el trabajo, requerimientos y datos de interés.

En el tercer capítulo se desempeña el desarrollo teórico y práctico del sistema software a realizar para cumplir con los objetivos del proyecto.

En el cuarto capítulo se llevan a cabo pruebas con dicho sistema y se analizan los resultados obtenidos.

En el quinto capítulo se exponen las conclusiones finales del trabajo y posibles vías de trabajo futuras.

2 Redes neuronales y entorno de desarrollo.

En este capítulo se describen breves conceptos claves sobre redes neuronales artificiales (RNA). También se comenta sobre el conjunto de datos de entrenamiento usado para el ajuste de una red neuronal artificial, así como el entorno de programación en el que se ha desarrollado el trabajo, lenguaje de programación y bibliotecas o módulos usados.

2.1 Historia y estado del arte.

Se podría considerar que unos de los primeros pasos hacia la IA fueron dados hace mucho tiempo por Aristóteles (384-322 a.C.), cuando se dispuso a explicar y codificar ciertos estilos de razonamiento deductivo que él llamó silogismos. Otro intento sería el de Ramón Llull (d.C. 1235-1316), místico y poeta catalán, quien construyó un conjunto de ruedas llamado Ars Magna, el cual se suponía iba a ser una máquina capaz de responder todas las preguntas [3].

Por su parte, Martin Gardner atribuye a Gottfried Leibniz (1646-1716) el sueño de "un álgebra universal por el cual todos los conocimientos, incluyendo las verdades morales y metafísicas, pueden algún día ser interpuestos dentro de un sistema deductivo único". Sin embargo, no existió un progreso sustancial hasta que George Boole comenzó a desarrollar los fundamentos de la lógica proposicional. Poco después, Gottlob Frege propuso un sistema de denotación para el razonamiento mecánico y al hacerlo inventó gran parte de lo que hoy conocemos como cálculo proposicional (lógica matemática moderna). Fue posteriormente, en 1958, que John McCarthy introdujo el término de "inteligencia artificial". La mayoría de los autores responsables del tema dividen la ciencia de la inteligencia artificial en 3 ramas:

- Lógica difusa: Permite a una computadora analizar información del mundo real en una escala entre lo falso y verdadero.
- Redes Neuronales Artificiales: Tratan de reproducir el proceso de solución de problemas del cerebro mediante la elaboración de modelos matemáticos complejos en forma de red.
- Algoritmos Genéticos: Técnica de búsqueda iterativa inspirada en los principios de selección natural. Los algoritmos genéticos no buscan modelar la evolución biológica sino derivar estrategias de optimización. El concepto se basa en la generación de poblaciones de individuos mediante la reproducción de los padres.

La ciencia de la inteligencia artificial posee logros realmente increíbles. En 1997, Deep Blue, una inteligencia artificial creada por IBM derroto al campeón del mundo de ajedrez del momento, el ruso Gary Kaspárov. Ha día

de hoy existen grandes inteligencias artificiales que alcanzan ratings de 3400 como Alpha Zero o Stockfish, mientras tanto, el actual campeón de ajedrez, el noruego Magnus Carlsen tiene alrededor de 2900. Además se tiene una situación similar en otros juegos como el go o el shogi, cuyos campeones mundiales también han sido derrotados.

En la actualidad, el reconocimiento preciso en textos escritos a máquina se considera un problema resuelto. Sin embargo, no ocurre lo mismo con el reconocimiento de la impresión manual, es decir, aquella que proviene de la caligrafía humana, la cual sigue siendo una fuente de intensa investigación [4]. Para el reconocimiento de textos escritos a máquina resulta simple la clasificación de sus caracteres con sistemas de reconocimiento óptico de caracteres (OCR), en los que tras la extracción de los caracteres individuales se comparan con una base de datos con distantes fuentes y tamaños de caracteres.

Esto no es posible para caracteres manuscritos que varían según la tipografía o el elemento de escritura. El problema del reconocimiento de caracteres manuscritos está ampliamente estudiado y desarrollo. Para su resolución se hacen uso de distintas técnicas como clasificadores linear, máquinas de soporte vectorial (SVM), sistemas de reconocimiento óptico de caracteres (SVM), Redes neuronales profundas, convolucionales o de tipo feedforward, métodos de clasificación no paramétricos como k-nearest neighbors (k-NN), y otras muchas técnicas de regresión y clasificación.

Otra técnicas propuestas están basadas es la elaboración de un algoritmo basado en reglas bien definidas para la clasificación de dígitos numéricos como por ejemplo la relación de pixeles entre distintos segmentos de una imagen, permitiendo obviar una etapa de entrenamiento [5], aunque sus resultados distan bastante de los que se pueden lograr con redes neuronales artificiales.

2.2 Modelos de neurona artificial.

A lo largo de los años desde el nacimiento de la ciencia conocida como inteligencia artificial en los años 50, se han propuesto varios modelos conceptuales discretos que poseen distintos tipos de inteligencia. Un modelo muy conocido, ampliamente estudiado y en desarrollo a día de hoy consiste en redes neuronales artificiales, cuyo propósito es imitar o simular la estructura cerebral de seres vivos complejos. Estos modelos consisten en capas compuestas por elementos básicos llamados neuronas artificiales, interconectadas entre sí mediante distintos tipos de topología que interaccionan entre sí formando una red de información. Se han propuesto gran cantidad de modelos representativos de neuronas artificiales. Los 2 más famosos y de mayor interés se exponen a continuación:

2.2.1 Perceptrón.

Es un tipo de neurona artificial de categoría lineal. “En 1957, Frank Rosenblatt publicó el mayor trabajo de investigación en computación neuronal realizado hasta esas fechas”. Su trabajo consistía en el desarrollo de un elemento llamado "Perceptrón" [3], como se puede ver en la Figura 3. Frank Rosenblat se inspiró en los trabajos previos de Warren McCulloch y Walter Pitts, razón por la cual al perceptrón también se le conoce como modelo de neurona McCulloch-Pitts. En el modelo de neurona del perceptrón, se toma como entrada un vector de valores que denominaremos con la letra ‘x’. Cada valor del vector tiene asociado lo que Rosenblatt denomino “peso” (weight en inglés), cuyo conjunto se denomina con la letra ‘w’ de forma que la salida de la neurona artificial podrá tomar los valores binarios de 0 o 1, según si la suma de los productos de las entradas por sus pesos asociados es mayor o no que un umbral o “bias” que se denomina con la letra ‘b’ [6] [7], es decir, tendrá como función de activación la función escalón. Poniéndolo de forma algebraica sería:

$$output = \begin{cases} 0\ si\ \sum x \cdot w \leq b \\ 1\ si\ \sum x \cdot w > b \end{cases} \quad (1)$$

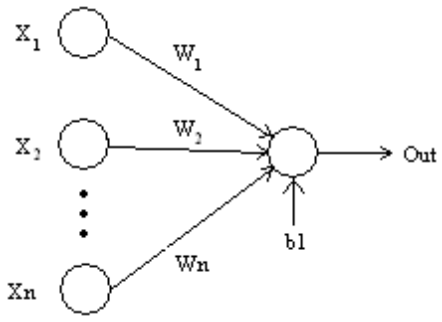


Figura 3. Representación gráfica del Perceptrón.

2.2.2 Neurona sigmoide.

En nuestro el presente trabajo no se usará el modelo de perceptrón como modelo de neurona sino un tipo más adecuado para nuestra aplicación, el modelo de neurona sigmoide. Usualmente, se somete la salida de una neurona a lo que se conoce como ‘función de activación’. De hecho, es así también para el perceptrón, cuya función de activación es una función escalón. Una vez dicho esto se puede expresar de forma algebraica la salida de una neurona como: $a = f(x, w, b)$ donde ‘a’ será la salida de la neurona y f(·) su función de activación. Se han propuesto muchas funciones de activación a lo largo de los años. Las más conocidas son la función sigmoide

(σ)[1], en la Figura 4, y la ReLu (R). Además de ser la más antigua y popular, se comporta muy bien a la hora de trabajar con ella, como se verá en apartados posteriores.

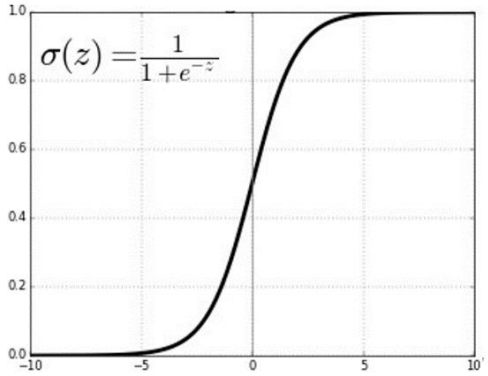


Figura 4. Función sigmoide.

Mediante el uso de este tipo de función de activación se puede acotar la salida de cada neurona a un valor entre 0 y 1. Con este cambio de comportamiento significativo con respecto al perceptrón podemos expresar la variación en la salida de una red compuesta de este tipo de neurona de la siguiente manera:

$$\Delta \text{output} \approx \frac{\partial \text{output}}{\partial w_j} \Delta w_j + \frac{\partial \text{output}}{\partial b_j} \Delta b_j \qquad (2)$$

Mediante esta ecuación se puede afirmar que la variación de la salida es una función lineal que depende de cada uno de los pesos y biases de la red.

2.3 Clasificación de redes neuronales artificiales.

Existen muchas clasificaciones de las redes neuronales gracias a la multitud de aspectos en que se pueden dividir. Los más interesantes y los que más caracterizan a una red neuronal son:

2.3.1 Topología.

Comenzando con un poco de terminología, se denomina 'capa de entrada' a la primera capa de la red, por la cual recibe los valores de entrada, 'capa de salida' a la capa correspondiente a la salida de la red y 'capas ocultas' a aquellas capas que se encuentren en el interior de la red, como se puede observar en la Figura 5. Hay que destacar que la capa de entrada no posee neuronas como tal, pues no tienen un bias asociado y no se les aplica ninguna función de activación, son única y exclusiva mente el vector de entrada de la red.

[1] Generalmente llamada como función logística.

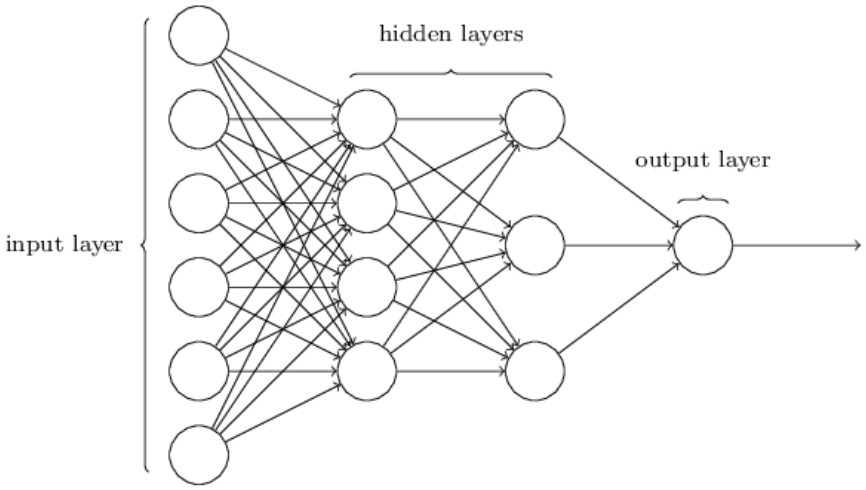


Figura 5. Ejemplo de RNA.

Al hacer la clasificación de las redes neuronales artificiales según el número de capas que posee se distinguen entre:

- Redes monocapa: en las que se establecen conexiones laterales entre las neuronas que pertenecen a la única capa que constituye la red. Ejemplos de redes de este tipo son la red HOPPFIELD o la red BRAIN-STATE-IN-A-BOX. Se utilizan típicamente en tareas relacionadas con lo que se conoce como auto asociación; por ejemplo, para regenerar informaciones de entrada que se presenta como incompleta o distorsionada.
- Redes multicapa: disponen las neuronas agrupadas en varias capas. Dado que este tipo de redes disponen de varias capas, las conexiones entre neuronas pueden ser del tipo feedforward (conexión hacia adelante) o del tipo feedback (conexión hacia atrás).

Haciendo referencia al tipo de conexión entre neuronas se distinguen entre:

- Redes unidireccionales (feedforward): en las que la información circula en un único sentido, desde las neuronas de entrada hacia las de salida.
- Redes recurrentes o realimentadas (convolutional): la información puede circular entre las capas en cualquier sentido.

2.3.2 Método de entrenamiento de la RNA.

Una de las principales características de las RNA es su capacidad de aprendizaje. El objetivo del entrenamiento de una RNA es conseguir que una aplicación determinada, para un conjunto de entradas, produzca el conjunto de salidas deseadas o mínimamente consistentes. El proceso de entrenamiento consiste en la aplicación secuencial de diferentes conjuntos o vectores de entrada para que se ajusten los pesos de las interconexiones y biases de las neuronas artificiales según un procedimiento predeterminado denominado algoritmo de

aprendizaje. Durante la sesión de entrenamiento los pesos y biases convergen gradualmente hacia los valores que hacen que cada entrada produzca el vector de salida deseado. Los algoritmos de entrenamiento o los procedimientos de ajuste de los valores de las conexiones de las RNA se pueden clasificar en dos grupos: Supervisado y No Supervisado.

En redes con entrenamiento supervisado se suministra junto con el conjunto de datos de entrenamiento la salida deseada. Los sistemas no supervisados son modelos de aprendizaje más lógicos y parecidos a los sistemas biológicos. Desarrollados por Kohonen (1984) y otros investigadores, estos sistemas de aprendizaje no supervisado no requieren de un vector de salidas deseadas y por tanto no se realizan comparaciones entre las salidas reales y salidas esperadas.

2.4 MNIST Database.

La base de datos modificada del Instituto Nacional de Estándares y Tecnología (MNIST) es una gran base de datos usada comúnmente para entrenamiento de redes neuronales artificiales junto con sistemas de procesamiento de imagen y visión por computador. Se ha decidido hacer uso de esta base de datos para el entrenamiento de la red neuronal a desarrollar por la calidad del conjunto de entrenamiento y por ampliamente conocida y utilizada. Posee 60000 muestras de entrenamiento, y un conjunto de 10000 muestras de prueba. El conjunto de muestras de entrenamiento fue escrito por un total de 250 personas aproximadamente, la mitad de los cuales eran empleados de la Oficina del Censo de EE. UU. y la otra mitad eran estudiantes de secundaria. El MNIST es parte de una base de datos aún mayor disponible en el NIST (Instituto Nacional de estadística y estandarización). Puede verse un ejemplo en la Figura 6.

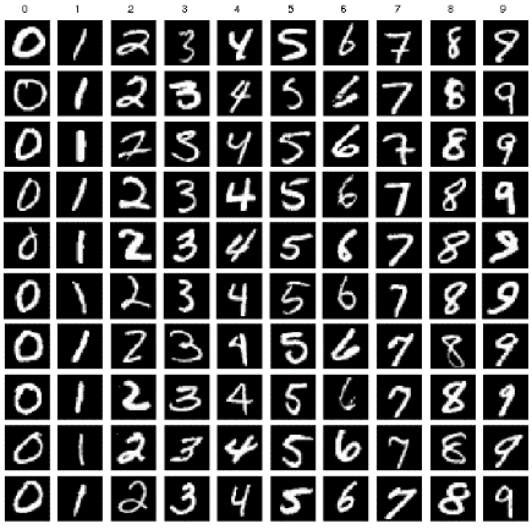


Figura 6. Ejemplo muestras del MNIST invertido.

El conjunto de entrenamiento MNIST está compuesto por 30.000 patrones de SD-3 y 30.000 patrones de SD-1. El conjunto de prueba está compuesto por 5,000 patrones de SD-3 y 5,000 patrones de SD-1. SD-1 contiene 58.527 imágenes de dígitos escritas por 500 escritores diferentes. A diferencia de SD-3, donde los bloques de datos de cada escritor están secuenciados, los datos de SD-1 están barajados de forma aleatoria.

El tamaño de las imágenes originales en blanco y negro del NIST esta normalizado para que quepan en un cuadro de 20x20 píxeles conservando su relación de aspecto. Las imágenes resultantes contienen niveles de gris como resultado de técnicas anti-aliasing utilizada por el algoritmo de normalización. Las imágenes están centradas en una imagen de 28x28 pixeles calculando el centro de masa de los píxeles y traduciendo la imagen para colocar este punto en el centro del campo de 28x28 [8].

2.4.1 Formato.

Los datos se almacenan en un formato de archivo muy simple diseñado para almacenar vectores y matrices multidimensionales, como se puede ver en la Figura 7. Todos los números enteros de los archivos se almacenan en el formato MSB-first utilizado por la mayoría de los procesadores que no son de Intel. Los píxeles están organizados por filas. Los valores de píxeles son de 0 a 255. 0 significa fondo (blanco), 255 significa primer plano (negro).

Hay 4 archivos:

- train-images-idx3-ubyte: training set images
- train-labels-idx1-ubyte: training set labels
- t10k-images-idx3-ubyte: test set images
- t10k-labels-idx1-ubyte: test set labels

TRAINING SET LABEL FILE (train-labels-idx1-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000801(2049)	magic number (MSB first)
0004	32 bit integer	60000	number of items
0008	unsigned byte	??	label
0009	unsigned byte	??	label
....			
xxxx	unsigned byte	??	label

The labels values are 0 to 9.

TRAINING SET IMAGE FILE (train-images-idx3-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	60000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
....			
xxxx	unsigned byte	??	pixel

Figura 7. Formato del conjunto de entrenamiento del MNIST.

2.5 Entorno de programación, lenguaje y módulos.

A la hora del desarrollo de un sistema software es imprescindible contar con un entorno de desarrollo adecuado para ello. Para el desarrollo de este proyecto se ha utiliza Visual Studio, un entorno de desarrollo integrado desarrollado por Microsoft en 2017 para Windows y macOS. Es compatible con múltiples lenguajes de programación, tales como C++, C#, Visual Basic .NET, F#, Java, Python, Ruby y PHP.

Como lenguaje de programación se ha usado únicamente Python 3, concretamente la versión 3.6 de 64 bits. Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código. Se trata de un lenguaje de programación multiparadigma, ya que soporta parcialmente la orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma muy utilizado en el desarrollo de aplicaciones relacionadas con análisis de datos o inteligencia artificial.

Existen muchas propiedades que se pueden agregar al lenguaje importando módulos. La instalación de módulos en Python se puede realizar mediante el entorno de desarrollo. A continuación se exponen los principales módulos usados para el desarrollo del proyecto junto con una breve descripción:

- matplotlib 3.3.0: Para la elaboración de gráficas y representación de conjunto de datos.
- numpy 1.19.1: Ofrece soporte para crear vectores y matrices grandes multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel para operar con ellas.
- scipy 1.5.2: Ofrece herramientas y algoritmos matemáticos.
- PyPDF2 1.26.0: Toolkit para archivos PDF.
- Tkinter: Biblioteca gráfica.
- OpenCV: Biblioteca de procesamiento de imágenes y visión por computador.
- openpyxl 3.0.7: Toolkit para archivos XLSX/XLSM/XLTX/XLTM.

3 Diseño del sistema software.

A lo largo de este capítulo se diseña el sistema software, núcleo de este proyecto fin de grado. Se expone el fundamento teórico de funcionamiento de las redes neuronales artificiales y métodos de aprendizaje. Se lleva a cabo la programación de la red neuronal, la determinación de sus hyperparametros mediante técnicas heurísticas y experimentales y se implementan distintas mejoras a una red neuronal artificial básica. Posteriormente se habla de las técnicas de visión por computador para la extracción de la información relevante de una prueba evaluativa común y su programación en el entorno de desarrollo.

3.1 Red neuronal. Fundamento teórico y método de aprendizaje.

3.1.1 Arquitectura.

Para el diseño de la red neuronal el primer paso a tomar es decidir la topología de la red. El problema de clasificación de objetos en la ciencia de inteligencia artificial ha sido estudiado por muchos científicos y autores desde hace más de 50 años, especialmente la clasificación de dígitos numéricos arábigos. En base a estas investigaciones se ha decidido usar una red neuronal feed-forward con entrenamiento supervisado. Como capa de entrada se ha escogido una de tamaño 784. Esto es debido a que se toma como entrada a la red el nivel de gris de cada pixel de una imagen de tamaño 28x28, pues este es el tamaño en el que se tiene el conjunto de datos de entrenamiento mencionado en el apartado 2.4 MNIST Database. Como el propósito de la red es clasificar los dígitos del 0-9, resulta intuitivo escoger como capa de salida de tamaño 10, donde cada neurona tendrá como salida un valor numérico basado en el funcionamiento de la propia red, tomando como salida aquella neurona cuya salida sea la máxima de la capa de salida. Sin embargo, existen muchas otras opciones como por ejemplo una capa de salida con 4 neuronas donde la salida estaría codificada en binario, todo depende de la interpretación que se le dé posteriormente a la salida. Aun así, se ha optado por una capa de salida de tamaño 10 por su simplicidad. Puesto que la salida de la red será una capa de salida de tamaño 10, donde la salida de cada neurona está asociada a uno de los 10 dígitos es usual aplicar una función Softmax para conocer la probabilidad de que la entrada de la red pertenezca a uno u otro dígito. Se defina la función Softmax como:

$$f(z)_j = \frac{e^{z_j}}{\sum_j^K e^{z_k}} para\ j = 1, \dots, K. \quad (3)$$

3.1.2 Gradiente descendente estocástico.

El entrenamiento de la red se realizará mediante técnicas de entrenamiento supervisado. Se denotará la salida deseada de la red correspondiente por: $y = y(x)$, donde y es un vector de 10 dimensiones. Por ejemplo, si una imagen de entrenamiento particular, x, representa un 6, entonces y (x) = (0,0,0,0,0,0,1,0,0,0)^T es la salida deseada de la red .

Una vez que tenemos un conjunto de muestras de entrenamiento con una salida deseada asociada, tenemos que determinar una forma de evaluar como de 'bien' se está comportando la red neuronal artificial. Esto se hace mediante lo que se denomina 'función de coste'. En este caso se definirá la función de coste como:

$$C(w,b) \equiv \frac{1}{2n}\sum_{n}\|y(x)-a\|^2 \qquad (4)$$

donde 'w' es el conjunto de todos los pesos de la red, 'b' es el conjunto de biases de la red, 'n' el número total de ejemplos de entrenamiento, 'a' es el vector de salida de la red para una entrada 'x', siendo también función de 'w' y 'b'. A la función de coste representada en la ecuación (4) también se la conoce como 'función de coste cuadrática', 'error medio cuadrado' o 'mean squared error' en inglés (MSE).

Dos propiedades fundamentales que debe tener toda función de coste son:

- Debe ser positiva para cada muestra de entrenamiento 'x'.
- Debe tender a 0 cuando la salida de la red se aproxima a la deseada para la muestra de entrenamiento.

Una vez determinada una función de coste, se busca encontrar el conjunto de pesos y biases que la reducen, es decir, encontrar el mínimo de la función. Se logra esto mediante el uso del algoritmo 'Gradiente descendente' [9]. Una forma de atacar el problema es utilizar el cálculo para tratar de encontrar el mínimo de esta función de forma analítica. Se puede calcular derivadas y luego intentar usarlas para encontrar lugares donde C es un extremo. Sin embargo, esto no es posible cuando la función depende de muchas variables como es nuestro caso. Para hacerse una idea de la magnitud y complejidad del problema, la mínima red que se podría construir sería una red sin capas ocultas, es decir, una red de [784, 10]. Esta red tendría un total de 10 biases y un total de 784 x 10 pesos, esto haría un total de 7850 parámetros o variables de los que depende la función.

Se puede suponer que la función de coste se asemejará en cierto modo a la topología de un valle. Partiendo de un punto aleatorio, simulando el movimiento que tendría una pelota (imaginaria) que desciende a lo largo del valle, al final acabará en un mínimo de la función. Se puede simular este comportamiento haciendo uso del cálculo, como se ve en la Figura 8.

Moviendo la pelota, tomando como ejemplo dos variables v_1 y v_2, una pequeña cantidad Δv_1 en la dirección v_1y una pequeña cantidad Δv_2 en la dirección v_2, el cálculo dicta que C cambia según:

$$\Delta C \approx \frac{\partial \mathrm{C}}{\partial v_1}\Delta v_1 + \frac{\partial \mathrm{C}}{\partial v_2}\Delta v_2 \tag{5}$$

Definiendo: $\nabla C \equiv (\frac{\partial \mathrm{C}}{\partial v_1}, \frac{\partial \mathrm{C}}{\partial v_2})^T$ podemos sobrescribir $la\ ecuacion\ (5)$ como:

$$\Delta C \approx \nabla C \cdot \Delta v \tag{6}$$

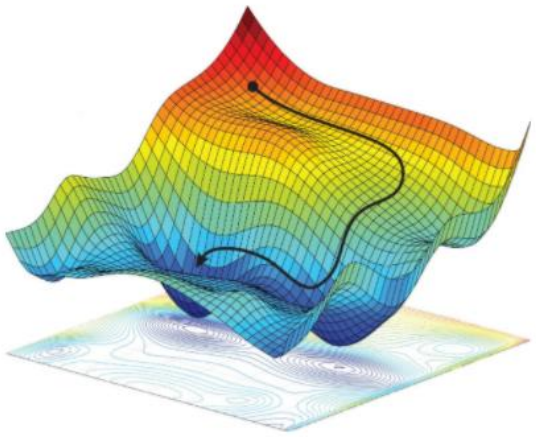


Figura 8. Representación gráfica del gradiente descendente de una función [10].

Como el objetivo es minimizar C, buscaremos hacer ΔC negativo, de modo que:

$$\Delta v = -\eta \nabla C \tag{7}$$

donde 'η' es como se denomina al 'ratio de aprendizaje'. Este se puede interpretar como, 'como de rápido aprenderá la red neuronal artificial', o 'cuanto desciende la pelota por el valle cada vez que se itera en el algoritmo'.

Juntando las ecuaciones (6) y (7) obtenemos que:

$$\Delta C \approx -\eta \|\nabla C\|^2 \tag{8}$$

$$v \to v' = v - \eta \nabla C \tag{9}$$

donde v' será la nueva posición de la pelota. Ya que $\|\nabla C\|^2$ será siempre ≥ 0, nos aseguramos de que $\Delta C \leq 0$. Actualizando la nueva posición de la pelota, como se ve en la ecuación (9), en la función y repitiendo este proceso, podemos alcanzar un mínimo de la función C. Para el algoritmo de gradiente descendente funcione

correctamente, es necesario escoger un ratio de aprendizaje pequeño, así la ecuación (7) es una aproximación valida.

Científicos han investigado muchas variaciones del gradiente descendente, incluidas variaciones que imitan más de cerca una pelota física real. Estas variaciones que imitan una pelota tienen algunas ventajas, pero también tienen una gran desventaja: resulta necesario calcular segundas derivadas parciales de C, y esto puede resultar bastante costoso y complicado, recordando que el orden de variables de las que dependerá C será de como mínimo 8000.

Sustituyendo nuestras variables, v_1 y v_2 por nuestro conjunto de pesos y biases, w_j y b_j:

$$w_j \to w'_j = w_j - \eta \frac{\partial C}{\partial w_j} \tag{10}$$

$$b_j \to b'_j = b_j - \eta \frac{\partial C}{\partial b_j} \tag{11}$$

Recordando, la función de coste C, vista en la ecuación (4), computa individualmente para cada uno de los ejemplos de entrenamiento y luego obtenemos una media. Sin embargo, al estar tratando con un elevado número de ejemplos de entrenamiento los tiempos de computación y costes serán muy elevados. Para solucionar este problema surge la idea del 'gradiente descendente estocástico' o 'stochastic gradient descent' en inglés (SGD) [11].

El SGD consiste en aplicar el algoritmo del gradiente descendente solo a una pequeña parte de las muestras de entrenamiento, seleccionadas de manera aleatoria, y utilizando una media local. De esta manera aceleramos el proceso de aprendizaje de la red. A cada pequeña muestra de entrenamiento mencionada anteriormente se le suele denominar 'mini-batch'. De modo que utilizando el método de aprendizaje de gradiente descendente estocástico, reescribimos las ecuaciones (10) y (11) como:

$$w_j \to w'_j = w_j - \frac{\eta}{m} \sum_j \frac{\partial C_{x_k}}{\partial w_j} \tag{12}$$

$$b_j \to b'_j = b_j - \frac{\eta}{m} \sum_j \frac{\partial C_{x_k}}{\partial b_j} \tag{13}$$

donde se denomina como 'X' al mini-batch y 'm' el tamaño del mini-batch. Se han mencionado varios parámetros pertenecientes a la red a parte del conjunto de pesos y biases, estos son el ratio de aprendizaje y el tamaño del mini-batch. A estos parámetros se les suele llamar hyperparametros de la red.

3.1.3 Retro propagación.

Una de las partes vitales del algoritmo SGD es que se tiene que computar la derivada parcial de la función de coste. Para describir en detalle cómo realizar esta derivada se hará uso de la siguiente notación:

- Se usará w_{jk}^{l} al peso que conecta la neurona 'k' de la capa $(l-1)$ con la neurona 'j' de la capa $'l'$ de un total de 'L' capas.
- Se usará b_j^l al bias de la neurona 'j' de la capa 'l' de un total de 'L' capas.

Con el uso de esta notación la activación a_j^l corresponde a la salida de la neurona 'j' de la capa 'l' y vendrá relacionada con la capa $(l-1)$ mediante la siguiente expresión:

$$a_j^l = \sigma(\sum_k w_{jk}^l a_j^{l-1} + b_j^l) = \sigma(\sum_k z_j^l) \tag{14}$$

que de forma matricial se puede escribir como:

$$a^l = \sigma(w^l a^{l-1} + b^l) = \sigma(z^l) \tag{15}$$

Para computar las derivadas parciales $\frac{\partial C}{\partial w_j}$ y $\frac{\partial C}{\partial b_j}$ se introduce un parámetro intermedio δ_j^l que será el error cometido en la neurona 'j' de la capa 'l'. Debido a la arquitectura de la red por capas, aparte de la propagación de la respuesta de cada neurona, también se propaga su error asociado. Sin embargo, la salida de la red corresponde únicamente a la última capa de la red, por lo tanto se necesita propagar el error cometido en la última capa en sentido inverso por la red hasta llegar a la capa de entrada.

Se define el error δ_j^l de una neurona como:

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l} \tag{16}$$

Teniendo en cuanta la definición del error cometido para cualquier neurona en la red, se puede definir el error en la capa de salida de la red neuronal artificial como:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L) \tag{17}$$

donde σ' es la derivada de la función sigmoide . Como se puede apreciar en la ecuación (17), en el caso de estar usando la función de coste cuadrática (ecuación (4)), $\partial C / \partial a_j^L = (a_j^L - y_l)$, la expresión que resulta es fácilmente computable, al igual que $\sigma'(z_j^L)$.

Escribiendo la ecuación (17) de forma matricial:

$$\delta^L = \nabla_a C \odot \sigma'(z_j^L) \tag{18}$$

donde $\odot$ es el producto 'hadamard', también conocido como 'puerta hadamard'.

También podemos definir el error δ^l en función del error de la siguiente capa δ^{l+1}:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{19}$$

Con el uso de esta expresión podemos propagar el error cometido en una capa 'l' de la red conocido su error en la siguiente capa '$(l+1)$', es decir, estamos propagando el error hacia atrás en la red.

Por último podemos definir también como afectará el cambio en el conjunto de pesos y biases de la red neuronal artificial al error producido en una capa 'l', δ^l:

$$\frac{\partial \mathrm{C}}{\partial b_j^l} = \delta^l \tag{20}$$

$$\frac{\partial \mathrm{C}}{\partial w_j^l} = a_j^{l-1} \delta^l \tag{21}$$

Para el caso de la ecuación (20) el cambio en el conjunto de biases 'b' será iguala a, precisamente, el error cometido. Y como ya habíamos visto en las ecuaciones (18) y (19) es fácilmente computable.

Para le caso de la ecuación (21), aunque resulta también fácilmente computable, como ya se ha visto en la ecuación (15), nos indica que cuando $a_j^{l-1} \approx 0$ el cambio en los pesos también será ≈ 0. Esto indica que los pesos aprenden más lentamente y puede llegar a ser un problema.

Otra conclusiones que podemos sacar de las ecuaciones (18), (19), (20) y (21) es que, debido a la forma de la función sigmoide σ que se aplana en los extremos, el aprendizaje también será lento cuando las salidas de las neuronas estén próximos a 1 o a 0, esto es, la neurona esta 'saturada' [12] [13].

Una vez comprendido el principio de funcionamiento de la red neuronal artificial feedforward y el método de aprendizaje por gradiente descendente estocástico describimos el algoritmo de aprendizaje de la red como:

1. Introducimos las muestras de entrenamiento a la red.
2. Propagación hacia delante en la red.
3. Determinación del error δ_j^L.
4. Retro propagación del error a través de la red para cada capa de la red.

5. Cálculo de $\frac{\partial C}{\partial b_j^l}$ y $\frac{\partial C}{\partial w_j^l}$.

Al número de veces que se ejecutan los algoritmos de feedforward y backpropagation se le denomina número 'epoch', otro hyperparametros de la red. En cada ciclo (epoch) todos los datos de entrenamiento pasan por la red neuronal para que esta aprenda sobre ellos, si existen 60000 datos (como es el caso) y 10 epochs, cada ciclo los 60000 datos pasaran por la red neuronal. De este modo, cuanto mayor sea el número de ciclos más nos acercaremos al mínimo de la función de coste.

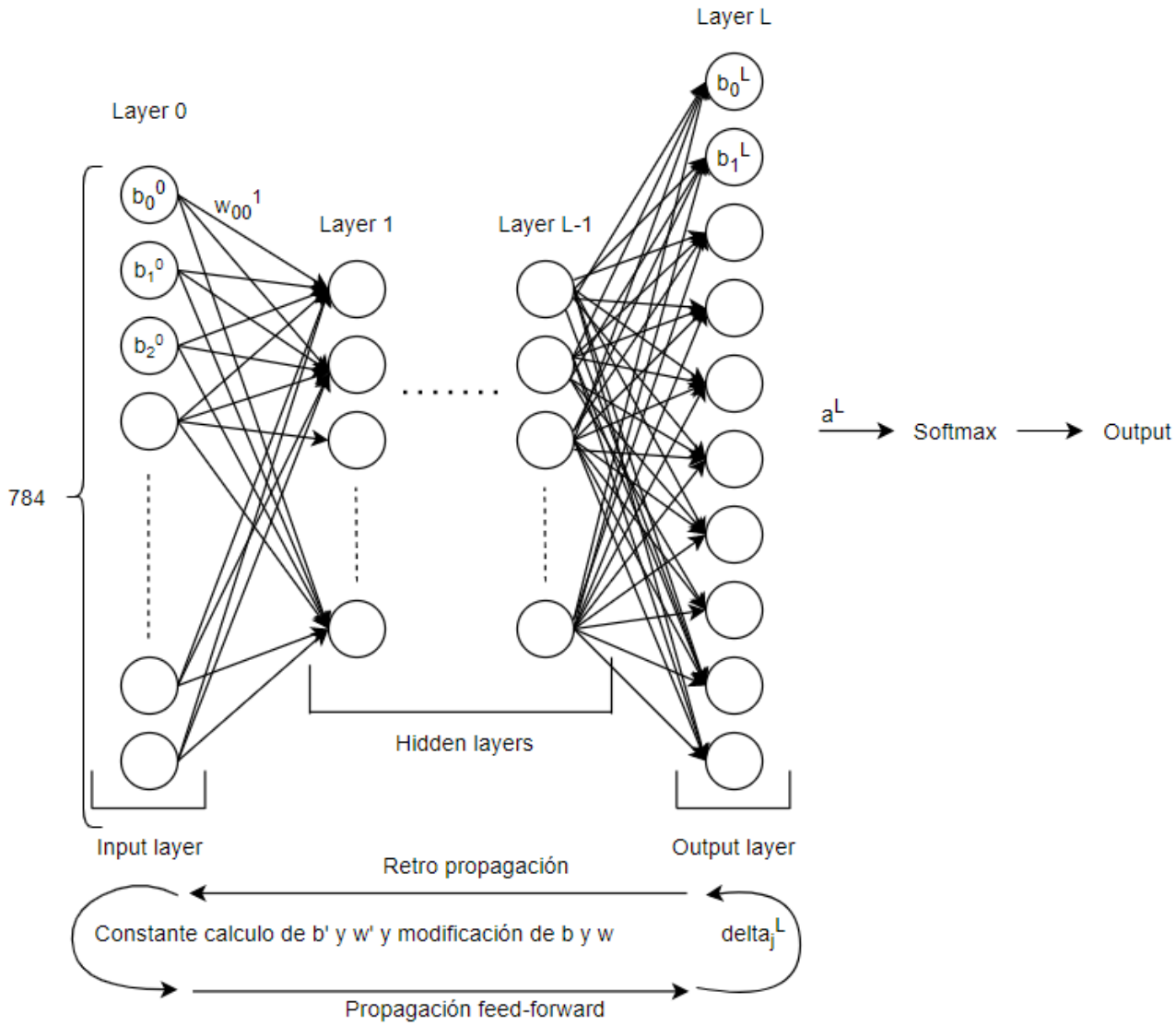


Figura 9. Esquema general de la RNA y SGD.

3.2 Determinación de hyperparametros.

Se ha separado el conjunto de entrenamiento de 60000 muestras en 2 distintos, uno de 50000 muestras que se denomina training_data y otro de 10000 muestras que se denomina validation_data. La razón por la cual se lleva

a cabo este procedimiento es que se han mencionado la existencia de varios hyperparametros pero no se ha indicado los valores de los mismos. Primero se lleva a cabo el entrenamiento con el conjunto de training_data y luego se evalúa la red con el conjunto test_data. De manera que, previamente, se habrá utilizado el conjunto validation_data para determinar mediante métodos heurísticos y experimentales los hyperparametros de la red.

Hasta ahora se han mencionado varios hyperparametros de la red, tales como el ratio de aprendizaje η, el número de capas ocultas y su tamaño, el tamaño de mini-batch 'm', el número epoch o la función de coste a utilizar. El único hyperparametros decidido hasta el momento es el de la función de coste, la función de coste cuadrática. Mientras que el tamaño del mini-batch 'm', que afecta mínimamente al rendimiento de la red y simplemente indica cuanto reducimos el tiempo de entrenamiento y el número epoch, cuya elección depende del sobre ajuste de la red y la velocidad de aprendizaje, determinar el resto de hyperparametros óptimos para la red resulta una tarea compleja cuyo problema no tiene una solución clara y concisa a día de hoy. La manera más adecuada de abordar este problema según muchos autores es mezclando técnicas heurísticas con experimentales.

Por conveniencia escogeremos un 'm' de 10 de modo que tendremos un total de 6000 muestras de 10 dígitos cada una. También seleccionaremos un número epoch lo suficientemente alto como para que el aprendizaje de la red se estabilice, al igual que un alto ratio de aprendizaje. A continuación se llevan a cabo una serie de experimentos para determinar unos hyperparametros aceptables para la red neuronal artificial.

Para llevar a cabo el entrenamiento de la red no se tiene que hacer más que la carga de los datos de entrenamiento mediante la función de la biblioteca 'mnist_loader', 'load_data_wrapper' y posteriormente, declarar un objeto tipo Network y hacer una llamada a la función SGD con los parámetro que se desee:

```
import Network1 as N1
import mnist_loader
import numpy as np

data_path = "D:/Backup/Carpeta/Universidad/Curso 2019-2020/TFG/NeuronalNet/data/"
input_layer = 784
mnist_data = 'mnist.pkl.gz'

print ("Cargando data...")
training_data, validation_data, test_data = mnist_loader.load_data_wrapper(mnist_data, input_layer) #Carga de los datos de entrenamiento
print ("Cargado")

#Entrenamiento de la red
net = N1.Network([784, 10])
net.SGD(data_path, 'net_data_EXP1', training_data[ :10000], 50, 10, 1, validation_data[ :2000])
```

3.2.1 Topología aproximada de la red.

En primer lugar se realizan algunos experimentos para determinar una topología aproximada adecuada para el aprendizaje de la RNA. A continuación se muestra la Tabla 1 con 6 experimentos. En este caso, en vez de usar el conjunto completo de 50000 muestras de entrenamiento se usaron solo 10000 muestras de entrenamiento y 2000 del conjunto de validación para ahorrar tiempo de computación:

	Topología	Epoch	m	ɲ	Precisión (%)
Experimento 1	[784,10]	50	10	0.5	67,09
Experimento 2	[784,30,10]	50	10	0.5	95,39
Experimento 3	[784,60,10]	50	10	0.5	76,67
Experimento 4	[784,30,10,10]	50	10	0.5	95,93
Experimento 5	[784,30,30,10]	50	10	0.5	97,03
Experimento 6	[784,60,30,10]	50	10	0.5	97,28

Tabla 1. Experimentos de topología de red 1

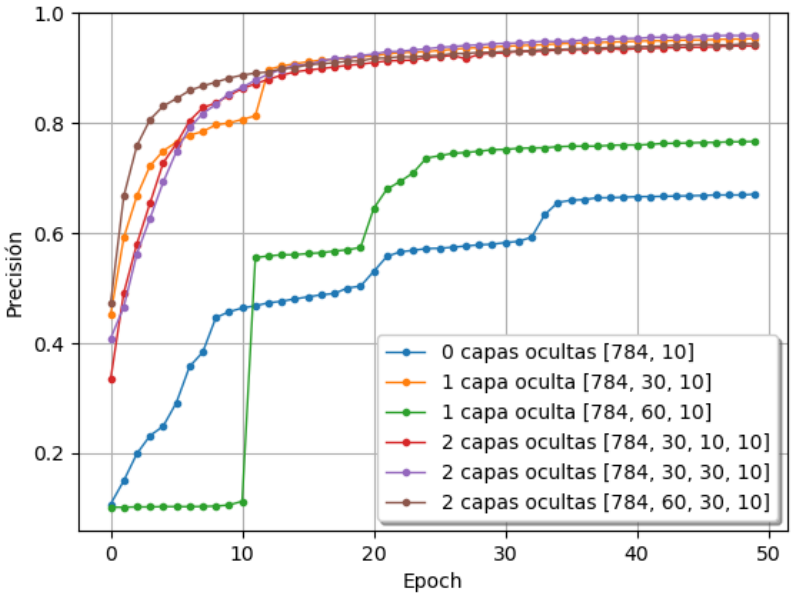


Figura 10. Experimentos de topología de red.

De estos experimentos podemos sacar la conclusión de que una topología de 2 capas ocultas de tamaño 60 y 30 se obtienen los mejores resultados, un 91,05%. Mediante esta serie de experimentos se hace uno idea de la topología aproximada que deberá tener la red neuronal artificial, 2 capas ocultas, ya que los 3 experimentos más exitosos poseen 2 capas ocultas. Con el uso de un mayor de capas ocultas se observó un comportamiento impredecible en los resultados de la red además de ralentizar exponencial los tiempos de entrenamiento.

3.2.2 Ratio de aprendizaje ɲ.

Se han llevado a cabo una serie de experimentos para estudiar la influencia del ratio de aprendizaje ɲ en el comportamiento de la red. Para ello se entrenó una red de topología [784,30,10] que ofrece resultados relativamente buenos junto con un tiempo de entrenamiento bastante reducido. Nuevamente se usaron únicamente 10000 muestras de entrenamiento y 2000 muestras del conjunto de validación para reducir los tiempos de entrenamiento.

	Topología	**Epoch**	**m**	**ɲ**	**Coste**
Experimento 1	[784,30,10]	50	10	0.01	0.12773
Experimento 2	[784,30,10]	50	10	0.1	0.04149
Experimento 3	[784,30,10]	50	10	1	0.00852
Experimento 4	[784,30,10]	50	10	10	0.01532
Experimento 5	[784,30,10]	50	10	100	0.07218

Tabla 2. Experimentos de ratio de aprendizaje de red.

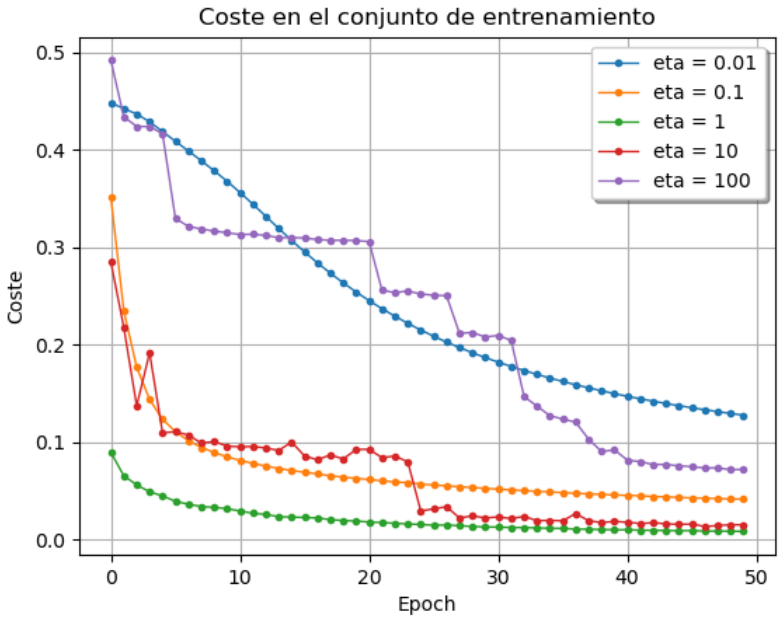


Figura 11. Experimentos de ratio de aprendizaje de red.

En la Figura 11, un ratio de aprendizaje η demasiado pequeño $\eta \ll 0.1$ ralentiza enormemente la velocidad de aprendizaje de la red. A la misma vez, un ratio de aprendizaje demasiado grande $\eta \gg 1$ provoca un comportamiento impredecible, largos periodos de tiempo en los que la red no experimenta un aprendizaje apreciable y pequeños periodos de tiempo en los que si se aprecia un aprendizaje inconsistente. Para valores de η entre 0.1 y 1 la red tiene un comportamiento aceptable y consistente. Este comportamiento se debe a la función de η en la ecuación (9). Partiendo de la idea conceptual de la pelota imaginaria que desciende a lo largo del valle, ratios de aprendizaje demasiado pequeños provocan que la pelota descienda lentamente, necesitando grandes periodos de entrenamiento para alcanzar un mínimo. Para ratios de aprendizaje demasiado grandes provoca que el paso de la pelota sea excesivo y 'sobrepase' un mínimo local de la función de coste, quedando estanca o descendiendo de golpe todo un valle [12]. Para futuros experimentos se seleccionarán valores de η comprendidos entre estos valores, 0.1 y 1.

3.3 Mejoras en la red neuronal artificial.

A continuación se exponen distintas técnicas usadas comúnmente para la mejora en el desempeño del aprendizaje de redes neuronales artificiales.

3.3.1 Función de cruce entrópica.

Como ya se mencionó en el apartado 3.1.3, el aprendizaje de la red se ralentiza cuando se produce al saturación de las neuronas de las red debido a la forma de la función sigmoide σ. Para solucionar este problema se usa en vez de la función de coste cuadrática (4), la función de coste de cruce entrópica o cross entropy en inglés [14]:

$$C = -\frac{1}{n}\sum_{x}[y \ln a + (1-y)\ln(1-a)] \tag{22}$$

Si se fija uno en la función de coste de cruce entrópica, cumple con las 2 condiciones ya expuestas que debe tener toda función de coste:

- Debe ser positiva para cada muestra de entrenamiento 'x'.
- Debe tender a 0 cuando la salida de la red se aproxima a la deseada para la muestra de entrenamiento.

Para ver el porqué de como el uso de esta función de coste elimina el problema de la ralentización en el aprendizaje cuando se produce saturación se calcula $\partial \mathrm{C}/\,\partial b_j$ y $\partial \mathrm{C}/\,\partial w_j$:

$$\frac{\partial \mathrm{C}}{\partial w_j} = \frac{1}{n}\sum_{x} x_j(a-y) \tag{23}$$

$$\frac{\partial \mathrm{C}}{\partial b_j} = \frac{1}{n}\sum_{x} (a-y) \tag{24}$$

Como se puede observar esta vez, la variación de la función de coste en función de los pesos y biases no viene como factor de la función sigmoide. Según las ecuaciones (23) y (24), la velocidad a la que aprenden los pesos y biases está controlada por $a-y$, es decir, por el error en la salida. Es más esto implica que la red aprenderá más rápido cuanto mayor sea el error cometido, un comportamiento lógico y más adecuado a una inteligencia biológica. Para probar esta afirmación se entrenó una RNA [784-30-10-10] con un ɲ=0.1 con un total de 1000 muestras de entrenamiento, variando en ambos casos la función de coste entre la función de error medio cuadrado y la función de coste entrópica:

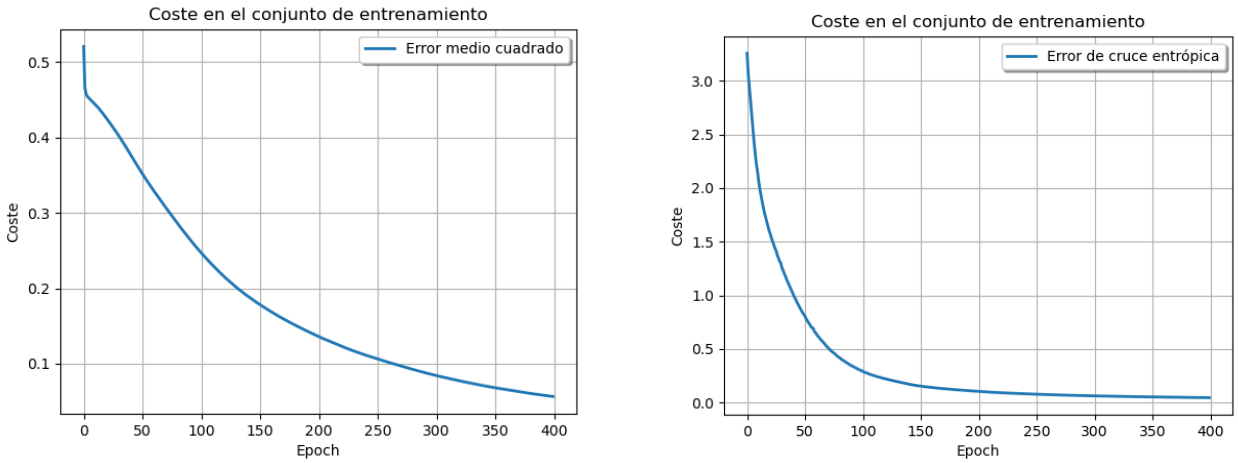


Figura 12. Comparación de velocidad de entrenamiento de una RNA.

A la izquierda se usó como función de coste MSE. A la derecha se usó como función de coste Cross Entropy.

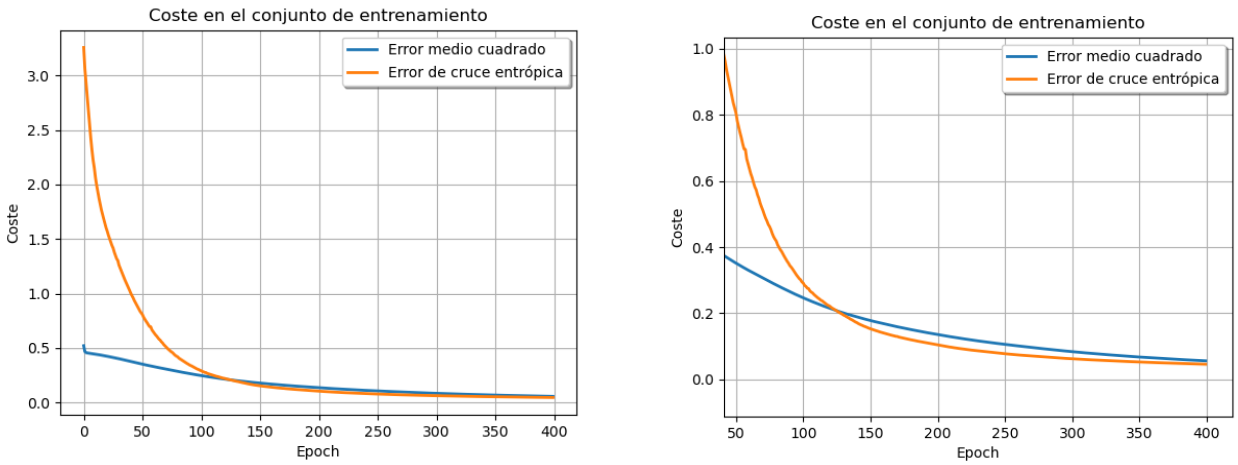


Figura 13. Comparación inmediata de velocidad de entrenamiento de una RNA.

A la derecha se presenta un zoom de la gráfica.

Como se puede observar en la Figura 12, con el uso de la función de cruce de entropía como función de coste de la red, la pendiente de la curva de aprendizaje de la red es mucho más pronunciada que para el uso de la función de error medio cuadrado, esto es, su velocidad de aprendizaje es mayor. Además también se puede observar que para, por ejemplo, un coste de 0.1, con el uso de la función de coste entrópica se alcanza alrededor del epoch 40 mientras que para la función de error medio cuadrado no se alcanza hasta el epoch 250.

3.3.2 Overfitting y técnicas de regularización.

El overfitting o sobreajuste se produce cuando se sobreentrena una red neuronal artificial. Esto produce que en vez de aprender a clasificar un conjunto de datos de carácter general, la red está memorizando el conjunto de

datos entrenamiento en concreto. Se ha realizado un entrenamiento con 1000 muestras de entrenamiento para una red de topología [784,30,10,10] con un ɲ=0.1 y un total de 400 epoch. Esto significa que nuestra red contará con un total de 2352050 parámetros. Teniendo en cuenta que el conjunto de entrenamiento está compuesto por un total de 1000 muestras de entrenamiento, la red podrá ajustarse fácilmente al conjunto de entrenamiento, como se puede ver en la Figura 14:

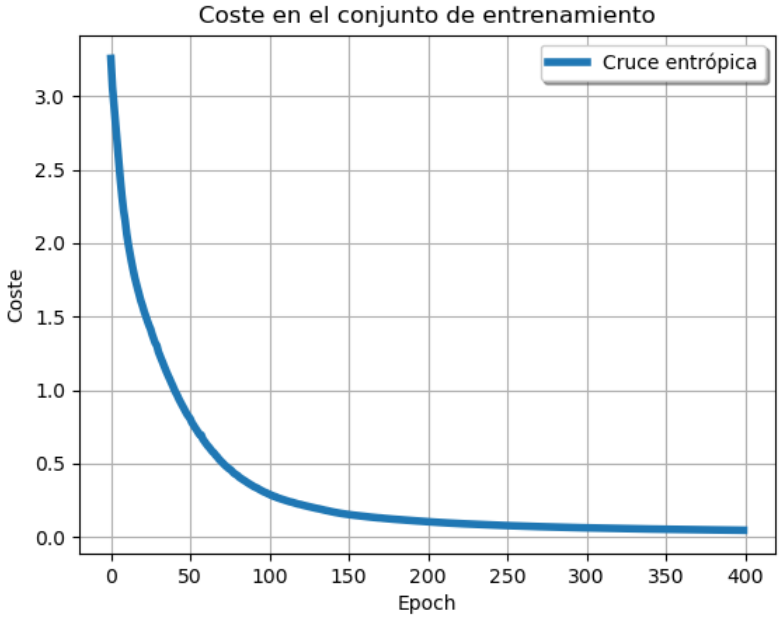


Figura 14. Coste del conjunto de entrenamiento de una RNA sobre ajustada.

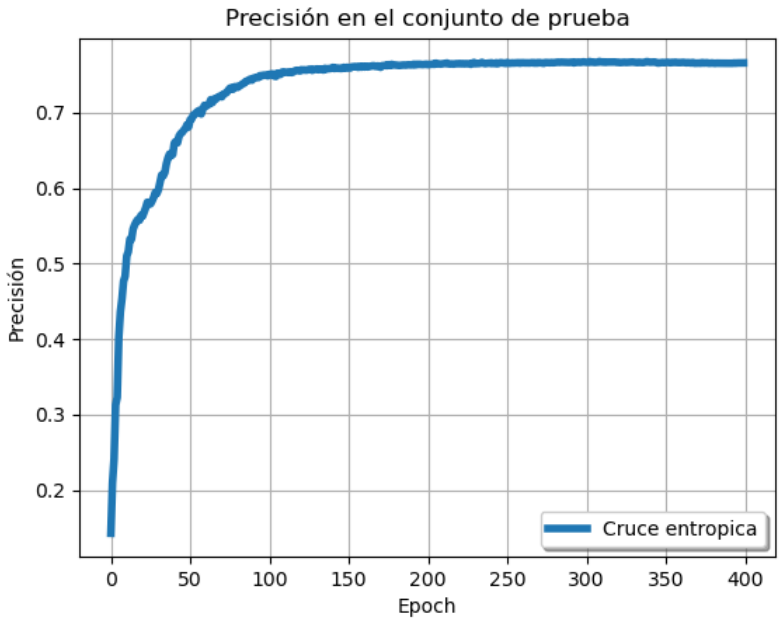


Figura 15. Precisión de una red sobre ajustada.

Se puede apreciar cómo aunque en la Figura 14, el coste a la salida de la red disminuye de manera significativa hasta un epoch de 170-180 aproximadamente, el aprendizaje real de la red se detiene a partir del epoch 100-110. Esto supone un problema ya que la red neuronal no estaría aprendiendo a clasificar dígitos, sino que estaría memorizando el conjunto de entrenamiento. Este comportamiento no es deseable ya que el propósito de la elaboración de la red es que sea capaz de clasificar dígitos de cualquier muestra y no únicamente del conjunto de entrenamiento. De hecho, como se puede observar en la Figura 16, el coste del conjunto de prueba aumenta a partir del epoch 80 mientras que la precisión de la red para el conjunto de entrenamiento alcanza casi el 100%. Esto prueba que en realidad, la red no está aprendiendo a clasificar dígitos numéricos, sino memorizando el conjunto de entrenamiento como se puede ver en la Figura 17.

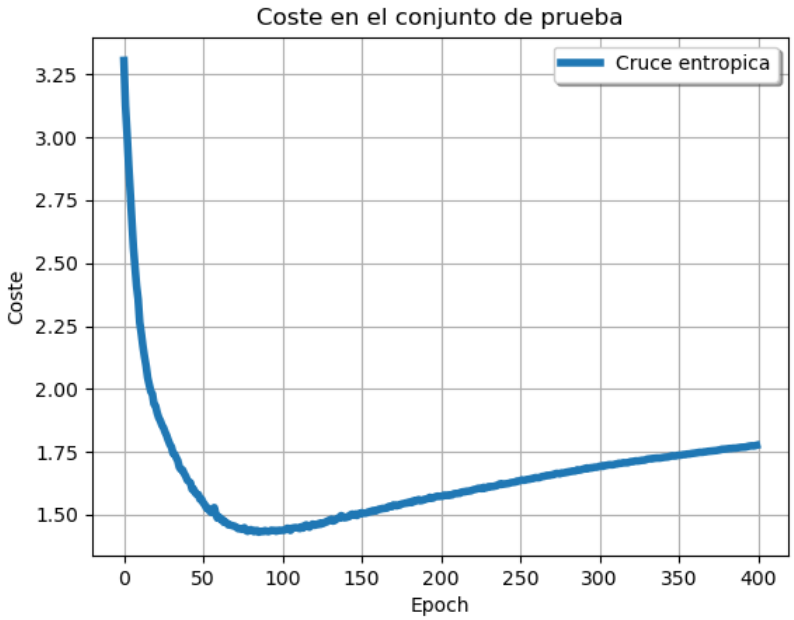


Figura 16. Coste del conjunto de prueba de una RNA sobre ajustada.

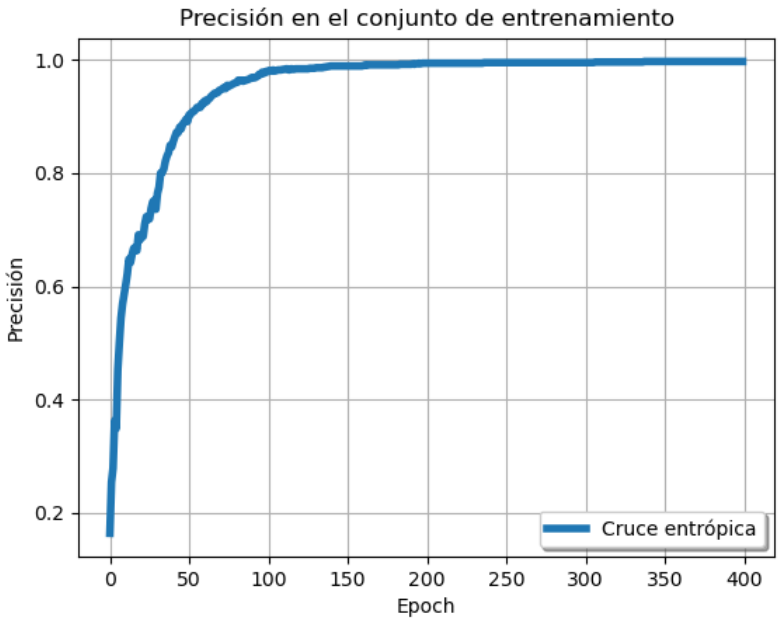


Figura 17. Precisión del conjunto de entrenamiento de una RNA sobre ajustada.

Existen varias maneras para abordar este problema, una de ellas consiste en detener el aprendizaje de la red cuando alcanzamos el porcentaje de precisión máximo de la red, o 'early stopping' en inglés. También podemos ampliar el conjunto de entrenamiento de forma artificial mediante transformaciones morfológicas, desplazamientos y deformaciones. Otra posible solución es reducir el tamaño de la red; a menor tamaño, menor número de parámetros que posee la red para ajustarse al conjunto de entrenamiento.

Aparte de estas técnicas, existen otro tipo de técnicas para resolver o minimizar el efecto del overfitting, conocidas como técnicas de regularización.

En concreto, se ha decidido usar la técnica de regularización 'L2' o 'weight decay' [15] [16].

En la técnica de regularización L2 se añade un término extra a la función de coste, también llamado termino de regularización, función del cuadrado de la suma de los pesos de la red:

$$C = -\frac{1}{n}\sum_{xj}[y_j \ln a_j^L + (1-y_j)\ln(1-a_j^L)] + \frac{\lambda}{2n}\sum_{w} w^2 \qquad (25)$$

De la expresión (25) se puede observar como a la función de coste de cruce entrópica se le aditiva la suma de los cuadrados del conjunto de pesos de la red, escalado por un factor lambda $\lambda/2n > 0$. Al hyperparametros λ también se le conoce como parámetro de regulación. Destacar que esta técnica es aplicable a cualquier tipo de función de coste, no solo a la de cruce entrópica.

Intuitivamente, se puede decir que la adición de este factor a la función de coste busca reducir el conjunto de los pesos de la red. Ya que la red se encargará de clasificar dígitos numéricos, se puede interpretar como darle una mayor importancia al conjunto de pixeles en su totalidad, en vez de a pixeles clave para el reconocimiento del digito. El factor de regularización λ indica la prioridad a reducir la función de coste o reducir el conjunto de pesos de la red.

Retomando de nuevo el cálculo de $\partial \mathrm{C}/\,\partial b_j$ y $\partial \mathrm{C}/\,\partial w_j$:

$$\frac{\partial \mathrm{C}}{\partial w} = \frac{\partial \mathrm{C}_0}{\partial w} + \frac{\lambda}{n} w \tag{26}$$

$$\frac{\partial \mathrm{C}}{\partial b} = \frac{\partial \mathrm{C}_0}{\partial b} \tag{27}$$

Donde C_0 es la ecuación de coste a usar por la red sin la técnica de regularización. Mientras que el uso de la técnica de regularización L2 no afecta al cómputo del conjunto de biases, para el caso del conjunto de pesos ya se ha visto que $\partial \mathrm{C}_0/\,\partial w$ es fácilmente computable al igual que el termino $\frac{\lambda}{n} w$.

De esta forma la regla de aprendizaje de los pesos y biases pasa a ser:

$$b \to b - \mathrm{ɲ} \frac{\partial \mathrm{C}_0}{\partial b} \tag{28}$$

$$w \to w - \mathrm{ɲ} \frac{\partial \mathrm{C}_0}{\partial w} + \frac{\lambda}{n} w = (1 - \frac{\mathrm{ɲ}\lambda}{n}) w - \mathrm{ɲ} \frac{\partial \mathrm{C}_0}{\partial w} \tag{29}$$

Se puede ver en la ecuación (29) que resulta ser la misma que teníamos anteriormente, la ecuación (10), escalada por un factor $= (1 - \frac{\mathrm{ɲ}\lambda}{n})$, también conocido como factor de decaimiento de peso o 'weight decay' en inglés.

Se ha llevado a cabo el entrenamiento de una RNA [784,30,10,10] con un ɲ=0.1 y un λ = 5.0 con 1000 muestras de entrenamiento:

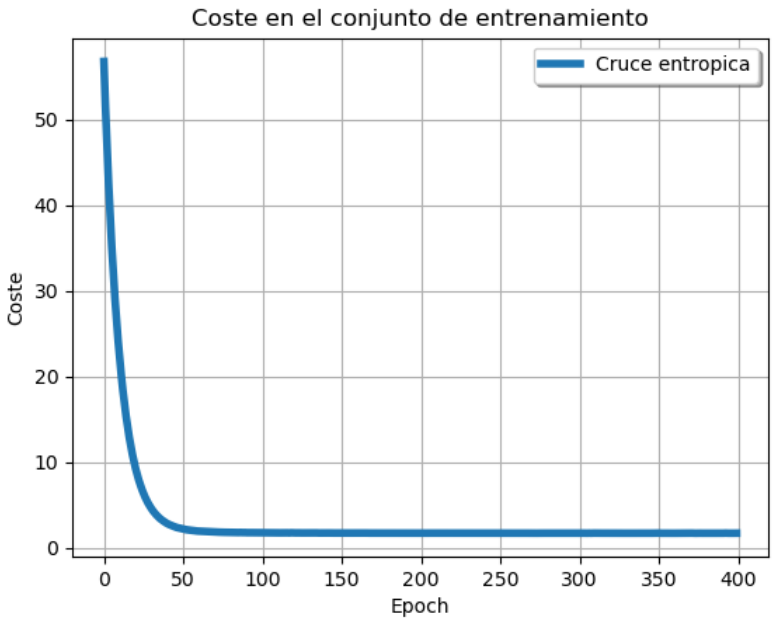


Figura 18. Coste del conjunto de entrenamiento con uso de regularización.

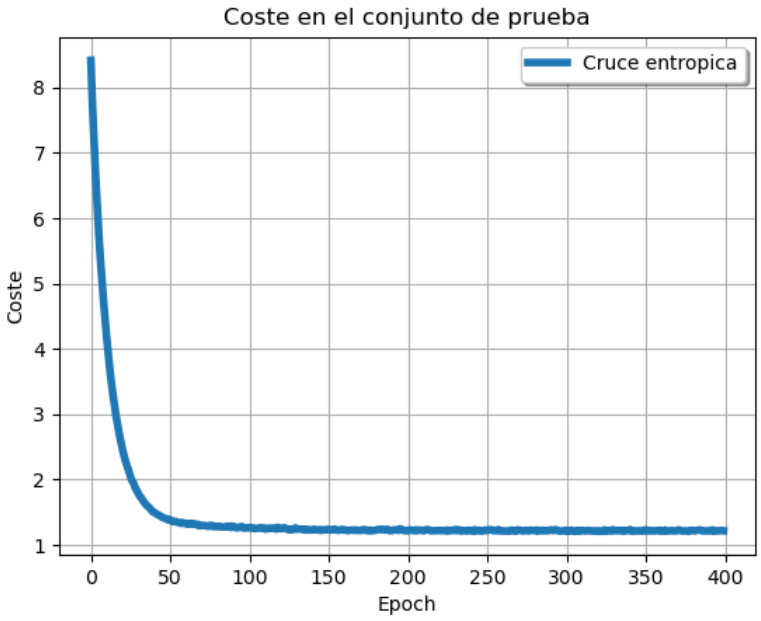


Figura 19. Coste del conjunto de validación con uso de regularización.

Como se puede observar en la Figura 19, con el uso de técnicas de regularización la función de coste de aprendizaje de la red evaluada para el conjunto de datos de validación se mantiene decreciendo a lo largo de todo el proceso de aprendizaje, a diferencia de lo que ocurría cuando se sobre ajustaba la RNA en la Figura 16. Además, llevando a cabo un entrenamiento con early-stopping y L2 para el conjunto completo de 50000 muestras de entrenamiento, se puede ver en la Figura 21 cómo se ha reducido la diferencia entre la respuesta

de la red ante el conjunto de entrenamiento y el de prueba. Esto implica una gran mejora en el funcionamiento de la red, con el que se obtiene una precisión para el conjunto de validación del 96.65% de aciertos.

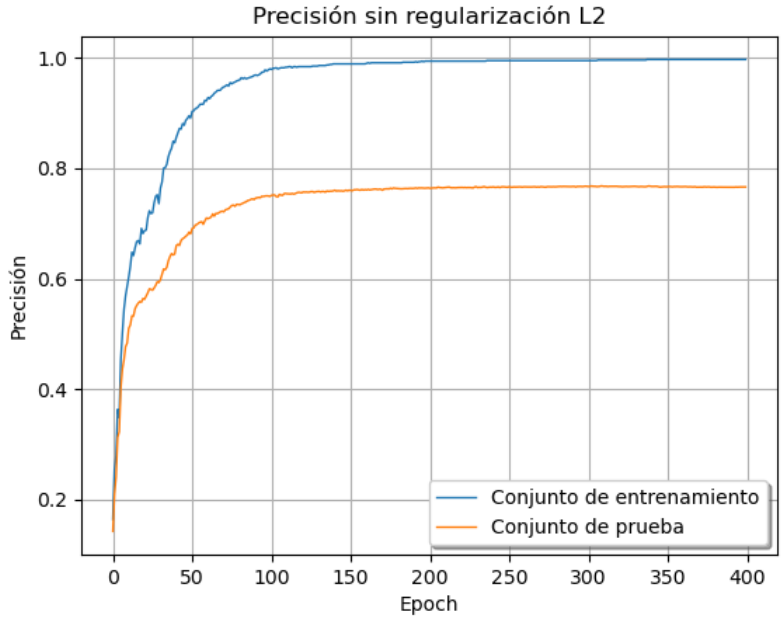


Figura 20. Precisión de una RNA sin aplicar L2.

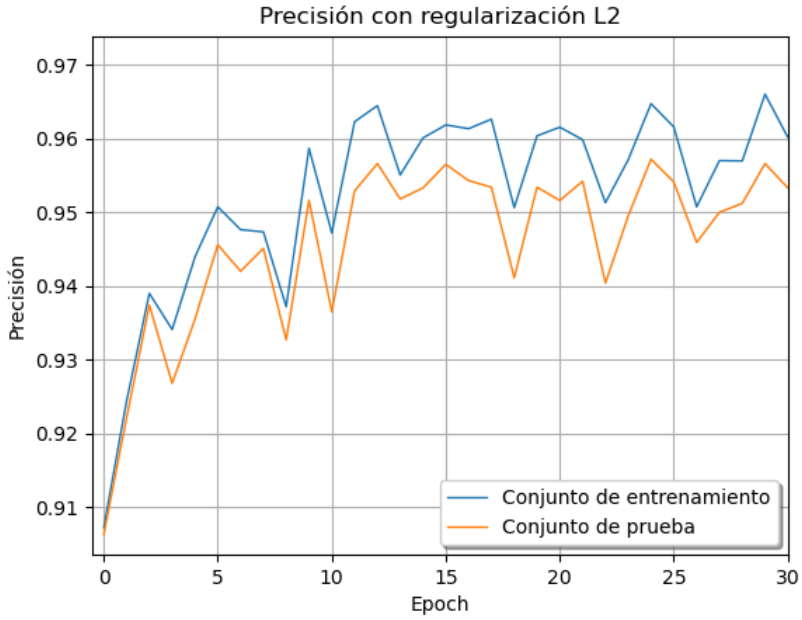


Figura 21. Precisión de una RNA aplicando early-stopping y L2.

3.3.3 Inicialización del conjunto de pesos y biases.

Hasta ahora la red inicializaba su conjunto de pesos y biases de forma aleatoria, siguiendo una distribución gaussiana de media 0 y desviación típica 1. Sin embargo otros autores [17] sugieren otras formas de inicialización de estos parámetros de la red. Este nuevo método consiste en la inicialización de los pesos y biases mediante una distribución gaussiana de media 0 y desviación típica $1/\sqrt{x}$ donde x es el tamaño de la capa de entrada. Esto se debe principalmente al problema mencionado anteriormente con el problema de la saturación de la neuronas ralentizando el aprendizaje de la red. Recordar que la entrada a la red será una imagen en escala de grises binarizada, esto implicará que muchos de los pixeles de la imagen tendrán valores muy próximos a 0 o a 1.Tomando una capa de entrada de tamaño 784 donde por ejemplo la mitad de los pixeles serán muy próximos a 0 y la otra mitad muy próximos a 1, ya que $z = \sum_j w_j x_j + b$ implicará que 390 pixeles no aportarán a esta suma. Luego z tendrá una distribución gaussiana de media 0 y $\sigma = \sqrt{391} \approx 19.77$. Observando el aspecto de esta distribución gaussiana en la Figura 22, z tomará valores generalmente $z \gg 1$ *o* $z \ll -1$, de modo que la activación de la neurona saturará.

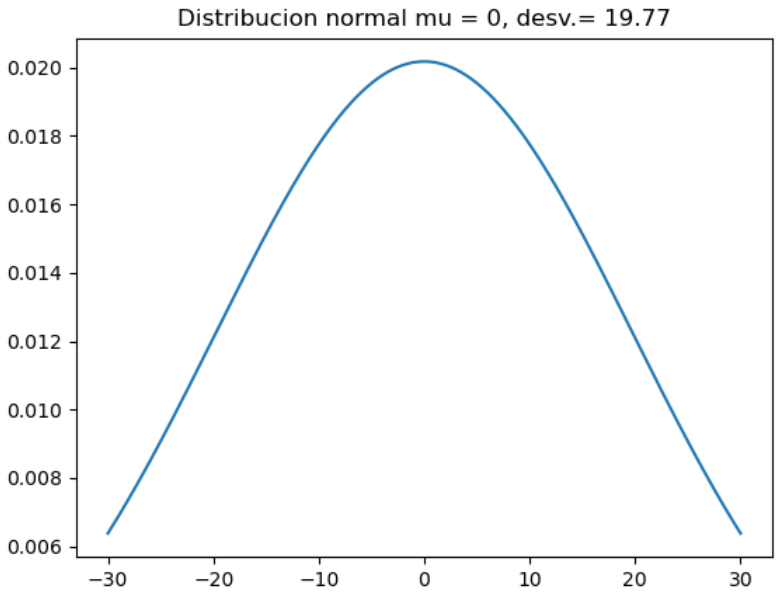


Figura 22. Distribución Normal μ=0, σ=19.77.

Ahora para el mismo caso donde 390 pixeles tendrán un valor muy próximo o igual a 0 pero usando como método de inicialización del conjunto de pesos una distribución gaussiana de media 0 y $\sigma = \sqrt{1/784} \approx \frac{1}{28} \approx 0.0357$, z tendrá seguirá una distribución gaussiana de media 0 y $\sigma = \sqrt{3/2} \approx 1.22$. Observando el aspecto de la distribución gaussiana de la Figura 23, esta vez z tomará valores próximos a 0, de manera que la neurona no saturará. Se puede ver en la Figura 24, como con el nuevo método de inicialización del conjunto de pesos se obtienen mejores resultados en los primeros epoch del periodo de aprendizaje.

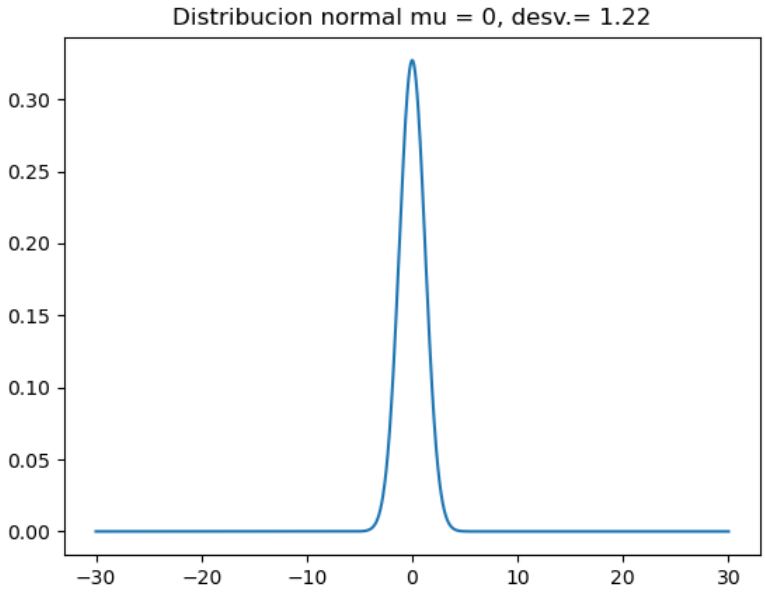


Figura 23. Distribución Normal μ=0, σ=1.22.

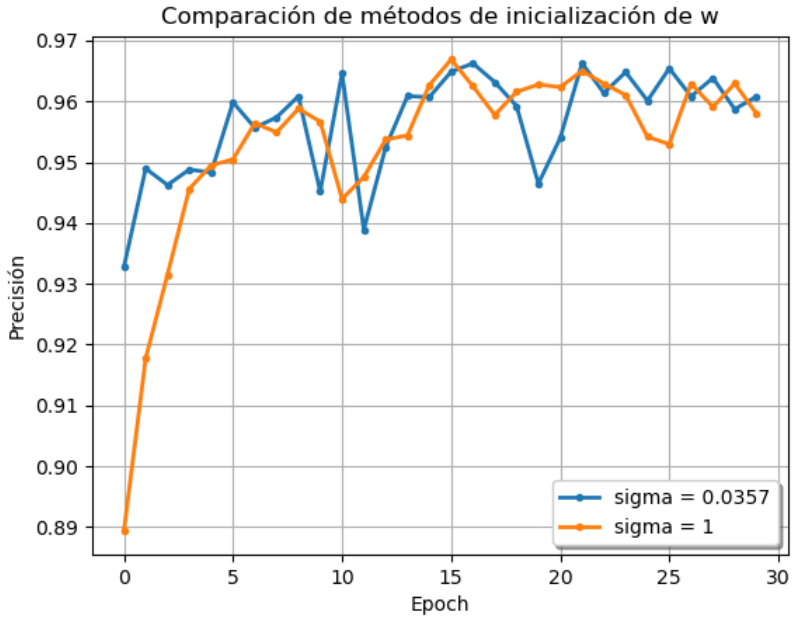


Figura 24. Comparación de métodos de inicialización del conjunto de pesos.

3.3.4 Ampliación del conjunto de entrenamiento de forma artificial.

Es una práctica común en el entrenamiento de redes neuronales artificiales la ampliación del conjunto de datos de entrenamiento de forma artificial. Esta ampliación suele tratarse de desplazamientos, giros, reflexiones y transformaciones morfológicas del conjunto de entrenamiento inicial [18]. En nuestro caso se llevó a cabo el desplazamiento de las muestras de entrenamiento un píxel en cada dirección, arriba, abajo, izquierda y derecha,

resultando un nuevo conjunto de 200000 nuevas muestras que añadidas a las 70000 (50000 del conjunto de entrenamiento, 10000 del conjunto de validación y 10000 del conjunto de prueba) originales forman un total de 350000 muestras en total. El uso de esta ampliación artificial reduce el efecto del sobre ajuste en redes neuronales como ya se mencionó en el apartado 3.3.2.

Se ha llevado el entrenamiento de una red de topología [784,30,10] con un epoch 30, un ratio de aprendizaje ɲ = 0,5 y un λ = 5 con el conjunto de entrenamiento expandido y sin expandir para observar las diferencias en el comportamiento de la red:

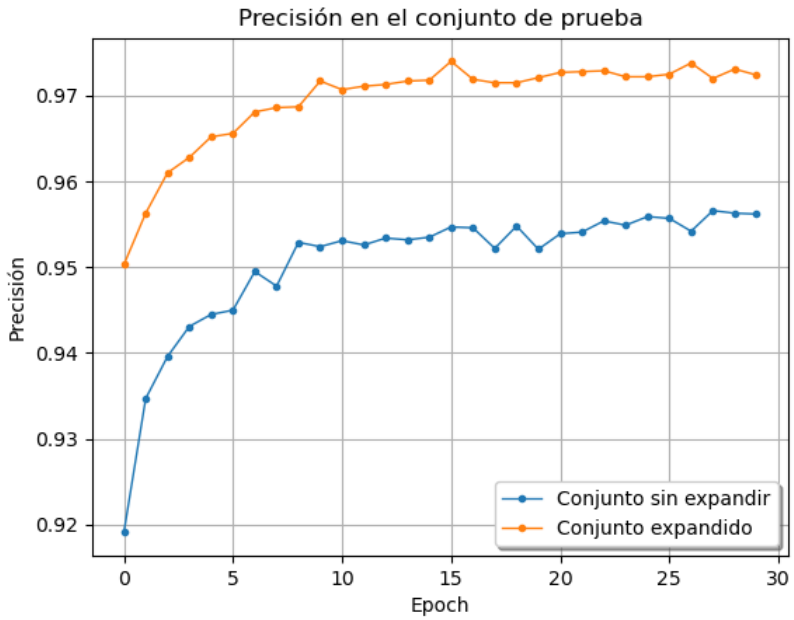


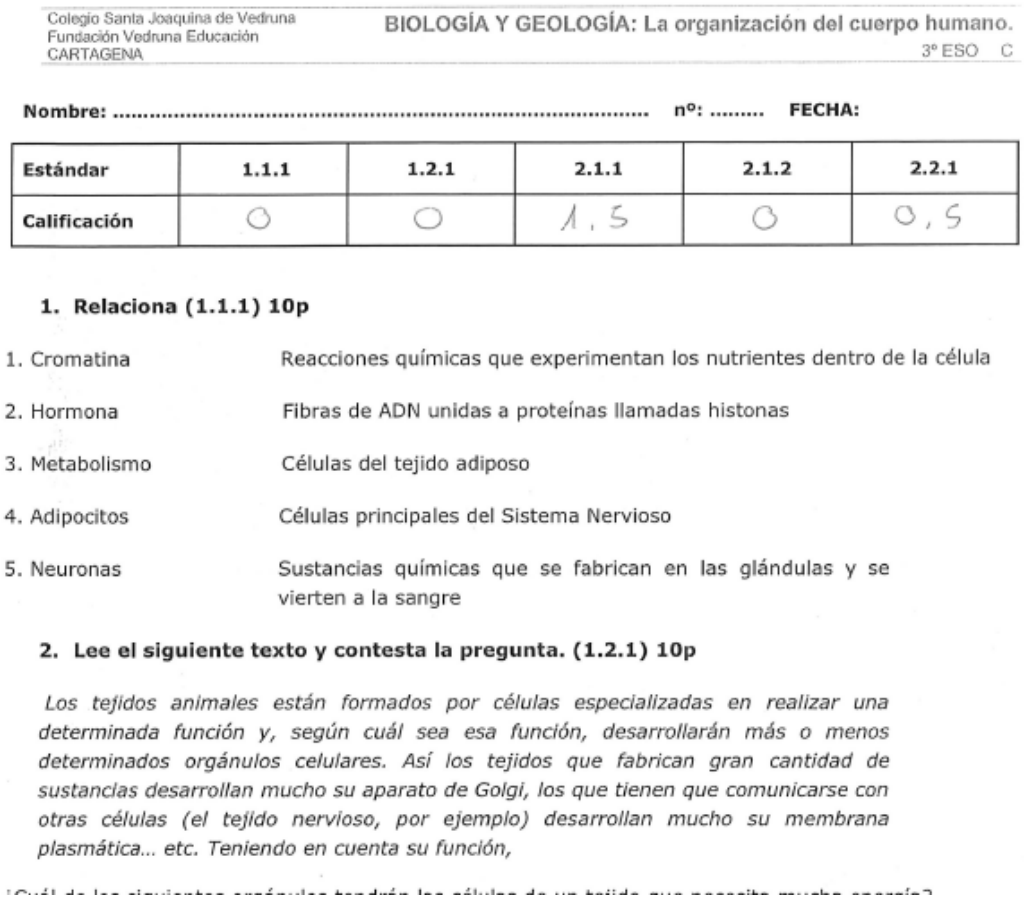
Figura 25. Comparación de precisión con el conjunto de datos expandido y sin expandir.

3.4 Procesamiento de imagen referente al examen.

La red neuronal artificial es capaz de clasificar dígitos del 0 al 9, pero no es capaz de detectar su ubicación en una imagen o de discernir cuando una símbolo es un digito y cuando no lo es, ya que no importa que imagen se le introduzca como entrada a la red, está siempre intentará clasificar el patrón de la imagen en uno de los 10 dígitos arábigos modernos. Usualmente, el docente evaluará entre 3 y 15 estándares por examen, esto quiere decir que además de ser capaces de detectar las calificaciones de cada estándar, es necesario vincular cada calificación a su estándar correspondiente, identificando también dichos estándares de evaluación. También será necesario, una vez detectada la calificación, descomponerla en sus dígitos individuales y signos de puntuación como 'comas' y 'puntos' en el caso de tratar con indicadores o números decimales. Para resolver todos estos problemas se hará uso de técnicas de visión por computador y la selección de un formato de examen con limitaciones.

3.4.1 Formato de examen.

Para solucionar parte de estos inconvenientes se propone el uso de un formato de examen ligeramente limitado en donde será imprescindible que se ubique una tabla horizontal, que contendrá los estándares y calificaciones de la prueba evaluativa, compuesta por 2 filas y tantas columnas como estándares se estén evaluando más una columna para etiquetas que se situará en la primera posición. Además será imprescindible que dicha tabla esté ubicada en la primera página del examen. Se muestra en la Figura 26 lo que sería un ejemplo de examen de Biología y Geología de 3° ESO. De esta forma se asignará a la primera fila los estándares a evaluar y en la segunda fila las calificaciones de dichos estándares.



Colegio Santa Joaquina de Vedruna
Fundación Vedruna Educación
CARTAGENA

BIOLOGÍA Y GEOLOGÍA: La organización del cuerpo humano.
3º ESO C

Nombre: ..... **nº:** ..... **FECHA:**

Estándar	1.1.1	1.2.1	2.1.1	2.1.2	2.2.1
Calificación	0	0	1,5	0	0,5

1. Relaciona (1.1.1) 10p

1. Cromatina	Reacciones químicas que experimentan los nutrientes dentro de la célula
2. Hormona	Fibras de ADN unidas a proteínas llamadas histonas
3. Metabolismo	Células del tejido adiposo
4. Adipocitos	Células principales del Sistema Nervioso
5. Neuronas	Sustancias químicas que se fabrican en las glándulas y se vierten a la sangre

2. Lee el siguiente texto y contesta la pregunta. (1.2.1) 10p

Los tejidos animales están formados por células especializadas en realizar una determinada función y, según cuál sea esa función, desarrollarán más o menos determinados orgánulos celulares. Así los tejidos que fabrican gran cantidad de sustancias desarrollan mucho su aparato de Golgi, los que tienen que comunicarse con otras células (el tejido nervioso, por ejemplo) desarrollan mucho su membrana plasmática... etc. Teniendo en cuenta su función,

Figura 26. Ejemplo de tabla de calificaciones.

3.4.2 Detección de la tabla de calificaciones y estándares.

En primera instancia se intentó elaborar un algoritmo de visión por computador capaz de extraer de una imagen de un examen, las celdas de la tabla de calificaciones. Para esto convertimos el formato PDF del examen a un formato de imagen PNG o similar. En primer lugar se le pedirá al usuario que seleccione de forma aproximada la ubicación de la tabla en el examen. Se recorta la tabla de calificaciones del examen y se lleva a cabo la

transformación de la imagen a escala de grises y su binarización para separar la información que contengan la imagen del fondo blanco del papel.

En esta parte del trabajo resulta de vital importancia el uso de la biblioteca de open source OpenCV, muy conocida y usada en el entorno de visión artificial. Para la extracción de las celdas de la tabla se combinan técnicas de extracción de líneas de una imagen junto con detección de blobs y detección de contornos. Para la detección de líneas se ha usado la función de OpenCV de HoughLinesP mientras que para la detección de contornos se usa el detector de Canny. A continuación se muestras en la Figura 27 la estructura detallas del sistema diseñado y marcado en purpura la parte correspondiente al algoritmo de detección de las calificaciones.

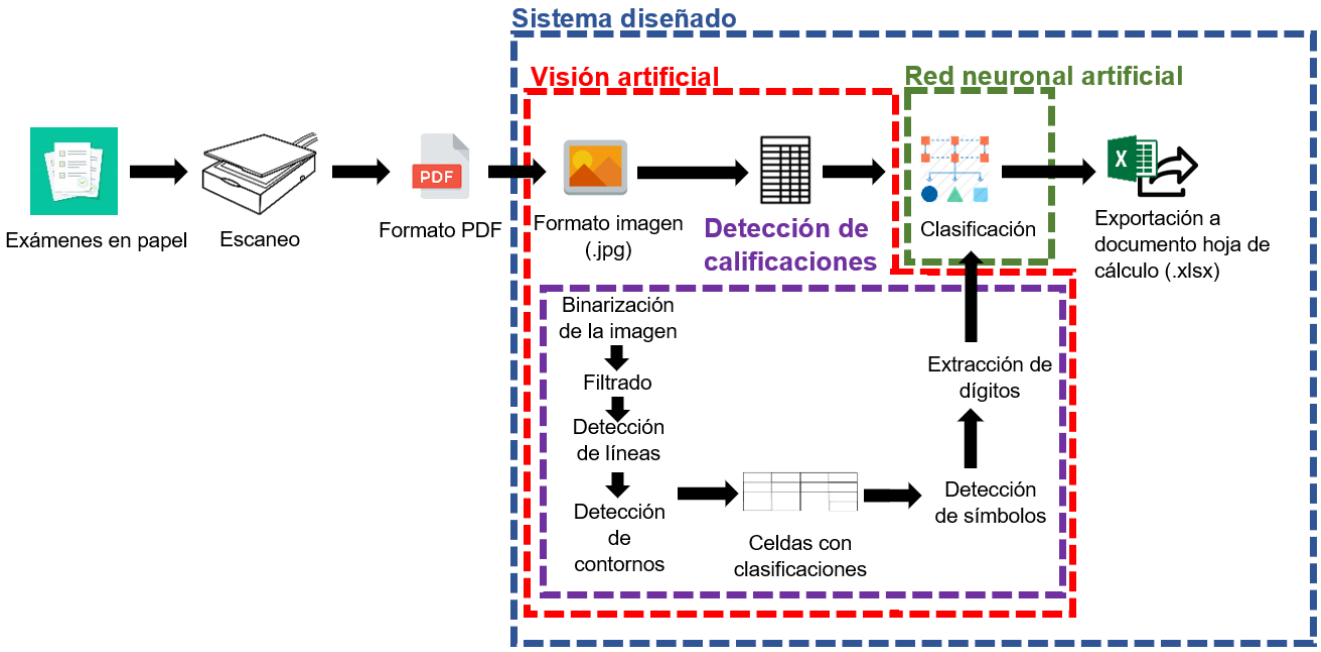


Figura 27. Estructura detalla del sistema software diseñado.

A continuación de muestra la función grid, encargada de devolver la tabla recortada por el usuario y las coordenadas de cada celda de la tabla en la imagen:

```
def grid(img, coord, kernel = cv2.getStructuringElement(cv2.MORPH_CROSS, (5, 5)), show = True):
  """Devuelve la imagen de la tabla seleccionada y un array con las coordenadas de las celdas de la tabla seleccionada"""

  celdas = [] #Lista donde almacenaremos cada casilla de la tabla

  tabla = img[coord[0]:coord[1],coord[2]:coord[3]] #tabla recortada del examen
  gray = cv2.cvtColor(tabla, cv2.COLOR_BGR2GRAY)
  ret, bin = cv2.threshold(gray, 230, 255, cv2.THRESH_BINARY_INV)

  bin = cv2.morphologyEx(bin, cv2.MORPH_DILATE, kernel) #Dilato la imagen binaria para completar posibles líneas discontinuas
```

```
img_line = np.zeros((tabla.shape), np.uint8) #Imagen donde pintaremos las líneas

lines = cv2.HoughLinesP(bin, 100, 90 * np.pi/180, 0, minLineLength = 10, maxLineGap = 7) #Detección de líneas
for line in lines:
    pt1 = (line[0][0], line[0][1])
    pt2 = (line[0][2], line[0][3])
    cv2.line(img_line, pt1, pt2, (255, 255, 255), thickness = 4) #Pintamos las líneas

img_line = cv2.cvtColor(img_line, cv2.COLOR_BGR2GRAY)
ret, img_line = cv2.threshold(img_line, 127, 255, cv2.THRESH_BINARY)

cnt, hierarchy = cv2.findContours(img_line, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE) #Sacamos los contornos de las líneas para obtener las celdas

img_ = np.zeros((tabla.shape), np.uint8)
```

La función findContours perteneciente al módulo de OpenCV devuelve 2 conjuntos de datos. En primer lugar cnt, donde se almacenan las coordenadas dentro de la imagen que contiene el contorno del objeto. En segundo lugar hierarchy, en donde se representa la jerarquía de los contornos para indicar cuando un contorno está situado dentro de otro y viceversa, como se puede ver en la Figura 28.

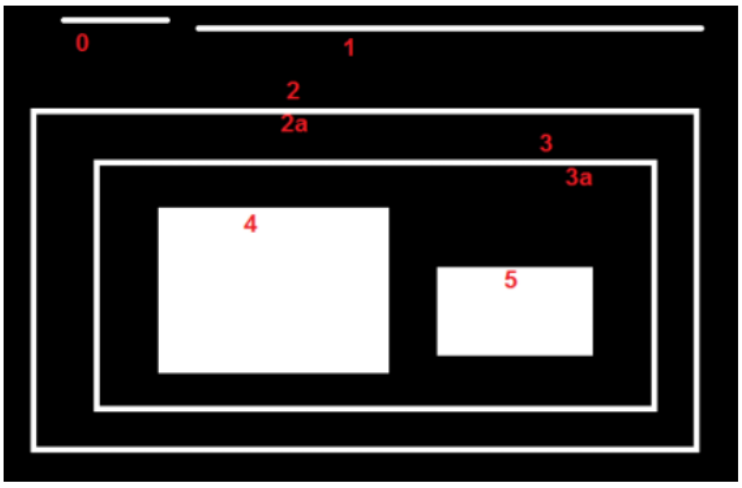


Figura 28. Jerarquía de detección de contornos en una imagen.

Haciendo uso de la variable hierarchy se eliminan aquellos contornos que no tengan contornos dentro, es decir, que no tengan contornos hijo, ya que todos estos contornos no representan una celda. Posteriormente se eliminan esta vez los contornos que no posean contornos exteriores, es decir, que no tengan ningún contorno padre.

```
#Eliminamos los contornos exteriores
for i in range(len(hierarchy[0])):
    if hierarchy[0][i][3] != -1: #Solo los contornos que tienen un padre
        x,y,w,h = cv2.boundingRect(cnt[i])
        cv2.rectangle(img_, (x,y), (x+w,y+h), (255, 255, 255), thickness = 1) #Dibujamos los contornos
```

```
img_ = cv2.cvtColor(img_, cv2.COLOR_BGR2GRAY)
ret, img_ = cv2.threshold(img_, 127, 255, cv2.THRESH_BINARY)

cnt, hierarchy = cv2.findContours(img_, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

#Repetimos la operacion anterior pero esta vez para eliminar los contornos interiores
img_cell = np.zeros((tabla.shape), np.uint8) #imagen donde pintaremos las celdas
for i in range(len(hierarchy[0])):
  if hierarchy[0][i][3] == -1: #Solo los contornos que NO tienen un padre
    x,y,w,h = cv2.boundingRect(cnt[i])
    cv2.rectangle(img_cell,(x,y),(x+w,y+h),(255,255,255), thickness = 1) #Pintamos las celdas

    celdas.append([x, y, w, h])

celdas = organize(celdas) #Función que organiza correctamente las celdas de una tabla
```

Una vez se han extraído los contornos de las celdas donde se encontrarán los estándares y calificaciones del examen se muestran por pantalla, se ordenan y se normalizan, como se muestra en la Figura 29.

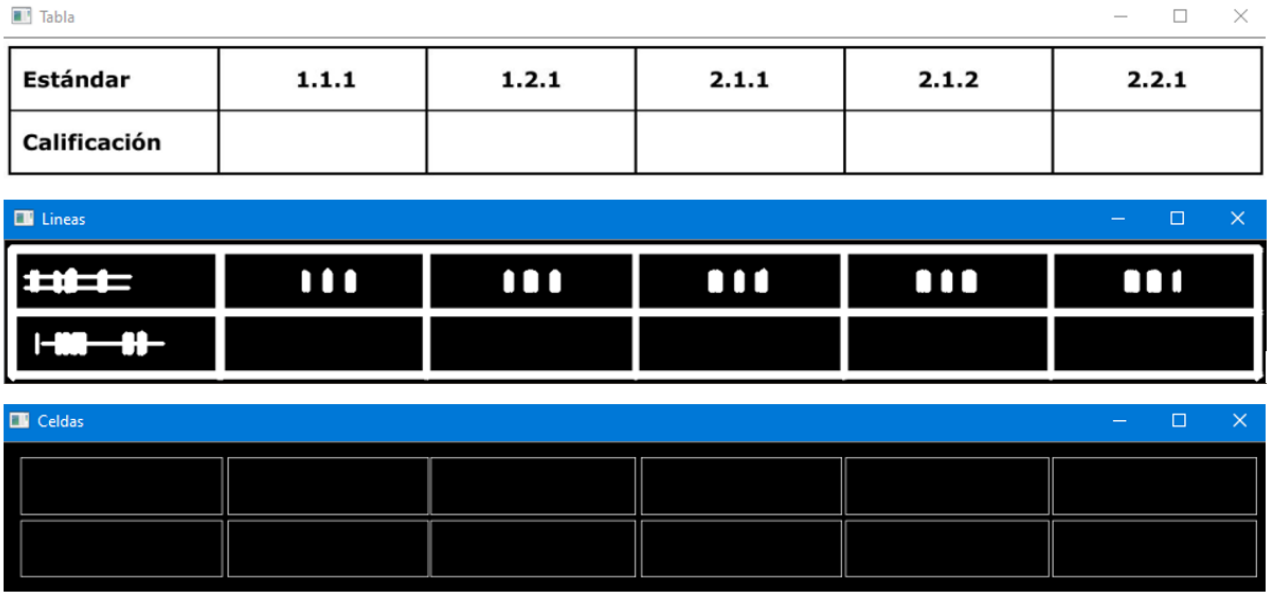


Figura 29. Detección de celdas de una tabla.

Y finalmente se muestra por pantalla:

```
if show:
  img = image_resize(img, None, root.winfo_screenheight())
  tabla_ = image_resize(tabla, int(root.winfo_screenwidth() / 2), None)
  img_line = image_resize(img_line, int(root.winfo_screenwidth() / 2), None)
  img_cell = image_resize(img_cell, int(root.winfo_screenwidth() / 2), None)
  cv2.imshow('Pagina', img)
  cv2.imshow('Tabla', tabla_)
  cv2.imshow('Lineas', img_line)
  cv2.imshow('Celdas', img_cell)
  cv2.waitKey(0)
```

```
    cv2.destroyAllWindows

  return tabla, celdas
```

Tras la extracción de las celdas se detecta para cada celda los simbolos que contiene dicha celda y se evalua si dicho simbolo es un signo de puntuación como una 'coma' o un 'punto' o si corresponde a un dígito. En caso de ser un digito, se le pasa a la red para que se lleve a cabo su clasificación. Para la detección de un signo de puntuación como una 'coma' o 'punto' se evaluará en función de la altura del simbolo con mayor altura en la celda con respecto al resto de simbolos en el interior de la celda. Si la altura del simbolo es menor o igual a la mitad de la altura del simbolo más alto en la celda, se evaluará como 'coma'. En caso contrario se interpreta como un dígito. Tras la detección de todos los dígitos de la tabla, ya que se conoce la posición de cada simbolo dentro de la celda, se puede saber el valor correspondiente a cada una ordenando las repuestas de la red de izquierda a derecha.

3.4.3 Exportación de los resultados a una hoja de datos en forma de tabla.

Una vez identificadas las calificaciones y estándares de la tabla de la prueba evaluativa se exportan a un archivo de tipo hoja de cálculo en forma de tabla. En este caso en concreto se exportan a un archivo con extensión '.xlsx'.

Para ello se ha elaborado la función Export, que crea un libro Excel y que escribirá la tabla a exportar en la primera hoja del libro.

```
def Export(data, filename): #Exporta una tabla a un documento tipo hoja de cálculo de forma organizada
  dirname = tk.filedialog.askdirectory() #Escoger la ruta donde se creará la hoja de cálculo
  book = opx.Workbook() #Creación el libro

  sheet = book.active
  sheet['A1'] = 'Estandares'
  sheet['A2'] = 'Calificaciones'
  sheet['A1'].font = Font(bold = True)
  sheet['A2'].font = Font(bold = True)

  c = 2
  for standar in data[1]: #Los estandares
    sheet.cell(row = 1, column = c).value = standar  #Ponemos los estándares
    c = c + 1

  r = 2
  for marks in data[2:]:
    c = 2
    for value in marks:
      sheet.cell(row = r, column = c).value = value  #Ponemos las calificaciones
      c = c + 1
    r = r + 1
```

```
#Guardado del libro
book.save(filename + '.xlsx')
if not os.path.isfile(dirname + '/' + filename + '.xlsx'): #Comprobar que no existe ya el libro
    shutil.move(filename + '.xlsx', dirname)
else: #Si existe, se reescribe
    print('Ya existe un archivo con el mismo nombre')
    shutil.move(filename + '.xlsx', dirname + '/' + filename + '.xlsx')
```

Como ya se ha mencionado anteriormente, se ignorará la primera columna de la tabla, reservada a las etiquetas de estándares y calificaciones, se le asigna el reconocimiento de la primera fila de la tabla a los estándares y la segunda a las calificaciones.

Tanto la función grid como muchas otras funciones de elaboración propias estan recogidas en un modulo propio creado con el proposito de presentar de forma más organizada y legible el código correspondiente a este trabajo que se ha denominado Miscellaneous.

4 Análisis de resultados.

En este capítulo se llevan a cabo distintas pruebas con el sistema software elaborado, se hace un análisis de los resultados obtenidos de dichas pruebas y se lleva a cabo una discusión sobre estos resultados.

El escaneo de la prueba manuscrita se realizó con equipo multifunción, como se puede ver en la Figura 30, RICOH mp c4504ex a una resolución de 300ppp, una resolución que alcanza cualquier escáner del mercado, incluso los de más bajas prestaciones y en escala de grises.

Figura 30. Equipo RICOH mp c4504ex.

4.1 Pruebas diseñadas.

Se ha llevado a cabo una prueba para un examen de 30 alumnos para observar y determinar la robustez y eficacia del sistema software. En primer lugar se entrenaron distintas redes neuronales seleccionando la que ofrecía mejores resultados, siendo finalmente una red de topología [784, 80, 50, 10] con un tamaño de mini-batch 10, un ratio de aprendizaje ɲ = 0,5 y un parámetro de regularización λ = 5 como se muestra en la Figura 31. Todas estas redes se entrenaron con el conjunto de entrenamiento del MNIST expandido, mencionado previamente en el apartado 3.3.4. La selección de estos hyperparametros se llevó a cabo mediante una combinación de las conclusiones obtenidas en el apartado 3.2, métodos heurísticos y ensayos de prueba y error. En este caso en concreto, se supuso que en la primera capa oculta se llevaba a cabo la activación de segmentos parciales de dígitos mientras que en la segunda capa se lleva a cabo la toma de decisión de posibles dígitos y sus formas básicas. Por ejemplo para el número 7, en la primera capa oculta se detecta si en la imagen existe un trazo

horizontal superior, un trazo horizontal medio o no y un trazo diagonal hacia el interior a lo largo de la vertical. Luego, en la segunda capa se evalúa de los posibles dígitos a un mayor nivel, en el caso del 7 si es solo una vertical, entonces sería un 1, si la vertical acaba de forma plana puede ser un 2 o un 8, etc.

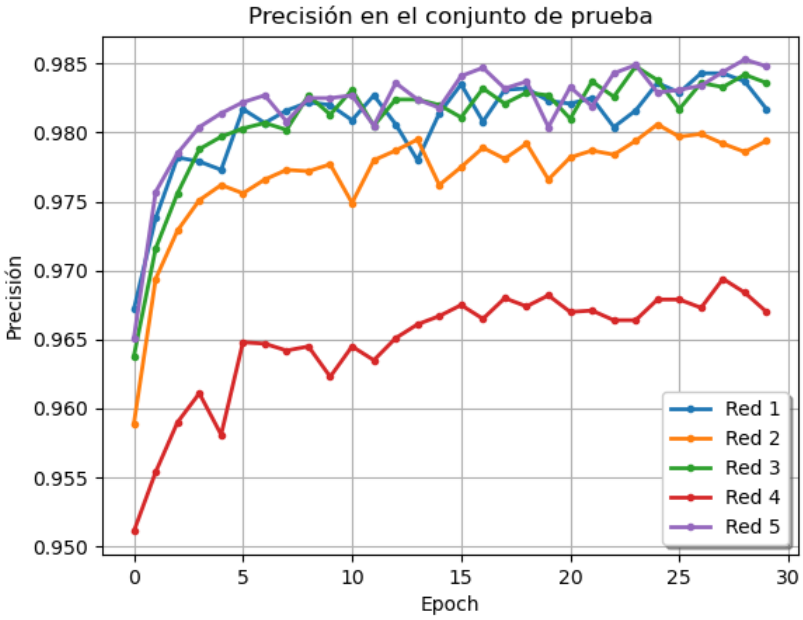


Figura 31. Desempeño de redes neuronales artificiales 2.

Como se puede observar en la Figura 31, se han alcanzado porcentajes de acierto de como máximo 98,5%, tiendo entonces un error del 1,5%. Comparando estos resultados con distintas redes y sistemas de clasificaron diseñados por científicos en este campo, como se puede ver en la Tabla 4, se alcanza unos resultados satisfactorios.

Se podría destacar el caso de la net 4, cuyo rendimiento es relativamente inferior al del resto de RNAs, con una caída del 2% aproximadamente respecto a los 250000 muestras de entrenamiento. El único factor diferenciador de cada red es su topología, pues todas poseen los mismo hyperparametros. Para este caso, su topología corresponde a [784, 20, 60, 10]. Se puede suponer que la causa de este bajo rendimiento es debido, por un lado, al tamaño de la primera capa oculta. Para el resto de redes se tiene una primera capa oculta mayor de 40. Por otro lado la relación entre las capas ocultas de la red. Mientras que la net 2 posee dos capas ocultas 40-40, las net 1, net 3 y net 5 poseen una primera capa oculta más pequeña que la segunda capa oculta, a diferencia de la net 2. A continuación se muestra una tabla con las topología de cada red.

Red	**net 1**	**net 2**	**net 3**	**net 4**	**net 5**
Topología	[784,60,30, 10]	[784,40,40,10]	[784,60,20,10]	[784,20,60,10]	[784,80,50,10]

Tabla 3. Topología de redes experimentales.

CLASSIFIER	PREPROCESSING	TEST ERROR RATE (%)	Reference
Neural Nets			
2-layer NN, 300 hidden units, mean square error	none	4.7	LeCun et al. 1998
2-layer NN, 300 HU, MSE, [distortions]	none	3.6	LeCun et al. 1998
2-layer NN, 300 HU	deskewing	1.6	LeCun et al. 1998
2-layer NN, 1000 hidden units	none	4.5	LeCun et al. 1998
2-layer NN, 1000 HU, [distortions]	none	3.8	LeCun et al. 1998
3-layer NN, 300+100 hidden units	none	3.05	LeCun et al. 1998
3-layer NN, 300+100 HU [distortions]	none	2.5	LeCun et al. 1998
3-layer NN, 500+150 hidden units	none	2.95	LeCun et al. 1998
3-layer NN, 500+150 HU [distortions]	none	2.45	LeCun et al. 1998
3-layer NN, 500+300 HU, softmax, cross entropy, weight decay	none	1.53	Hinton, unpublished, 2005
2-layer NN, 800 HU, Cross-Entropy Loss	none	1.6	Simard et al., ICDAR 2003
2-layer NN, 800 HU, cross-entropy [affine distortions]	none	1.1	Simard et al., ICDAR 2003
2-layer NN, 800 HU, MSE [elastic distortions]	none	0.9	Simard et al., ICDAR 2003
2-layer NN, 800 HU, cross-entropy [elastic distortions]	none	0.7	Simard et al., ICDAR 2003
NN, 784-500-500-2000-30 + nearest neighbor, RBM + NCA training [no distortions]	none	1.0	Salakhutdinov and Hinton, AI-Stats 2007
6-layer NN 784-2500-2000-1500-1000-500-10 (on GPU) [elastic distortions]	none	0.35	Ciresan et al. Neural Computation 10, 2010 and arXiv 1003.0358, 2010
committee of 25 NN 784-800-10 [elastic distortions]	width normalization, deslanting	0.39	Meier et al. ICDAR 2011
deep convex net, unsup pre-training [no distortions]	none	0.83	Deng et al. Interspeech 2010

Tabla 4. Sistemas de clasificación y RNAs del MNIST [8].

Tras los resultados obtenidos por las distintas redes podemos llegar a la conclusión de que le conjunto de datos de entrenamiento resulta crucial en el rendimiento obtenido por la RNA, seguido por la importancia del algoritmo de aprendizaje y finalmente unos hyperparametros adecuados para la red.

Una vez seleccionada la red neuronal artificial, se cargan sus parámetros mediante su función correspondiente y se prueba con la mencionada prueba manuscrita. Se ha habilitado la opción de visualización para mostrar por pantalla la detección de celdas en la Figura 32, las entradas introducidas a la red neuronal entrenada en la Figura

33 y posteriormente se ha modificado la hoja de cálculo mostrada en la Figura 34, en este caso de tipo Excel, para mostrar de forma más legible los resultados, como se ve en la Figura 35.

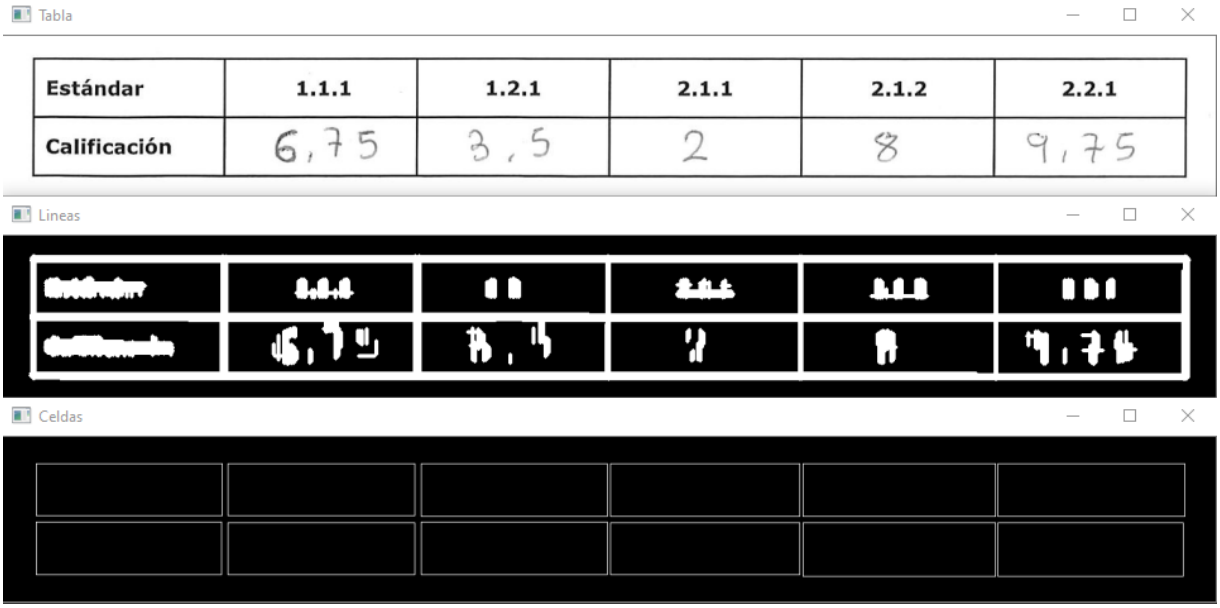
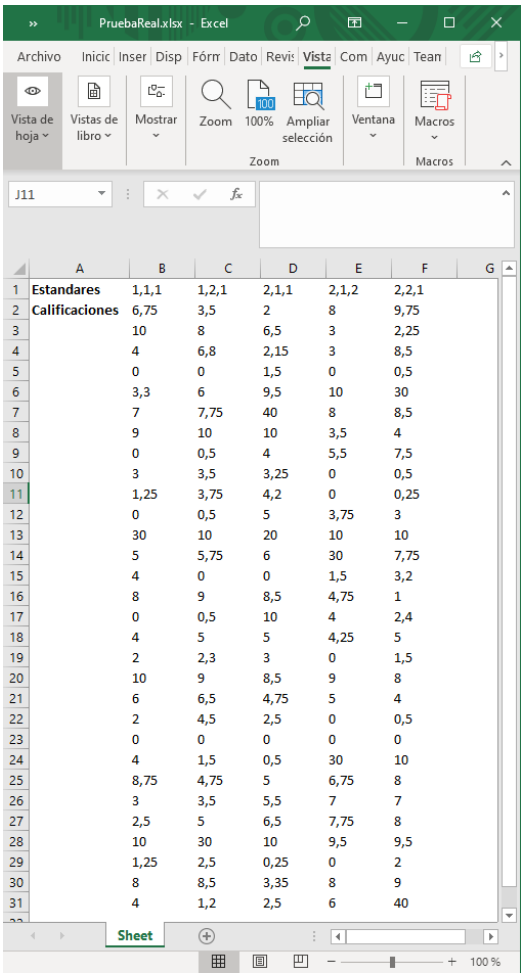


Figura 32. Ejemplo real detección de celdas.

Figura 33. Símbolos de entrada a la red.



	A	B	C	D	E	F	G
1	**Estandares**	1,1,1	1,2,1	2,1,1	2,1,2	2,2,1	
2	**Calificaciones**	6,75	3,5	2	8	9,75	
3		10	8	6,5	3	2,25	
4		4	6,8	2,15	3	8,5	
5		0	0	1,5	0	0,5	
6		3,3	6	9,5	10	30	
7		7	7,75	40	8	8,5	
8		9	10	10	3,5	4	
9		0	0,5	4	5,5	7,5	
10		3	3,5	3,25	0	0,5	
11		1,25	3,75	4,2	0	0,25	
12		0	0,5	5	3,75	3	
13		30	10	20	10	10	
14		5	5,75	6	30	7,75	
15		4	0	0	1,5	3,2	
16		8	9	8,5	4,75	1	
17		0	0,5	10	4	2,4	
18		4	5	5	4,25	5	
19		2	2,3	3	0	1,5	
20		10	9	8,5	9	8	
21		6	6,5	4,75	5	4	
22		2	4,5	2,5	0	0,5	
23		0	0	0	0	0	
24		4	1,5	0,5	30	10	
25		8,75	4,75	5	6,75	8	
26		3	3,5	5,5	7	7	
27		2,5	5	6,5	7,75	8	
28		10	30	10	9,5	9,5	
29		1,25	2,5	0,25	0	2	
30		8	8,5	3,35	8	9	
31		4	1,2	2,5	6	40	

Figura 34. Exportación de la tabla de calificaciones.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1													
2		Resultados de la RNA							Valores Reales				
3	**Estandares**	1,1,1	1,2,1	2,1,1	2,1,2	2,2,1			1,1,1	1,2,1	2,1,1	2,1,2	2,2,1
4	**Calificaciones**	6,75	3,5	2	8	9,75			6,75	3,5	2	8	9,75
5		10	8	6,5	3	2,25			10	8	6,5	3	1,25
6		4	6,8	2,15	3	8,5			4	6,8	2,15	3	8,5
7		0	0	1,5	0	0,5			0	0	1,5	0	0,5
8		3,3	6	9,5	10	30			3,3	6	9,5	10	10
9		7	7,75	40	8	8,5			7	7,75	10	8	8,5
10		9	10	10	3,5	4			9	10	10	3,5	4
11		0	0,5	4	5,5	7,5			0	0,5	4	5,5	7,5
12		3	3,5	3,25	0	0,5			3	3,5	3,25	0	0,5
13		1,25	3,75	4,2	0	0,25			1,25	3,75	4,2	0	0,25
14		0	0,5	5	3,75	3			0	0,5	5	3,75	3
15		30	10	20	10	10			10	10	10	10	10
16		5	5,75	6	30	7,75			5	5,75	6	10	7,75
17		4	0	0	1,5	3,2			4	0	0	1,5	3,2
18		8	9	8,5	4,75	1			8	9	8,5	4,75	1
19		0	0,5	10	4	2,4			0	0,5	10	4	2,4
20		4	5	5	4,25	5			4	5	5	4,25	5
21		2	2,3	3	0	1,5			2	2,3	3	0	1,5
22		10	9	8,5	9	8			10	9	8,5	9	8
23		6	6,5	4,75	5	4			6	6,5	4,75	5	4
24		2	4,5	2,5	0	0,5			1	1,5	1,5	0	0,5
25		0	0	0	0	0			0	0	0	0	0
26		4	1,5	0,5	30	10			1	1,5	0,5	10	10
27		8,75	4,75	5	6,75	8			8,75	4,75	5	6,75	8
28		3	3,5	5,5	7	7			3	3,5	5,5	7	7
29		2,5	5	6,5	7,75	8			2,5	5	6,5	7,75	8
30		10	30	10	9,5	9,5			10	10	10	9,5	9,5
31		1,25	2,5	0,25	0	2			1,25	1,5	0,25	0	2
32		8	8,5	3,35	8	9			8	8,5	3,35	8	9
33		4	1,2	2,5	6	40			1	1,2	3,5	6	10

Figura 35. Resultados de digitalización de examen.

4.2 Resultados obtenidos.

Para el análisis de resultados de la RNA, se lleva a cabo al digitalización de la tabla de calificaciones de un examen a un tipo de documento hoja de cálculo, contabilizando como acierto si todos los símbolos de una celda han sido clasificados correctamente y como fallos en caso contrario. Para un total de 155 celdas reconocidas (150 celdas correspondientes a calificaciones y las 5 primeras celdas a estándares). Como se puede ver en la Figura 35, para un total de 155 celdas, no hubo ningún fallo en la detección de la celda en la tabla de calificaciones en los 30 exámenes así como en la detección de signos de puntuación como comas o puntos. En la clasificación de los dígitos numéricos hubo un total de 14 fallos. Esto supone una tasa de acierto del 91%.

En vista de estos resultados, se puede concluir que el algoritmo para la detección de celdas mencionado en el apartado 3.4.2 resulta adecuado ya que se detectaron las 155 celdas correctamente, sin que se diera ningún falso positivo. Además también se puede afirmar lo mismo para la detección de signos de puntuación como comas o puntos, ya que tampoco hubo ningún fallo en su detección y clasificación. Todas las comas se clasificaron correctamente sin confundir ninguna con un dígito o viceversa, tampoco dándose falsos positivos.

Destaca que de los 14 fallos que se produjeron, 13 se debieron a la clasificación incorrecta del número 1. Esto puede deberse a la tipografía del calificador que puede en este caso en particular diferir en gran medida del conjunto de entrenamiento, como se puede observar en la Figura 36.

Figura 36. Distinción de tipografía. A la derecha, un 1 del docente. A la izquierda, un 1 del MNIST.

5 Conclusiones y vías de trabajos futuros.

5.1 Conclusiones.

Como ya se mencionó al principio de este trabajo, el presente trabajo busca reducir parte de la carga de trabajo del personal docente en la digitalización de calificaciones en pruebas manuscritas. Se puede concluir que se ha logrado este objetivo ya que, con el uso del sistema software expuesto, se tardan apenas unos segundos en la digitalización de todas las calificaciones de un examen. En el caso de que dicha tarea fuera realizada por un ser humano y no una inteligencia artificial, se tardaría alrededor de 6-7 minutos.

Para el caso por el cual la digitalización de un examen fuera realizada por un ser humano, se le exigiría un margen de error mínimo (o nulo en la medida de lo posible) ya que resultaría inadmisible calificar a alumnos con calificaciones incorrectas y, por tanto, se le debe exigir a una inteligencia artificial los mismos requerimientos que se exigirían a una inteligencia biológica. Teniendo en mente casos en los que el docente comete fallos en la suma o ponderación de los estándares o simplemente olvida calificar alguno, resulta debatible si es práctico y fiable utilizar una inteligencia de estas características para realizar dicha tarea ya que se alcanza una precisión del 91% mencionado en el apartado 4.2. Además siempre se presenta la opción al alumno de revisar su examen y la corrección.

No obstante, una enorme ventaja del uso del sistema predispuesto es que no presenta ningún deterioro en lo que respecta a la tasa de error con el tiempo. Depende del caso, pero es de suponer que la tasa de error de un docente crecerá a partir de las 2-3 horas de trabajo sin descanso mientras que no es así para una máquina, como es el caso.

Aun mencionado todo lo anterior, se han cumplido otros objetivos referentes a la adquisición de competencias en el campo de la inteligencia artificial y visión por computador, así como otros logros. Se ha conseguido la programación e implementación de todo un sistema software para la digitalización de calificaciones en exámenes sin el uso de bibliotecas y módulos específicos de este campo. Además se ha tratado en todo momento que la interacción por parte del usuario con el sistema fuera mínima y que los requisitos de su uso también fueran los menores posibles.

Se espera que los avances conseguidos en este trabajo sirvan para el desarrollo de nuevas técnicas y formas de automatizar la digitalización de documentos así como de inspirar el uso de la inteligencia artificial para otras aplicaciones tanto dentro como fuera del ámbito educativo.

5.2 Vías de avance futuras.

Tras la realización de este trabajo fin de grado resultan interesante distintos avances para el futuro en el desarrollo de este trabajo como se exponen a continuación.

5.2.1 Mejorar la segmentación de símbolos pertenecientes a cadenas de caracteres.

Como se ha visto a lo largo de todo el trabajo, la digitalización de calificaciones se limita únicamente a la de dígitos numéricos, excluyendo caracteres alfabéticos. Esto implica que no es posible identificar el alumno al que corresponde el examen a menos que se le asigne un cogido numérico. Resulta interesante añadir esta función de modo que se pueda identificar cada examen según un código asignado a un alumno.

Además resulta aún más interesante elaborar una RNA capaz de identificar tanto caracteres numéricos como alfabéticos de modo que sea capaz de digitalizar otra información relevante como el nombre del alumno. Para ello sería destacable hacer uso de sistemas de reconocimiento óptico de caracteres (OCR) y otras técnicas, aunque hay que remarcar el reto que supone la segmentación de palabras o el ruido generado por el dispositivo de captación de imagen.

5.2.2 Desarrollo de una aplicación completamente funcional.

Resulta muy interesante el posterior desarrollo del sistema software y su implementación en una aplicación apta para cualquier personal docente. Principalmente la adición de una interfaz gráfica intuitiva y características estéticas, así como un sistema de retroalimentación de aprendizaje de manera que una vez digitalizadas las calificaciones para un cierto caso, el docente pudiera seleccionar los casos acertados que posteriormente puedan servir como un nuevo conjunto de entrenamiento para la red.

Además para lograr unos mejores resultados independientemente de la tipografía del docente, es posible llevar a cabo un calibrado de la red haciendo uso del overfitting, mencionado en el apartado 3.3.2. Para ello se elaboraría una plantilla predeterminada donde el usuario escribiría los 10 dígitos numéricos un determinado número de veces (por ejemplo 10 veces cada dígito para un total de 100 muestras de entrenamiento) y mediante técnicas de visión artificial elaborar un nuevo conjunto de datos y ampliarlo de forma artificial. Una vez realizado este nuevo conjunto, entrenar a la RNA con este conjunto de datos sobre el entrenamiento previamente realizado con un alto número epoch y así personalizar la clasificación de los dígitos manuscritos para cada docente.

6 Referencias.

[1] Ministerio de Educación, Cultura y Deporte, «Agencia Estatal Boletín Oficial del Estado,» 1 Marzo 2014. [En línea]. Available: https://www.boe.es/buscar/doc.php?id=BOE-A-2014-2222.

[2] Ministerio de Educación, Cultura y Deporte, «Agencia Estatal Boletín Oficial del Estado,» 1 Marzo 2014. [En línea]. Available: https://www.boe.es/buscar/doc.php?id=BOE-A-2015-37.

[3] P. P. Cruz, Inteligencia artificial con aplicaciones a la ingeniería, A. Herrera, Ed., Mexico: Alfaomega, 2010.

[4] P. M. L. L. C. R. José Luis Garbi, «Reconocimiento de Números Manuscritos,» *XIII Congreso Argentino de Ciencias de la Computación,* p. 10.

[5] J. A. F. NATALIA CASILLAS GIL, *SISTEMA BASADO EN REDES NEURONALES PARA EL RECONOCIMIENTO DE DÍGITOS,* Leganés: UNIVERSIDAD CARLOS III DE MADRID, ESCUELA POLITÉCNICA SUPERIOR , 2012.

[6] E. S. J. D. M. Antonio J. Serrano, Redes neuronales artificiales, 2009.

[7] X. B. Olabe, Redes neuronales artificiales y sus aplicaciones.

[8] C. C. C. J. B. Yann LeCun, «The MINST Database of handwritten digits,» [En línea]. Available: http://yann.lecun.com/exdb/mnist/.

[9] J. D. L. H. L. L. W. X. Z. Simon S. Du, «Gradient Descent Finds Global Minima of Deep Neural Networks,» *Proceedings of Machine Learning Research,* vol. 97, p. 11, 2019.

[10] N. A. Ruhi, «Stochastic Gradient/Mirror Descent: Minimax Optimality and Implicit Regularization,» [En línea]. Available: http://www.its.caltech.edu/~nazizanr/papers/SMD.html.

[11] G. E. H. &. R. J. W. David E. Rumelhart, «Learning representations by back-propagating errors,» *Nature,* vol. 323, 1986.

[12] M. Nielsen, «Neural Networks and Deep Learning,» Diciembre 2009. [En línea]. Available: http://neuralnetworksanddeeplearning.com/index.html.

[13] A. E. Anna Sergeevna Bosman, «Measuring Saturation in Neural Networks,» 2015.

[14] M. R. S. Zhilu Zhang, «Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels,» p. 11, 2018.

[15] B. H. Navid Azizan, «STOCHASTIC GRADIENT/MIRROR DESCENT: MINIMAX OPTIMALITY AND IMPLICIT REGULARIZATION,» *ICLR,* p. 18, 2019.

[16] B. H. U. G. R. S. Antonio Valerio Miceli Barone, «Regularization techniques for fine-tuning in neural machine translation,» p. 6, 2017.

[17] Y. Bengio, «Practical Recommendations for Gradient-Based Training of Deep,» p. 33, 2012.

[18] Y. A. Jiajun Shen, «Deformable Classifiers,» p. 18, 2017.

[19] M. Nielsen, «GitHub,» 12 Marzo 2018. [En línea]. Available: https://github.com/mnielsen/neural-networks-and-deep-learning.

[20] J. A. T. Thomas M. Cover, Elements of Information Theory, John Wiley & Sons, 2012.

[21] R. R. S. A. Amit Choudharya, «Off-Line Handwritten Character Recognition using Features Extracted from Binarization Technique,» *ScienceDirect,* p. 7, 2013.

[22] Arnold Jonk, «An Axiomatric Approach To Clustering Line-segments,» *Faculty of Mathematics and computer Science, University of Amsterdam,* p. 4.

[23] L. Yan, *Recognizing Handwritten Characters,* Stanford University.

[24] H. N. K. Y. Ikuro Sato, «APAC: Augmented PAttern Classification with Neural Networks,» p. 9, 2015.

[25] «Open Source Computer Vision,» Google, [En línea]. Available: https://docs.opencv.org/master/index.html.

[26] N. D. A. D. E. P. Athanasios Voulodimos, «Deep Learning for Computer Vision: A Brief Review,» *Hindawi,* vol. 2018, nº 7068349, p. 13, 2017.

[27] L. B. Y. B. P. H. Yann LeCun, «Gradient-Based Learning Applied to Document Recognition,» *Proceedings of the IEEE,* vol. 86, nº 11, p. 48, 1998.

7 Anexos.

7.1 Anexo I: Programación de la red neuronal 1.

Para la programación de la red se ha optado por tomar parte del código de una red ya programa que se puede encontrar en [19], que posteriormente se ha modificado y actualizado a la versión de Python 3.6. El total de la programación de la red ocupa unas escasas 82 líneas de código

El primer paso es importar la librerías. En este caso se importarán únicamente las bibliotecas 'random', integrada dentro del módulo de Python 3.6 y numpy (1.19.1), la cual ya se ha mencionado en el apartado 2.5 Entorno de programación, lenguaje y módulos.

```
"""-Implementacion de la red-"""
import random
import numpy as np
```

A continuación se declara la clase que contendrá nuestro objeto tipo 'Network', junto con un método de inicialización propio. A la hora de crear una red neuronal artificial es primordial determinar el número de capas y tamaño de cada una. También es necesario estableces unos pesos y biases iniciales. Como se ha mencionado anteriormente, la red aprenderá a clasificar dígitos numéricos modificando el conjunto de pesos y biases, sin embargo es imprescindible partir de unos iniciales. Para la selección de estos pesos y biases iniciales se hará de forma aleatoria siguiendo una distribución gaussiana de media 0 y desviación estándar 1:

```
class Network(object):

  def __init__(self, sizes): #Metodo para inicializar la red neuronal.
    #Sizes--->Lista que contiene el número de neuronas en las respectivas layers (index).
    self.num_layers = len(sizes)
    self.sizes = sizes
    #Inicializacion de los weights y biases mediante una distribución gaussiana de media 0  y desviacion estandar 1.
    self.biases = [np.random.randn(y, 1) for y in sizes[1:]]
    self.weights = [np.random.randn(y, x) for x, y in zip(sizes[:-1], sizes[1:])]
```

Definimos la función logística sigmoide $\sigma(z)$:

```
def sigmoid(z): #Definicion de sigmoide.
  return 1.0/(1.0+np.exp(-z))
```

A continuación definimos un método dentro de la clase Network el cual se encargará de calcular la salida de la red 'a' a partir de un vector de entrada también 'a':

```
  def feedforward(self, a): #Metodo para obtener el output de la red a partir de un input 'a'.
    for b, w in zip(self.biases, self.weights):
      a = sigmoid(np.dot(w, a)+b)
```

```
    return a
```

El siguiente paso es definir un método 'backpropagation' que se encarga de la retro propagación del error a través de la red. Este se encarga de computar las ecuaciones (18), (19), (20) y (21) para cada capa de la red:

```
def backprop(self, x, y): #Devuelve una tupla (nabla_b, nabla_w) representando para la funcion coste C_x.
    #nabla_b and nabla_w son listas de arrays numpy, similares a self.biases y self.weights.
    nabla_b = [np.zeros(b.shape) for b in self.biases] #incializamos las listas que van a almacenar cuanto queremos que cambien los biases.
    nabla_w = [np.zeros(w.shape) for w in self.weights] #incializamos las listas que van a almacenar cuanto queremos que cambien los weights.
    # feedforward
    activation = x #Dato que obtenemos de la red (activación de la capa de salida).
    activations = [x] #Lista para almacenar las activaciones de todas las capas.
    zs = [] #Lista para almacenar las activaciones de todas las capas antes de la función de activación (sigmoide).
    for b, w in zip(self.biases, self.weights):
        z = np.dot(w, activation)+b #feedforward sin la función de activación.
        zs.append(z)
        activation = sigmoid(z)
        activations.append(activation)
    delta = self.cost_derivative(activations[-1], y) * sigmoid_prime(zs[-1]) #Calculo de delta, el error a la salida. Ecuación 17.
    nabla_b[-1] = delta #Ecuacion 19.
    nabla_w[-1] = np.dot(delta, activations[-2].transpose()) #Ecuacion 20.

    # backward---Calculo del gradiente de C.
    for l in range(2, self.num_layers): #iteracion hacia atrás en la red.
        z = zs[-l]
        sp = sigmoid_prime(z)
        delta = np.dot(self.weights[-l+1].transpose(), delta) * sp #Ecucacion 18.
        nabla_b[-l] = delta #Ecucacion 19.
        nabla_w[-l] = np.dot(delta, activations[-l-1].transpose()) #Ecucacion 20.
    return (nabla_b, nabla_w)

def cost_derivative(self, output_activations, y):
    return (output_activations-y)
```

También necesitamos un método que actualice los nuevos pesos y biases conforme vamos iterando a través del algoritmo, según se ha visto en las ecuaciones (12) y (13):

```
def update_mini_batch(self, mini_batch, eta): #Actualiza los weights y biases aplicando gradient descent mediante backpropagation a un mini-batch.
    #El "mini_batch" es una lista de tuplas "(x, y)".
    #'eta' es el ratio de aprendizaje.
    nabla_b = [np.zeros(b.shape) for b in self.biases]
    nabla_w = [np.zeros(w.shape) for w in self.weights]
    for x, y in mini_batch:
        delta_nabla_b, delta_nabla_w = self.backprop(x, y)
        nabla_b = [nb+dnb for nb, dnb in zip(nabla_b, delta_nabla_b)] #Acumulacion del cambio en los biases para cada muestra en el mini-batch.
        nabla_w = [nw+dnw for nw, dnw in zip(nabla_w, delta_nabla_w)] #Acumulacion del cambio en los weights para cada muestra en el mini-batch.
```

```
    self.weights = [w-(eta/len(mini_batch))*nw
                for w, nw in zip(self.weights, nabla_w)] #Cambio en los weights de la red.
    self.biases = [b-(eta/len(mini_batch))*nb
                for b, nb in zip(self.biases, nabla_b)] #Cambio en los biases de la red.
```

A continuación se añade el que es el núcleo de la red, el método en donde englobamos el algoritmo de aprendizaje mediante gradiente descendente estocástico:

```
def SGD(self, path, filename, training_data, epochs, mini_batch_size, eta, test_data=None): #Entrena la red neuronal usando mini-batch stochastic gradient descent.
    #El "training_data" es una lista de tuplas"(x, y)" representando el input y el output deseado.
    #Si "test_data", entonces la red se evalúa después de cada epoch imprimiendo el progreso.
    if test_data: n_test = len(test_data)
    n = len(training_data)
    for j in range(epochs):
      training_data = list(training_data)
      random.shuffle(training_data)
      training_data = tuple(training_data)
      mini_batches = [training_data[k:k+mini_batch_size] for k in range(0, n, mini_batch_size)] #Lista con los mini-batches.
      for mini_batch in mini_batches:
        self.update_mini_batch(mini_batch, eta)
      if test_data:
        print ("Epoch {0}: {1} / {2}".format(j, self.evaluate(test_data), n_test))
      else:
        print ("Epoch {0} complete".format(j))
      #Guardado de la evaluacion de la red
  f = open(path + filename, "w")
      data = {"evaluation_accuracy": evaluation_accuracy,
              "training_accuracy": training_accuracy}
      json.dump(data, f)
      f.close()
```

Por último un método para evaluar el comportamiento de la red:

```
def evaluate(self, test_data): #Devuelve el número de aciertos para un input. Se toma como elección la neurona con mayor activacion.
    test_results = [(np.argmax(self.feedforward(x)), y) #Lista con la elección de la red y el valor real
```

Una vez programada la red neuronal artificial es también necesario elaborar un script para la carga del conjunto de datos de entrenamiento del MNIST. Cargamos las librerías 'pickle' y 'gzip' para la descompresión de los datos:

```
"""-Libreria para la carga de los datos de entrenamiento, validacion y test de la red-"""

import pickle
import gzip
import numpy as np
```

A continuación definimos una función que lea los conjuntos de entrenamiento y prueba por separado:

```
def load_data(mnist):#Devuelve el MNIST data como una tuple con el training-data (50000), validation-data (10000) y test-data (10000)
```

```
    f = gzip.open('D:/Backup/Carpeta/Universidad/Curso 2019-2020/TFG/NeuronalNet/data/' + mnist, 'rb')
    training_data, validation_data, test_data = pickle.load(f, encoding = "latin1")
    f.close()
    return (training_data, validation_data, test_data)
```

Y por último añadimos una función que devuelva los datos de entrenamiento y testeo por separado:

```
def load_data_wrapper(mnist, input):
    """
    Devuelve una tupla (training_data, validation_data, test_data) basada en load_data

    "training_data" es una tuple de 50000 tuples "(x, y)"
    siendo "x" un array 2-D con la imagen (784 floats entre 0 y 1)
    e "y" un array 10-D representando el vector con el digito correspondiente.

    "validation_data" y "test_data" son tuples de 10000 tuples "(x, y)"
    siendo "x" un array 2-D con la imagen (784 floats entre 0 y 1)
    e "y" una lista con 10000 digitos correspondientes a su imagen.
    """
    tr_d, va_d, te_d = load_data(mnist)
    training_inputs = [np.reshape(x, (input, 1)) for x in tr_d[0]]
    training_results = [vectorized_result(y) for y in tr_d[1]]
    training_data = tuple(zip(training_inputs, training_results))

    validation_inputs = [np.reshape(x, (input, 1)) for x in va_d[0]]
    validation_data = tuple(zip(validation_inputs, va_d[1]))
    test_inputs = [np.reshape(x, (input, 1)) for x in te_d[0]]
    test_data = tuple(zip(test_inputs, te_d[1]))
    return (training_data, validation_data, test_data)
```

También se añade una función para modificar el formato del conjunto que resulta a conveniencia.

```
def vectorized_result(j): #Devuelve un vector 10-D con un 1 en la posición j y 0 en el resto
    e = np.zeros((10, 1))
    e[j] = 1.0
    return e
```

7.2 Anexo II: Programación de la red neuronal artificial 2.

A continuación se muestra la implementación en código de todas las mejoras en la red:

En primer lugar la declaración de las bibliotecas necesarias:

```
"""-Implementacion de una red mejorada-"""

import json
import random
import sys
import numpy as np
```

Se definen ambas funciones de coste usadas durante el desarrollo de este trabajo:

```
#Definicion de las funciones de coste cuadratica y de entropia cruzada.
class QuadraticCost(object):
  @staticmethod
  def fn(a, y):
    return 0.5*np.linalg.norm(a-y)**2 #Coste asociado.

  @staticmethod
  def delta(z, a, y):
    return (a-y) * sigmoid_prime(z) #Error Delta de la capa de salida.

class CrossEntropyCost(object):
  @staticmethod
  def fn(a, y):
    return np.sum(np.nan_to_num(-y*np.log(a)-(1-y)*np.log(1-a))) #Coste asociado.

  @staticmethod
  def delta(z, a, y): #(z--> Consistencia con la función de coste cuadratica).
    return (a-y) #Error Delta de la capa de salida.
```

Se declara el objeto de tipo Network:

```
class Network(object):

  def __init__(self, sizes, cost=CrossEntropyCost): #Metodo para inicializar la red neuronal.
    #Sizes--->Lista que contiene el número de neuronas en las respectivas layers (index).
    self.num_layers = len(sizes)
    self.sizes = sizes
    self.default_weight_initializer()
    self.cost_funct = cost
```

Funciones para la inicialización del conjunto de pesos y biases de la red:

```
  def default_weight_initializer(self):
    self.biases = [np.random.randn(y, 1) for y in self.sizes[1:]] #Inicializacion de los biases mediante.
                                    #una distribucion gaussiana de media 0 y desviacion estandar 1.
    self.weights = [np.random.randn(y, x)/np.sqrt(x)
            for x, y in zip(self.sizes[:-1], self.sizes[1:])] #Inicializacion de los weights modificado.

  def large_weight_initializer(self): #Inicializacion de los weights y biases mediante
                #una distribucion gaussiana de media 0 y desviacion estandar 1.
    self.biases = [np.random.randn(y, 1) for y in self.sizes[1:]]
    self.weights = [np.random.randn(y, x)
            for x, y in zip(self.sizes[:-1], self.sizes[1:])]
```

Un método para guardar los parámetros de la red para una posterior carga de la red sin necesidad de un nuevo entrenamiento:

```
  def save(self, path, filename): #Guarda la red neuronal.
    data = {"sizes": self.sizes,
        "weights": [w.tolist() for w in self.weights],
```

```
            "biases": [b.tolist() for b in self.biases],
            "cost": str(self.cost_funct.__name__)}
        f = open(path + filename, "w")
        json.dump(data, f)
        f.close()
```

El método correspondiente para la carga de una RNA ya entrenada:

```
def load(path, filename): #Cargar una red neuronal.
    f = open(path + filename, "r")
    data = json.load(f)
    f.close()
    cost = getattr(sys.modules[__name__], data["cost"])
    net = Network(data["sizes"], cost = cost)
    net.weights = [np.array(w) for w in data["weights"]]
    net.biases = [np.array(b) for b in data["biases"]]
    return net
```

Finalmente el método SGD modificado:

```
def SGD(self, filename, training_data, epochs, mini_batch_size, eta, lmbda = 0.0, evaluation_data=None):
        #Entrena la red neuronal usando mini-batch stochastic gradient descent.
        #El "training_data" es una tuple de tuples"(x, y)" representando el input y el output deseado.
        #Si "evaluation_data", entonces la red se evalúa después de cada epoch imprimiendo el progreso.
        #Guarda 4 listas:
        #Coste de evaluation_data por epoch,
        #La precisión de evaluation_data por epoch,
        #Coste de training_data por epoch,
        #Precision de training_data por epoch.
        if evaluation_data: n_data = len(evaluation_data)
        n = len(training_data)
        evaluation_cost, evaluation_accuracy = [], []
        training_cost, training_accuracy = [], []
        for j in range(epochs):
            training_data = list(training_data)
            random.shuffle(training_data)
            training_data = tuple(training_data)
            mini_batches = [training_data[k:k+mini_batch_size] for k in range(0, n, mini_batch_size)]
            for mini_batch in mini_batches:
                self.update_mini_batch(mini_batch, eta, lmbda, len(training_data))
            print ("Epoch {} training complete" .format(j))
            cost = self.total_cost(training_data, lmbda)
            training_cost.append(cost)
            print ("Cost on training data: {}".format(cost))
            accuracy = self.accuracy(training_data, convert=True)/n
            training_accuracy.append(accuracy)
            print ("Accuracy on training data: {}".format(accuracy))
            cost = self.total_cost(evaluation_data, lmbda, convert=True)
            evaluation_cost.append(cost)
            print ("Cost on evaluation data: {}".format(cost))
            accuracy = self.accuracy(evaluation_data)/n_data
            evaluation_accuracy.append(accuracy)
```

```
    print ("Accuracy on evaluation data: {}".format(accuracy))

#Guardado de la evaluación de la red
f = open(filename, "w")
data = {"evaluation_cost": evaluation_cost,
     "evaluation_accuracy": evaluation_accuracy,
     "training_cost": training_cost,
     "training_accuracy": training_accuracy}
json.dump(data, f)
f.close()
```