UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN
SONIDO E IMAGEN

PROYECTO FINAL DE CARRERA

SISTEMA BASADO EN REDES NEURONALES PARA EL RECONOCIMIENTO DE DÍGITOS MANUSCRITOS

AUTOR: NATALIA CASILLAS GIL

TUTOR: JESÚS ARIAS FISTEUS

Leganés, febrero de 2012

TÍTULO: *SISTEMA BASADO EN REDES NEURONALES PARA EL RECONOCIMIENTO DE DÍGITOS MANUSCRITOS.*

AUTOR: *NATALIA CASILLAS GIL*

TUTOR: *JESÚS ARIAS FISTEUS*

La defensa del presente Proyecto Fin de Carrera se realizó el día 16 de febrero de 2012; siendo calificada por el siguiente tribunal:

PRESIDENTE: *Raquel Crespo García*

SECRETARIO *Isaac Seone Pujol*

VOCAL *Marcelino Lázaro Teja*

Habiendo obtenido la siguiente calificación:

CALIFICACIÓN:

Presidente **Secretario** **Vocal**

Agradecimientos

Ahora que termina este largo trabajo y una etapa de mi vida, quiero dar las gracias a todos aquellos, familiares, amigos, profesores, que han contribuido de alguna manera a que hoy esté escribiendo esto.

Sobre todo a toda mi familia, en especial a mis padres, Ricardo y Elena y a mi hermana Tamara, por su ayuda para que yo haya llegado a este punto, por su apoyo y confianza que finalmente han hecho posible que haya llegado a la culminación de mi época universitaria.

A todos mis amigos, en especial a mis amigos de la universidad, por haber sufrido conmigo momentos de desesperación y por haber celebrado los triunfos cuando han llegado. Gracias, porque sin vosotros estos años de universidad no hubieran sido lo mismo. Habéis sido un gran apoyo.

A toda la gente que he conocido en la universidad, la gente de clase, y en especial a la gente estupenda que me encontré en la beca del Aula de Danza, he aprendido mucho de vosotros.

A Jesús, mi tutor, por haberme ayudado en este proyecto, con todo lo que conlleva.

Y finalmente a toda la gente que en algún momento ha entrado en mi vida y me ha hecho crecer como persona.

Gracias a todos.

Resumen

Este proyecto surge de la necesidad de mejorar el módulo de reconocimiento de dígitos del sistema Eyegrade. Este sistema realiza la corrección automática de exámenes tipo test mediante una cámara web. El modulo a mejorar sirve para identificar cada examen con un número de identificación del que dispone cada alumno. De esta manera la corrección es totalmente automática, sencilla de realizar e independiente de la acción humana.

En este trabajo se han estudiado diferentes alternativas por las que resolver este problema, usando la "Inteligencia Computacional", nuevas técnicas que cada vez van tomando más relevancia en el mercado para automatizar decisiones que suele hacer el ser humano.

Después de estudiar y evaluar las diferentes alternativas para el problema planteado y las diferentes opciones que había para implementarlas, se decidió optar por las redes neuronales. Se pensó en hacer primero un prototipo de la solución en un entorno que nos facilitara su implementación, ya que era una gran ventaja. Se decidió que fuera MATLAB ©. Una vez se tuvieron todos los parámetros, tanto de las imágenes como de la red neuronal, se implementó el nuevo modulo en el lenguaje de Python,y así poderlo integrar con el sistema del cual surge esta necesidad.

Las imágenes utilizadas para los experimentos, tanto en el prototipo de la solución como en el sistema final, eran reales y extraídas de exámenes de la Universidad Carlos III de Madrid, con las que se pudieron extraer los resultados finales.

Abstract

This project emerges from the necessity to improve the digit recognition module Eye-grade system. This system executes the automatic correction for multiple-choice exams through a web-cam. The module at improving is useful to identify each exam with an ID number which belongs to each student. In this way, the correction is fully automatic, easy to perform and independently of human action.

In this work, different alternatives have been studied to solve this problem, using "soft computing´´, new techniques which are becoming more relevant in the market to automate decisions that often are done by humans.

After studying and evaluating the different alternatives for the problem and the various options that had to be implemented, the chosen option was the neural networks. Firstly, a prototype of the solution was made in an environment that we provide implementation, as it was a great advantage. MATLAB©was the chosen one. When all the parameters were taken not only the images but the neural network as well, it was implemented in the Python language, for integrating with the system.

Both the prototype of the solution and the final system, were performed with real images from exams of the Carlos III University of Madrid. The final results were obtained with the mentioned images.

Índice general

1. INTRODUCCIÓN **19**

1.1. Motivación 19

1.2. Objetivos 21

1.3. Plan de trabajo 22

1.4. Estructura del proyecto 22

2. ESTADO DEL ARTE **25**

2.1. Inteligencia Computacional 25

2.1.1. Historia 26

2.2. Redes Neuronales 28

2.2.1. Introducción 30

2.2.2. Modelo de Neurona 31

2.2.3. Clasificación 34

2.3. Software para implementar las Redes Neuronales 38

2.3.1. MATLAB© 39

2.3.2. Python 40

2.4. Eyegrade 40

3. REQUISITOS **43**

3.1. Funcionales 43

3.2. No funcionales 43

4. PROTOTIPO DE LA SOLUCIÓN **45**

4.1. Introducción 45

4.2. Segmentación 47

4.3. Preprocesamiento 47

4.4. Extracción de características 50

4.5. Reconocimiento 53

4.6. Experimentos realizados y resultados 58

4.6.1. Descripción de los datos 59

4.6.2. Experimento 1 59

4.6.3. Experimento 2 62

4.6.4. Experimento 3 63

4.6.5. Experimento 4 66

4.6.6. Experimento 5 67

4.6.7. Experimento 6 69

4.6.8. Conclusión 71

5. SOLUCIÓN **73**

5.1. Introducción 73

5.2. Procesamiento de las imágenes 74

5.3. Extracción de características 75

5.4. Red Neuronal 78

5.5. Integración con Eyegrade 80

5.6. Resultados 81

5.6.1. Comparación con el prototipo de la solución 81

5.6.2. Comparación con el modulo de reconocimiento de dígitos del sistema Eyegrade 84

6. CONCLUSIONES Y LÍNEAS FUTURAS DE TRABAJO **85**

6.1. Conclusiones 85

6.2. Líneas Futuras de trabajo 87

APÉNDICES **91**

A. PRESUPUESTO DEL PROYECTO **91**

Lista de Figuras

2.1. Evolución del error de aprendizaje y del error de generalización 29

2.2. Iteracción entre una nuerona presináptica y otra postsináptica [1] 32

2.3. Funciones de activación habituales 34

2.4. Arquitectura unidireccional de tres capas, de entrada, oculta y de salida . . 36

2.5. Clasificación de las redes neuronales por el tipo de aprendizaje y la arquitectura 37

4.1. Ejemplo relleno de un examen 46

4.2. Diagrama del reconocedor óptico de caracteres (OCR) 46

4.3. Ejemplo de segmentación de la identificación de un alumno 48

4.4. Aplicación de un Filtro de Mediana 48

4.5. Aplicación de la operación morfológica de cierre 49

4.6. Aplicación de la operación morfológica de esqueletización 49

4.7. Normalización de la imagen 50

4.8. Arquitectura del MLP 54

4.9. Arquitectura de la red neuronal usada en el Experimento 1 59

4.10. Representación del error en el entrenamiento para la red del Experimento 1 60

4.11. Representación del error en el entrenamiento utilizando el procedimiento de validación cruzada para la red del Experimento 1 62

4.12. Representación del error en el entrenamiento utilizando el procedimiento de validación cruzada para la red del Experimento 2 63

4.13. Arquitectura de la red neuronal empleada en el Experimento 3 64

4.14. Representación del error en el entrenamiento utilizando el procedimiento de validación cruzada para la red del Experimento 3 64

4.15. Representación del error en el entrenamiento utilizando el procedimiento de validación cruzada para la red del Experimento 4 66

4.16. Arquitectura de la red neuronal empleada en el Experimento 5 68

4.17. Representación del error en el entrenamiento utilizando el procedimiento de validación cruzada para la red del Experimento 5 68

4.18. Arquitectura de la red neuronal empleada en el Experimento 6 70

4.19. Representación del error en el entrenamiento utilizando el procedimiento de validación cruzada para la red del Experimento 6 70

5.1. Ejemplo de una imagen dividida en las 9 partes 77

5.2. Ejemplo de la aplicación a una imagen de la esqueletización en ambos sistemas 82

5.3. Representación del error en el entrenamiento para la red del prototipo de la solución con las 12 características extraídas desde Python 83

Lista de Tablas

2.1. *Comparación entre cerebro y computador convencional*[1] 30

4.1. *Tasa de acierto de la red neuronal del Experimento 1* 61

4.2. *Tasa de acierto de la red neuronal utilizando el procedimiento de validación cruzada del Experimento 1* 61

4.3. *Tasa de acierto de la red neuronal utilizando el procedimiento de validación cruzada del Experimento 3* 65

4.4. *Proporción de dígitos en las muestras de entrenamiento* 65

4.5. *Tasa de acierto de la red neuronal utilizando el procedimiento de validación cruzada del Experimento 4* 67

4.6. *Tasa de acierto de la red neuronal añadiendo dos características a las ya existentes* 69

A.1. Fases del Proyecto 92

A.2. Costes de material 92

A.3. Presupuesto 93

Capítulo 1

INTRODUCCIÓN

La pretensión del primer capítulo de este proyecto es dar una visión general del contexto donde se enmarca el problema que se va a resolver y un esbozo de la propuesta para resolverlo. Además, se exponen los objetivos que se van a intentar alcanzar, así como la estructura del proyecto con una breve descripción de cada capítulo.

1.1. Motivación

Cuando se tiene información en documentos impresos o en otro formato como imágenes y se quiere procesarla digitalmente para tratarla por ejemplo en un editor de texto, dicha información debe ser convertida previamente a un formato digital. Para ello podemos introducirla a través de un teclado, situación muy pesada y tediosa, o automatizar esta operación mediante un sistema de reconocimiento óptico de caracteres (OCR). Esta última opción sería la mejor, ya que facilitaría mucho el trabajo en bastantes situaciones de la vida cotidiana y además evitaría los errores humanos que se pueden cometer al introducir la información mediante teclado.

El reconocimiento óptico de caracteres tiene como objetivo asignar una imagen a un símbolo perteneciente a un conjunto denominado alfabeto. Se distingue entre el recono-

cimiento de información tipográfica y manuscrita. En la primera los caracteres de una tipografía van a tener siempre la misma forma aunque cambie el tamaño por ejemplo. En cambio los caracteres manuscritos dependerá de la caligrafía de cada persona, ya que aunque escriba un carácter varias veces nunca serán iguales. Por eso la complejidad del sistema de reconocimiento dependerá del tipo de caracteres que se vayan a analizar.

Existen numerosas aplicaciones que utilizan el reconocimiento óptico de caracteres tanto manuscritos como tipográficos. Algunos ejemplos son el reconocimiento de los dígitos de las matrículas de vehículos, talones bancarios, formularios escritos a mano,...

Este proyecto surge de la idea de mejorar el módulo de reconocimiento de dígitos de la aplicación Eyegrade implementada en Python. Esta aplicación corrige exámenes tipo test. Las pruebas piloto del sistema se han realizado con exámenes de alumnos de la Universidad Carlos III de Madrid en diferentes asignaturas. Estos alumnos deben escribir en una plantilla su número de identificación (NIA) y elegir entre cuatro respuestas la correcta marcándola con un aspa (X). La aplicación reconoce de manera razonable las respuestas de los alumnos, pero la tasa de error del reconocimiento de los dígitos puede ser mejorable, ya que el algoritmo utilizado para ello no es el más adecuado. Lo que se va a tratar de hacer es disminuir dicha tasa con uno de los sistemas que existen en estos momentos como las redes neuronales.

Hoy en día coexisten dos corrientes importantes dentro de la búsqueda de sistemas o máquinas inteligentes que son complementarias. Por una parte la Inteligencia Artificial convencional, basada en algoritmos manipuladores de información simbólica y que operan sobre la base de la lógica digital. Y por otra parte la Inteligencia Computacional o *soft computing*, donde se agrupan las redes neuronales, los sistemas borrosos y otras técnicas que tratan de imitar construcciones desarrolladas por la naturaleza. Las más populares y desarrolladas en estos momentos son las redes neuronales ya que estos sistemas imitan a la estructura del cerebro para tratar de reproducir algunas de sus capacidades.

Como MATLAB es uno de los lenguajes matemáticos más desarrollados en estas técnicas se implementarán varias redes neuronales modificando sus parámetros y entrenándolas

con las imágenes reales de exámenes. Después se compararán los resultados y la red cuyos resultados sean los mejores será implementada en Python para añadirlo a la aplicación descrita anteriormente mejorando así los resultados del reconocimiento de dígitos del NIA.

1.2. Objetivos

Con este proyecto se busca desarrollar y evaluar distintas redes neuronales en lenguaje MATLAB e implementar la que tenga mejores resultados en lenguaje Python mejorando la aplicación Eyegrade, ya que ayudará a reconocer los dígitos del número de identificación personal (NIA) de manera más rápida y efectiva.

Por tanto los objetivos específicos son:

- Segmentar las imágenes digitalizadas de los exámenes para extraer cada número de manera individual.
- Procesar la imagen de cada dígito y tratar dicha imagen (recortar la región de interés, normalizarla,...)
- Definir los métodos para la extracción de las características de las imágenes.
- Extracción de las características de las imágenes.
- Evaluar los resultados obtenidos y extraer conclusiones.
- Implementar en Python los métodos de extracción de las características y la red neuronal.
- Integrar el reconocedor de dígitos manuscritos en Python y evaluar los resultados comparándolos con los anteriores.

1.3. Plan de trabajo

El plan de trabajo en que se va a dividir el proyecto son 4 fases, las cuales se detallan a continuación:

1. Estudio y familiarización con el sistema Eyegrade y con el problema que se pretende resolver: se estima que tendrá una dedicación de 1 persona-mes. Esta parte consistirá en saber como funciona el sistema que se intentará mejorar y buscar documentación para llevar a cabo una solución.

2. Desarrollo del software: esta fase, y la siguiente, son la parte central del proyecto. Se intentará que no suponga un esfuerzo de más de 1 persona-mes. Esta parte tratara de poner en práctica la solución encontrada en la documentación.

3. Análisis de los datos: esta parte es la que se llevará la mayor parte del tiempo, ya que consiste en comparar los datos obtenidos con el anterior sistema. En esta fase se emplearan aproximadamente 4 personas-mes.

4. Redacción de la memoria del proyecto: consistirá en llevar a papel todo lo realizado en el trabajo, desde la documentación a las últimas pruebas realizadas. En esta fase se dedicará un esfuerzo de alrededor de 1 personas-mes.

1.4. Estructura del proyecto

La memoria de este proyecto esta dividida en 6 capítulos, cada uno de los cuales se centra en un aspecto concreto del proyecto. La estructura y contenido de cada uno de ellos se describe a continuación:

- En este capítulo, *Introducción* se pretende dar una visión global del contexto en el que se desarrolla el proyecto. Se introduce brevemente la idea que se plantea, así como el marco del que nace. También, se cita escuetamente la base teórica que se

va a utilizar para resolver el problema. Se exponen los objetivos propuestos y se describe brevemente el contenido de los capítulos siguientes, junto con la estructura general del documento.

- En el capítulo 2, *Estado del Arte*, se describen con un mayor nivel de detalle los conceptos teóricos que han servido de base para la realización del proyecto. En la primera parte se detallan los paradigmas utilizados centrándose en las redes neuronales. En la segunda parte se centra en la descripción del software utilizado para desarrollar el proyecto. La última parte se describe la herramienta con la que se va a integrar y que se va a tratar de mejorar.

- En el capitulo 3, *Requisitos*, se detallan los requerimientos que tiene el proyecto, tanto funcionales sin los cuales no se puede llevar a cabo el proyecto, como no funcionales para complementar a los anteriores.

- En el capítulo 4, *Prototipo de la solución*, es junto con el siguiente capítulo el eje central del proyecto ya que describe todo el trabajo llevado a cabo. En el capítulo se describen todos los pasos que se han seguido para llegar a la primera solución del problema o, denominado en el trabajo como prototipo de la solución. En él se detallan y justifican todas las decisiones tomadas y los resultados obtenidos con cada una de ellas. También, se llega a una solución que será la que se adapte e implemente en el sistema para el que estamos desarrollando este proyecto.

- En el capítulo 5, *Solución*, se detalla la implementación en Python de la solución escogida en el anterior capítulo. Se explican todos los pasos que se han implementado así como de la integración en el sistema Eyegrade y finalmente, los resultados que se obtienen.

- En el capítulo 6, *Conclusiones y líneas futuras de trabajo*, se hace un breve resumen de todo lo realizado en el trabajo, así como una valoración de los resultados obtenidos. Además, se marcan las pautas para seguir y mejorar los resultados.

Finalmente, el documento concluye con un apéndice en el que se detalla el presupuesto del proyecto.

Capítulo 2

ESTADO DEL ARTE

En este capítulo se introducen los conceptos teóricos sobre los que se apoya el proyecto, así como una breve introducción sobre el *software* utilizado para implementarlo. También, se detallan los paradigmas en los que se basa el proyecto para resolver el problema propuesto en el primer capítulo. Además, se resumen las principales características de la tecnología empleada, las redes neuronales, y se da una visión general de la herramienta que se va a mejorar con el desarrollo del proyecto.

2.1. Inteligencia Computacional

El paradigma de procesamiento de la información desarrollado a finales del S.XIX y principios del S.XX es la base de los actuales sistemas de procesamiento digitales. Sin embargo, este esquema tiene problemas a la hora de abordar tareas donde la información se presenta de manera masiva, imprecisa y distorsionada.

Cada vez es más evidente el cambio en la orientación de la Ciencia, desde las ciencias de lo natural a las ciencias de lo artificial; de la observación y el análisis a la manipulación y la síntesis; de la precisión de la computación a la imprecisión del mundo real.

Conforme más se avanza hacia una era de máquinas inteligentes y automatización del

razonamiento, más evidente resulta la necesidad de que se refleje en estas metodologías la capacidad del ser humano de tomar decisiones en un entorno de imprecisión e incertidumbre, cuyo último fin es diseñar y construir máquinas con un coeficiente de inteligencia elevado.

Durante el siglo pasado comenzó a desarrollarse nuevas técnicas que se inspiraban en las soluciones que la naturaleza había encontrado a lo largo de millones de años de evolución para el tratamiento de información masiva y distorsionada procedente del entorno natural. A esto se le dio el nombre de inteligencia computacional o *soft computing*, cuyas principales ramas son la lógica borrosa, las redes neuronales, la computación evolutiva, la computación probabilística, el caos y el aprendizaje en máquinas [1].

Hoy en día cada vez se invierten más esfuerzos en la investigación dentro de este campo. Las dos técnicas que están causando mayor impacto son las redes neuronales y los sistemas borrosos, que pese a su juventud son ampliamente utilizadas ya que permiten incorporar cierta inteligencia en sistemas de procesamiento y control. Las redes neuronales artificiales mediante una computación paralela, distribuida y adaptativa son capaces de aprender a partir de ejemplos reproduciendo algunas de las capacidades del cerebro humano. Los sistemas borrosos se introducen para manejar conceptos vagos e imprecisos como los empleados en la vida cotidiana y que nuestro cerebro está acostumbrado a manejar [2].

Por tanto, se espera que estas tecnologías se extiendan cada día más y desempeñen un papel importante en la construcción de máquinas que emulen la capacidad humana de tomar decisiones en entornos imprecisos y con ruido.

2.1.1. Historia

Los primeros intentos hacia la construcción de máquinas inteligentes comienzan en la Segunda Guerra Mundial [3], con el diseño de ordenadores analógicos que controlaban cañones antiaéreos. Esto, junto con los conocimientos en los sistemas nerviosos de los seres vivos dieron lugar a máquinas capaces de responder como los animales. A este estudio se

le acuñó el término de **cibernética** por Norbert Wiener.

Tras dos décadas de vigencia, comienza el desarrollo de los ordenadores digitales basados en la separación de *hardware* y *software*, corriente que se denominó **computación algorítmica**. Un punto clave fue en 1937 con la máquina ideal de Alan Turing más conocida como máquina de Turing [4]. Siguiendo sus pasos el húngaro John Von Neumann concibe una computadora basada en la lógica digital que ejecuta en serie las instrucciones que componen un algoritmo almacenado en memoria. Debido a su eficacia, el binomio lógica booleana-máquina de Von Neumann, es la base sobre la que se asientan la mayor parte de los computadores digitales actuales.

A finales de los años 50, se continuó trabajando en este sentido diseñando programas que permitieran al ordenador razonar. En 1960, John McCarthy llamó **Inteligencia Artificial** a los métodos algorítmicos capaces de hacer pensar a los ordenadores. La principal desventaja de estos algoritmos es que solo eran capaces de resolver aquellos problemas para los que habían sido diseñados.

Estos resultados eran tan alentadores que se creía que en una década se conseguiría una máquina realmente inteligente. Sin embargo, los ordenadores actuales son miles de veces más potentes y no resultan mucho más inteligentes. El problema reside en que el binomio lógica booleana-máquina de Von Neuman tiene problemas a la hora de abordar información masiva, imprecisa y distorsionada como la del *mundo real*. Para solucionarlo se han vuelto a retomar paradigmas como las redes neuronales, sistemas borrosos, algoritmos genéticos o computación alternativa.

Por tanto, el resurgimiento de las redes neuronales también denominadas sistemas neuronales artificiales o ANS se debe a la dificultad citada anteriormente, pudiéndose mostrar su potencial con la simulación de estos sistemas gracias al desarrollo de la integración VLSI [5], que utilizan sistemas de cálculo en paralelo, ya que la máquina de Von Neumann no es la apropiada [6]. Además, se encontró la forma de entrenar un perceptrón multicapa [7] resolviendo los problemas y las objeciones a los ANS [8].

Por tanto, existen dos corrientes complementarias en la búsqueda de la inteligencia. Por

un lado la *Inteligencia Artificial* que se basa en algoritmos ejecutados sobre ordenadores de Von Neumann y lógica digital, y por otro la *Inteligencia Computacional* que imitan las construcciones desarrolladas por la naturaleza [9].

2.2. Redes Neuronales

Las Redes Neuronales Artificiales o ANS, fueron originalmente una simulación abstracta de los sistemas biológicos, formados por un conjunto de unidades, llamadas neuronas, interconectadas unas con otras. Su principal objetivo era simular, mediante algoritmos, las tareas cognitivas en las cuales el ser humano se desenvuelve muy bien.

Existen dos fases en toda aplicación de las redes neuronales: la fase de aprendizaje o entrenamiento y la fase de prueba. En la fase de entrenamiento, se usa un conjunto de datos o patrones de entrenamiento para determinar los pesos (parámetros de diseño) que definen el modelo neuronal. Una vez entrenado el modelo y con sus parámetros definidos y fijos, se usará en la fase de prueba, en la que se procesan los patrones de prueba que constituyen la entrada habitual de la red, analizándose las prestaciones definitivas del modelo neuronal.

- **Fase de Entrenamiento** Una característica de las redes neuronales es su capacidad de aprender mediante la actualización o cambio de los pesos sinápticos que caracterizan a las conexiones. Los pesos son adaptados de acuerdo a la información extraída de los patrones de entrenamiento nuevos que se van presentando, y pueden ser calculados de una vez o adaptados iterativamente, según el tipo de red neuronal. Normalmente, los pesos óptimos se obtienen optimizando alguna función de energía.
- **Fase de Prueba** Una vez calculados y fijados los pesos de la red, se prueba la generalización de la red neuronal comparándose la salida con los valores esperados.

Uno de los aspectos fundamentales de los ANS es su capacidad de generalizar a partir de ejemplos, lo que constituye el problema de la memorización frente a la generalización.

Por generalización se entiende la capacidad de la red de dar una respuesta correcta ante patrones que no han sido empleados en su entrenamiento.

Por tanto, podemos entrenar una red neuronal haciendo uso de un conjunto de aprendizaje, y comprobar su eficacia real, o error de generalización, mediante un conjunto de red. Si representamos el error de aprendizaje y el error de generalización durante el transcurso del aprendizaje, se obtiene una gráfica como la presentada en la Figura 2.1. Como se observa el aprendizaje empieza a disminuir monótonamente, mientras que el error de generalización a partir de cierto punto empieza a incrementarse. Esto indica que el sistema se ajusta demasiado a los patrones de entrenamiento, es decir está memorizando el conjunto de aprendizaje, lo que técnicamente se denomina **sobreajuste**. Idealmente, dada una arquitectura de red neuronal, ésta debería entrenarse hasta un punto óptimo en el que el error de generalización es mínimo. Este procedimiento se denomina validación cruzada y es ampliamente utilizado en la fase de desarrollo las redes neuronales.

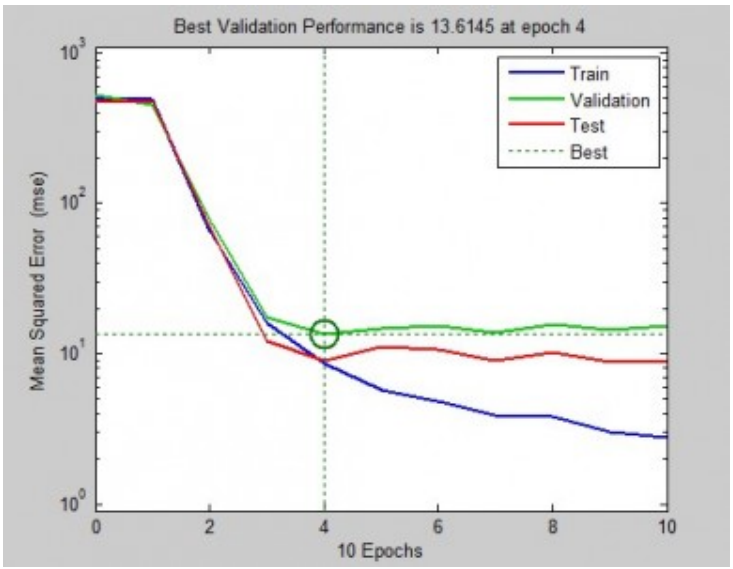


Figura 2.1: Evolución del error de aprendizaje y del error de generalización

Por tanto, para entrenar y validar una red neuronal se necesitan 3 conjuntos: de entrenamiento, para el ajuste de los parámetros de la red; de validación, que se emplean para realizar la validación cruzada y de test, que determinan de manera objetiva el rendimiento final de la red. Una proporción de estos conjuntos respecto de todos los patrones

disponibles suele ser 60 %, 20 % y 20 % respectivamente.

2.2.1. Introducción

Las redes neuronales artificiales comienzan a finales del siglo XIX con el estudio sobre el sistema nervioso [10], donde se demostró que estaba compuesto por células individuales, *las neuronas*, ampliamente interconectadas entre sí. Las neuronas constituyen procesadores sencillos con un canal de entrada, un órgano de cómputo y un canal de salida, componiéndose el cerebro de millones de estos procesadores elementales trabajando en paralelo y conformando redes de neuronas.

Las neuronas aprenden a partir de las señales que reciben de su entorno e influyéndose mutuamente a través de sinapsis que causan una compleja dinámica de activaciones y desactivaciones en ellas. En definitiva, las neuronas se autoorganizan aprendiendo del entorno y adaptándose a él, y hacen del cerebro un sistema de procesamiento, no lineal, adaptativo, que trabaja en paralelo. Se puede encontrar una comparación de sus características frente a las de un computador convencional en la Tabla 2.1.

Características	**Cerebro**	**Computador**
Velocidad de proceso	$10^{-2} seg.(100 Hz.)$	$10^{-9} seg.(1 GHz)$
Estilo de procesamiento	paralelo	secuencial
Número de procesadores	$10^{11} - 10^{14}$	pocos
Conexiones	10000 por procesador	pocas
Almacenamiento del conocimiento	distribuido	direcciones fijas
Tolerancia a fallos	amplia	nula
Tipo de control del proceso	auto-organizado	centralizado

Tabla 2.1: *Comparación entre cerebro y computador convencional*[1]

Por tanto, la idea de partida de las redes neuronales artificiales es que para ejecutar el tipo de tareas donde el cerebro se desenvuelve con eficacia puede resultar interesante imitar la estructura *hardware*, creando un sistema compuesto por múltiples neuronas interconectadas y estudiar si a partir de su autoorganización pueden reproducirse sus capacidades.

Los tres conceptos clave de los sistemas nerviosos, que se pretenden emular en las redes neuronales artificiales son:

- *Procesamiento en paralelo*: el cálculo en paralelo de sus neuronas hace que se obtengan más rápidamente los resultados esperados que si se realizara la misma operación de manera secuencial.
- *Memoria distribuida*: aunque una parte de la memoria resulte dañada solo se pierde una parte pequeña de información, que además con la redundancia existente en las neuronas es fácilmente recuperable.
- *Adaptabilidad al entorno*: modifican su sinapsis y aprenden de ejemplos generalizando a partir de ellos.

2.2.2. Modelo de Neurona

Aunque el comportamiento de algunos sistemas neuronales reales sea lineal, en general, esto no sucede, siendo en su mayoría de tipo no lineal. Esta característica fue contemplada desde la neurona original de McCulloch-Pitts [11] y es una de las características más destacables y de mayor interés, ya que este tipo de problemas no suele ser fácilmente abordables.

La descripción de la estructura genérica de la neurona artificial comienza con la definición de neurona. Una neurona es un procesador elemental de cálculo que a partir de un vector de entradas procedentes del exterior o de otras neuronas, proporciona una única respuesta de salida [1]. Los elementos que constituyen la neurona son los siguientes:

- **Entradas y salidas.**

  Las variables de entrada y salida pueden ser binarias (digitales) o continuas (analógicas), dependiendo del modelo y aplicación.

  Dependiendo del tipo de salida, las neuronas suelen recibir nombres específicos [12]. Así, las neuronas cuya salida sólo puede tomar valores 0 o 1 se suelen denominar

genéricamente **neuronas de tipo McCulloch-Pitts**, mientras que aquéllas que únicamente pueden tener por salidas -1 o +1 se suelen denominar **neuronas tipo Ising**. Si pueden obtener diversos valores discretos en la salida se dice que se trata de una **neurona de tipo Potts**.

- **Regla de propagación.**

  La regla de propagación permite obtener, a partir de las entradas y los pesos sinápticos el valor potencial postsináptico h_i de la neurona.

$$h_i(t) = \sigma_i(w_{ij}, x_j(t)) \tag{2.1}$$

Los pesos sinápticos de la neurona i, w_{ij}, representan la intensidad de interacción entre la neurona presináptica j (las que envían las señales) y la neurona postsináptica i (las que reciben las señales)(véase la Figura 2.2).Dada una entrada positiva, si el peso es positivo tenderá a excitar a la neurona postsináptica, si el peso es negativo tenderá a inhibirla.

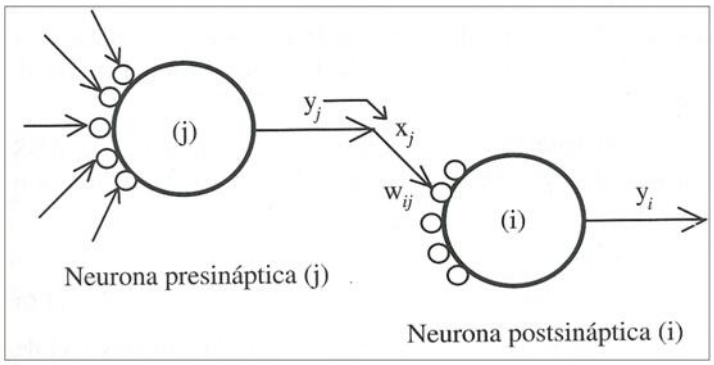


Figura 2.2: Iteracción entre una nuerona presináptica y otra postsináptica [1]

La función más habitual es de tipo lineal y se basa en la suma ponderada de las entradas con los pesos sinápticos.

$$h_i(t) = \sum_j w_{ij} x_j \tag{2.2}$$

Otra regla de propagación habitual, especialmente en los modelos de ANS basados en el cálculo de distancias entre vectores es la **distancia euclídea.**

$$h_i^2(t) = \sum_j (x_j - w_{ij})^2 \tag{2.3}$$

- **Función de activación o de transferencia.**

  La función de activación proporciona el estado de activación actual $a_i(t)$ a partir del potencial postsináptico $h_i(t)$ y del propio estado de activación anterior $a_i(t-1)$.

$$a_i(t) = f_i(a_i(t-1), h_i(t)) \tag{2.4}$$

  Sin embargo, en muchos modelos de ANS se considera que el estado actual de la neurona no depende de su estado anterior, sino únicamente del actual quedando la ecuación 2.4 como:

$$a_i(t) = f_i(h_i(t)) \tag{2.5}$$

  La función de activación $f(.)$ se suele considerar determinista, y en la mayor parte de los modelos es monótona creciente y continua. La forma $y = f(x)$ de la funciones de activación más empleadas en los ANS se muestra en la Figura 2.2.2. La más simple de todas es la función identidad. Otro caso muy simple es la función escalón, empleadas en redes como el Perceptrón Simple o Hopfield.

  En ocasiones, los algoritmos de aprendizaje requieren que la función de activación cumpla la condición de ser derivable. Las más empleadas en este sentido son las funciones de tipo sigmoideo o la gaussiana.

- **Función de salida.**

  Esta función proporciona la salida global de la neurona $y_i(t)$ en función de su estado de activación actual $a_i(t)$. Normalmente la función de salida es la identidad $F(x) = x$, de modo que el estado de activación de la neurona se considera la propia salida.

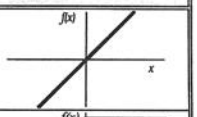
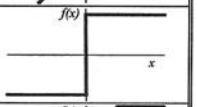
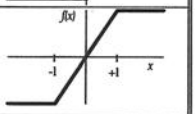
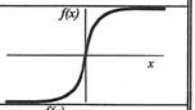
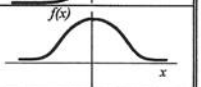
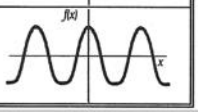
	Función	Rango	Gráfica
Identidad	$y = x$	$[-\infty, +\infty]$	
Escalón	$y = sign(x)$ $y = H(x)$	$\{-1, +1\}$ $\{0, +1\}$	
Lineal a tramos	$y = \begin{cases} -1, & si\ x < -l \\ x, & si\ +l \leq x \leq -l \\ +1, & si\ x > +l \end{cases}$	$[-1, +1]$	
Sigmoidea	$y = \frac{1}{1+e^{-x}}$ $y = tgh(x)$	$[0, +1]$ $[-1, +1]$	
Gaussiana	$y = Ae^{-Bx^2}$	$[0,+1]$	
Sinusoidal	$y = A\,\mathrm{sen}(\omega x + \varphi)$	$[-1,+1]$	

Figura 2.3: Funciones de activación habituales

[1]

$$y_i(t) = F_i(a_i(t)) = a_i(t) \tag{2.6}$$

La función de salida puede ser también de tipo escalón, lo que supone que la neurona no se dispare hasta que supere cierto umbral. Otra de las funciones utilizadas es la estocástica, con lo que la neurona tendrá un comportamiento probabilístico.

2.2.3. Clasificación

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

Modos de aprendizaje

El aprendizaje puede definirse como el proceso por el que se ajustan los pesos de la red a partir de una información de entrada y el entorno que la rodea [13]. El tipo de aprendizaje vendrá determinado por la forma en que dichos pesos son adaptados.

El entrenamiento o aprendizaje se puede llevar a cabo a dos niveles. El más convencional consiste en modificar los pesos sinápticos siguiendo una regla de aprendizaje, construida a partir de la optimización de una función de coste, que mide la eficacia actual de la operación.

$$\triangle w_{ij}(t+1) = w_{ij}(t) + \triangle w_{ij}(t) \tag{2.7}$$

Este proceso se repite hasta que la neurona alcanza el rendimiento deseado.

Otros modelos incluyen otro nivel en el aprendizaje, la creación o destrucción de neuronas, modificando la propia arquitectura de la red.

Los dos tipos básicos de aprendizaje:

- **Aprendizaje supervisado.** Se presentan a la red un conjunto de patrones, junto con la salida deseada, e iterativamente se ajustan los pesos hasta que la salida tiende a ser la deseada utilizando la información detallada del error que comete en cada paso.

- **Aprendizaje no supervisado o autoorganizado.** Se presentan a la red multitud de patrones sin adjuntar la respuesta que deseamos. La red, por medio de la regla de aprendizaje, estima la función de probabilidad, $p(\mathbf{x})$, a partir de la cual se reconocen regularidades en el conjunto de entrada, se extraen rasgos y se agrupan patrones según su similitud.

- **Aprendizaje híbrido.** En este caso coexisten los dos tipos de aprendizaje anteriores, supervisado y no supervisado, los cuales tienen lugar en distintas capas de las

neuronas.

- **Aprendizaje reforzado.** Se sitúa entre el aprendizaje supervisado y en autoorganizado. En este caso, se emplea información sobre el error cometido, pero en una única señal que representa el índice global del rendimiento de la red. Además no se suministra la salida deseada.

Topología de red

Se denomina topología o arquitectura de red a la organización y disposición de las neuronas en la red formando capas más o menos alejadas de la entrada y salida de la red. Los parámetros fundamentales de la red son:

- **Número de neuronas por capa.** Las neuronas se suelen agrupar en unidades estructurales que denominaremos capas. Se distinguen tres tipos de capas. Una capa de entrada que está compuesta por neuronas que reciben datos. Una capa de salida que es aquella cuyas neuronas proporcionan la respuesta de la red neuronal. Una capa oculta que es aquella que no tiene conexión directa con el entorno, pero proporciona a la red grados de libertad adicionales (ver Figura 2.2.3).

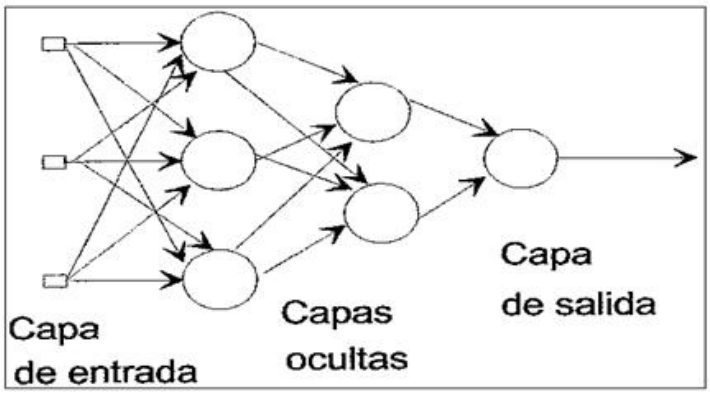


Figura 2.4: Arquitectura unidireccional de tres capas, de entrada, oculta y de salida [1]

- **Número de capas.**

- *Redes monocapa*: compuesta por una única capa de neuronas.
- *Redes multicapa*: compuesta por varias capas de neuronas.

- **Grado de conectividad.**

  - *Conexión inhiboria*: con un peso sináptico negativo.
  - *Conexión excitatoria*: con un peso sináptico positivo.

- **Tipo de conexiones entre neuronas.**

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

Con estos criterios surge la clasificación mostrada en la Figura 2.5

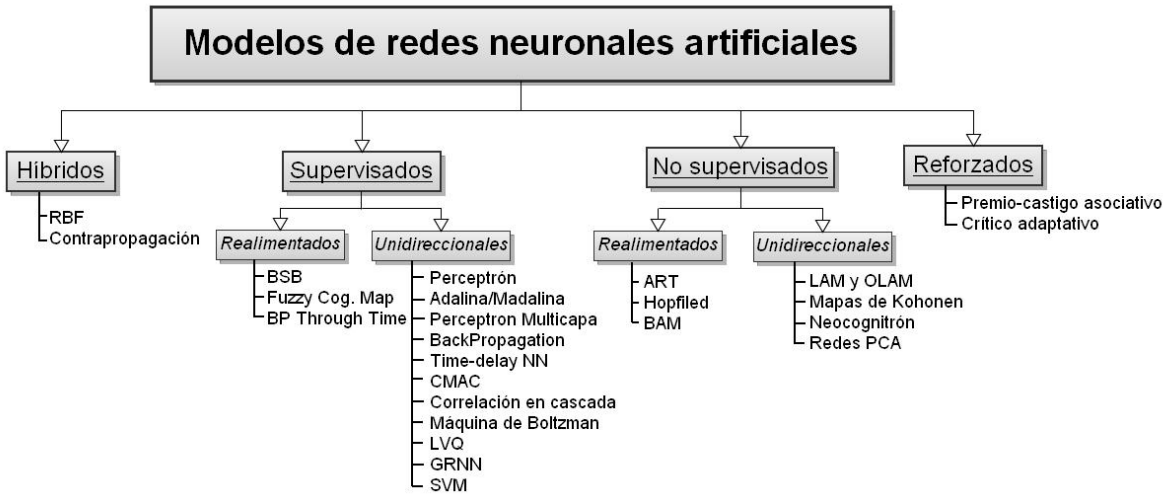


Figura 2.5: Clasificación de las redes neuronales por el tipo de aprendizaje y la arquitectura

2.3. Software para implementar las Redes Neuronales

Las redes neuronales pueden simularse mediante programas ejecutados en computadores convencionales. Consiste en modelar el sistema neuronal mediante un programa, que puede ejecutarse en un computador secuencial convencional tipo Von Neumann. Es el procedimiento más simple, rápido y económico. Por otra parte, es una solución muy versátil que permite el ensayo con cualquier tipo de arquitectura.

El proceso de simulación de las redes neuronales comienza con el modelado mediante programas de ordenador escritos en lenguaje de alto nivel, que se ejecutarán en computadoras convencionales compatibles. Aunque de esta manera se pierde su capacidad de cálculo en paralelo, las prestaciones que ofrecen los ordenadores actuales resultan suficientes para resolver numerosos problemas, al permitir simular redes de tamaño medio a una velocidad razonable.

Los programas de simulación de las redes neuronales se pueden clasificar en tres tipos:

- **Comerciales**, desarrollados por empresas.
- **Libre distribución**, realizados por grupos de investigadores de Universidades o particulares, que se pueden obtener de forma gratuita.
- **Producción propia**, que cada cual se puede confeccionar.

Las ventajas de los programas comerciales son los entornos gráficos que presentan, ya que son intuitivos y fáciles de usar, el elevado nivel de prestaciones que ofrecen, y en algunos casos, el gran número de modelos de redes que permiten simular. El principal inconveniente es su elevado precio y no poder manipular los modelos según convenga al usuario, ya que se tratan de productos cerrados.

En nuestra caso hemos optado por uno de estos programas, muy utilizado en nuestra universidad y que se puede utilizar en cualquier ordenador de la misma, MATLAB©. Este programa nos ha ayudado a simular y realizar toda clase de pruebas, ya que permite

explorar gráficamente y de forma rápida todas las arquitecturas de redes neuronales y encontrar la que mejor se adapta a nuestro problema.

2.3.1. MATLAB©

MATLAB (abreviatura de MATrix LABoratory, "laboratorio de matrices") [14] es un *software* matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M) y que se ha convertido casi en estándar en el campo de la ingeniería, particularmente en el de la enseñanza. Está disponible para las plataformas Unix, Windows y Apple Mac OS X.

Entre sus prestaciones básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispositivos *hardware*. El paquete MATLAB dispone de dos herramientas adicionales que expanden sus prestaciones, Simulink (plataforma de simulación multidominio) y GUIDE (editor de interfaces de usuario - GUI). Además, se pueden ampliar las capacidades de MATLAB con las cajas de herramientas (toolboxes); y las de Simulink con los paquetes de bloques (blocksets).

Las cajas de herramientas simplifican enormemente el trabajo en campos como el procesamiento de la señal e imagen, identificación y control de sistemas,etc. En nuestro caso, hemos utilizado **MATLAB Neural Networks Tolbox**, que está bastante extendido tanto en enseñanza como en investigación. Proporciona herramientas para el diseño, implementación, visualización y simulación de redes neuronales y contempla un gran número de sus modelos (BP, Elman, RBF, asociativas,etc) pudiendo adaptarlos a las necesidades de cada usuario. No obstante, puede perder algo de velocidad respecto a algunos simuladores, especialmente los desarrollados específicamente para las redes neuronales.

2.3.2. Python

Python [15] es un lenguaje de programación de alto nivel cuya filosofía hace hincapié en una sintaxis muy limpia y que favorezca un código legible.

Se trata de un lenguaje de programación multiparadigma ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico, es fuertemente tipado y multiplataforma.

Es administrado por *Python Software Foundation*. Posee una licencia de código abierto, denominada Python Software Foundation License, que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.

Además de la biblioteca estándar, usada para una diversidad de tareas, se han utilizado varias bibliotecas para implementar funcionalidades necesarias que proporciona MATLAB y que Python no tenía. Entre ellas están Image [16], numpy [17], pymorph [18] y Open CV [19].

2.4. Eyegrade

Eyegrade [20] es un sistema automático que corrige exámenes tipo test. Este tipo de herramientas aparecieron hace varias décadas, pero su uso todavía es muy limitado debido al coste de los equipos necesarios (escáneres y ordenadores), por lo que el coste de puesta en marcha es alto.

Eyegrade ofrece una solución rápida para el problema de la evaluación de los conocimientos a través de exámenes tipo test en papel, pero con un coste bajo ya que solo es necesario un PC con webcam.

Además el sistema permite configurar los exámenes, eligiendo el número de preguntas, pudiendo ser estas multirespuesta o de respuesta simple.

El sistema crea varios modelos mezclando las preguntas y genera las hojas de respuestas correspondientes a los modelos. Una vez el alumno ha rellenado el examen con su identificador y marcando las respuestas que considere correctas, se pasará por delante de la webcam. El sistema extraerá la información y mostrará por pantalla tanto las respuestas detectadas, como el número de respuestas correctas e incorrectas y el identificador del estudiante. En caso de que el sistema detecte incorrectamente una respuesta, la interfaz permite corregirlo con un click en la celda correcta. También es posible modificar el identificador del alumno en caso de que no se detectara correctamente. Cuando los datos son correctos, el sistema almacena la imagen con los datos (identificador del estudiante, el modelo de examen, las respuestas correctas e incorrectas y las respuestas de cada pregunta). Además genera un fichero con estos datos que puede ser exportado a otras herramientas para la gestión de las estadísticas de las preguntas o puntuaciones.

Los puntos más débiles del sistema que se detectaron en un primer experimento son la configuración del sistema, el tiempo necesario para detectar algunos exámenes y la fiabilidad en la detección de los identificadores de los alumnos.

Los resultados, procesando 230 exámenes, muestran que 97,0 % de las decisiones automáticas del sistema eran correctas. Con respecto a los identificadores un 85,2 % fueron correctos. Para conseguir este factor se implementó un algoritmo que mostraba el identificador más probable de una lista que se le ha pasado anteriormente. Sin este algoritmo, la precisión hubiera sido del 13,5 %. La tasa de acierto en la identificación de cada dígito aislado es del 72,86 %.

Este proyecto consiste en implementar una solución para mejorar esta tasa y hacer el sistema más robusto en este sentido, ya que se implementaron varias mejoras para decrementar el tiempo de detección de algunos exámenes siendo bastantes satisfactorios los resultados.

Capítulo 3

REQUISITOS

En este capítulo se expondrán los requisitos, tanto funcionales como no funcionales que debe tener el *software* para solucionar el problema expuesto anteriormente.

3.1. Funcionales

A continuación, se enumeran los requerimientos indispensables sin los cuales el módulo a desarrollar no cumple su funcionalidad.

- El sistema a implementar, debe reconocer los dígitos que distintas personas han escrito de manera manual, y que se le pasan con el formato de imágenes.
- En cada decisión, el sistema debe indicar cuál es el dígito más probable, así como proporcionar la lista de pesos estimados para todos los dígitos posibles.

3.2. No funcionales

A continuación, se especifican los requerimientos complementarios a los anteriores que son necesitados por el sistema.

- Utilizar técnicas de aprendizaje máquina, en concreto redes neuronales.
- Se realizarán varios prototipos en MATLAB para encontrar la mejor red neuronal.
- Debe integrarse en el sistema Eyegrade.
- Debe ser compatible en interfaz con el módulo de OCR actual de Eyegrade.
- Debe tener un tiempo de ejecución razonable para no ralentizar el sistema Eyegrade significativamente.
- Debe mejorar la tasa de acierto que tiene actualmente el sistema Eyegrade.

Capítulo 4

PROTOTIPO DE LA SOLUCIÓN

La finalidad de este capítulo es explicar la solución adoptada para resolver el problema presentado en los capítulos anteriores. Se irán explicando las diferentes etapas por las que se debe pasar, junto con las decisiones tomadas y su justificación. Además en el último apartado se explicarán los resultados que se obtuvieron y finalmente la decisión que se tomó para ser implementada en Python.

4.1. Introducción

Como se ha explicado en los capítulos anteriores, el problema que se debe resolver es reconocer correctamente los dígitos que rellenan los alumnos en los exámenes, es decir su identificador, para asignar automáticamente la nota de cada examen al alumno. Estos dígitos están encapsulados en celdas, teniendo que escribir cada dígito en una de ellas (ver Figura 4.1).

Para abordarlo el trabajo se centró en la tecnología de reconocimiento de caracteres, OCR (*Opctical Character Recognition*), que engloba un conjunto de técnicas basadas en estadísticas, en la forma de los caracteres, en transformaciones y comparaciones, que complementándose entre sí, distinguen de forma automática los diferentes caracteres dentro

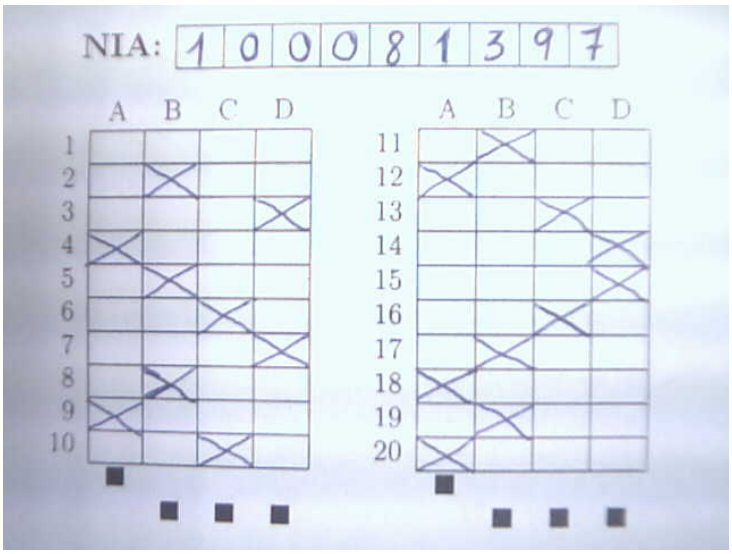


Figura 4.1: Ejemplo relleno de un examen

de un alfabeto.

En todo sistema de reconocimiento óptico de caracteres se distinguen al menos 4 etapas:

- Segmentación, selección de la zona de interés.
- Preprocesamiento, adecuación de la imagen.
- Extracción de características, representación digital de la imagen.
- Reconocimiento, distinción del carácter contenido en la imagen.

A lo largo del capítulo se van a describir las distintas etapas en las que se divide el sistema, basándose en las redes neuronales (ver Figura 4.2).

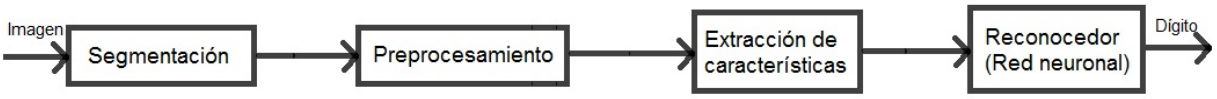


Figura 4.2: Diagrama del reconocedor óptico de caracteres (OCR)

4.2. Segmentación

La segmentación, dentro del procesamiento de imágenes, es el proceso de dividir una imagen en varias partes u objetos que se quieren localizar o analizar posteriormente.

En este caso, la segmentación permitirá la descomposición de la imagen capturada del examen en tantas partes como dígitos contenga. Estas partes serán también imágenes que se procesarán posteriormente.

En general, la segmentación de la imagen constituye una de las mayores dificultades dentro del procesado de imágenes, y por tanto del reconocimiento óptico de caracteres. En la mayor parte de los casos, una buena segmentación dará lugar a una solución correcta, por lo que es fundamental realizarla con el menor error posible [21].

En esta ocasión, esta operación es muy sencilla ya que el propio sistema Eyegrade devuelve datos sobre la imagen. Algunos de estos datos son las coordenadas (x, y) de las esquinas de cada celda. También devuelve las esquinas interiores de las celdas. Por tanto no se detecta, sino simplemente se debe recortar la imagen con estas coordenadas, obteniendo así la celda con el dígito correspondiente (ver Figura 4.3).

Por cada imagen, la segmentación se realizará tantas veces como dígitos contenga, obteniendo así distintas imágenes que se utilizarán y tratarán independientemente, obteniendo al final el dígito estimado para cada una de ellas.

4.3. Preprocesamiento

El objetivo que persigue esta fase es, mediante la aplicación de un conjunto de técnicas, mejorar la imagen original, eliminando cualquier tipo de ruido o imperfección que no pertenezca al carácter, y resaltando determinadas características para que resulte más fácil el reconocimiento. Además se normalizará el tamaño de la imagen para que no influya posteriormente en las siguientes etapas.

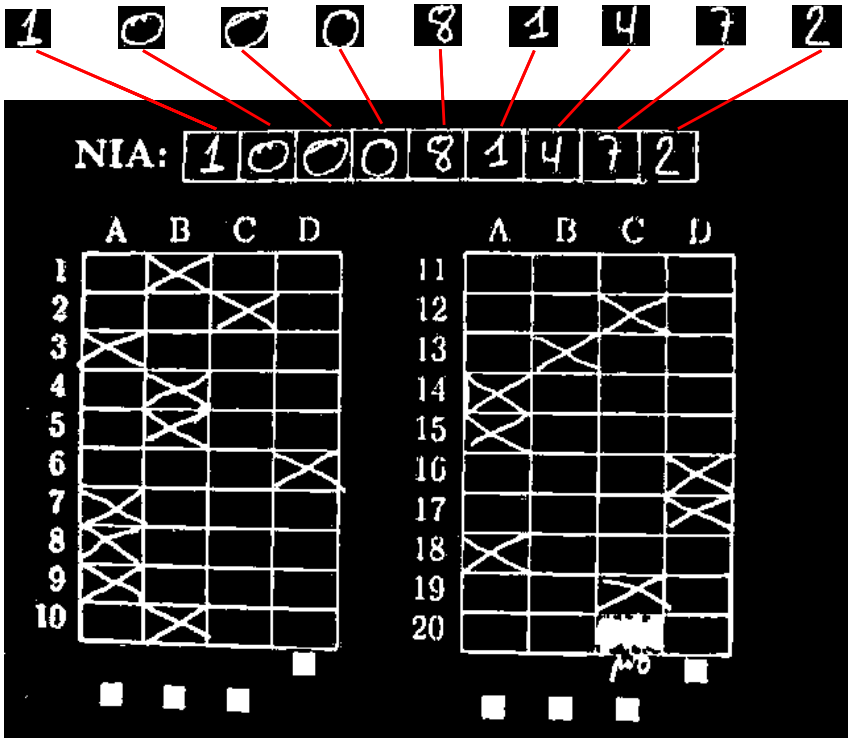


Figura 4.3: Ejemplo de segmentación de la identificación de un alumno

Las imágenes que se van a procesar son monocromáticas, cuyo fondo es negro (píxeles a 0) y cuyas líneas son blancas (píxeles a 255).

En primer lugar, se eliminará el ruido de la imagen aplicándole un filtro estadístico de mediana. Este filtro, no lineal, elimina el ruido impulsivo, también llamado "sal y pimienta", preservando los bordes de la imagen y sin reducir su nitidez [22] (ver Figura 4.4).

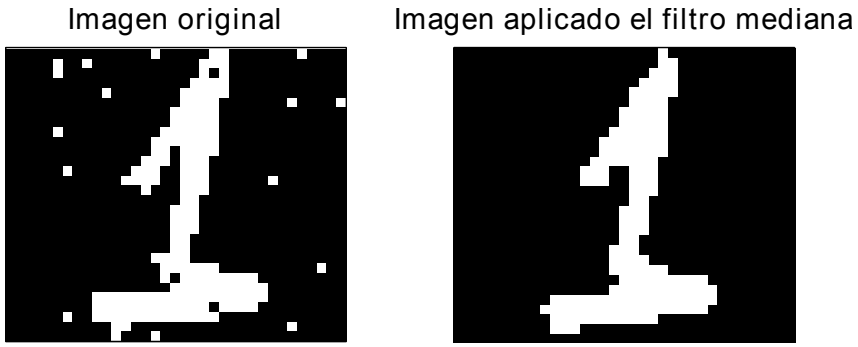


Figura 4.4: Aplicación de un Filtro de Mediana

Sobre la imagen resultante del filtro de mediana, se aplica la operación morfológica de cierre [23]. Con ella se consigue alisar el contorno, fusionar las grietas, rellenar los huecos y conectar objetos que están próximos entre sí. Se puede ver un ejemplo en la Figura 4.5.

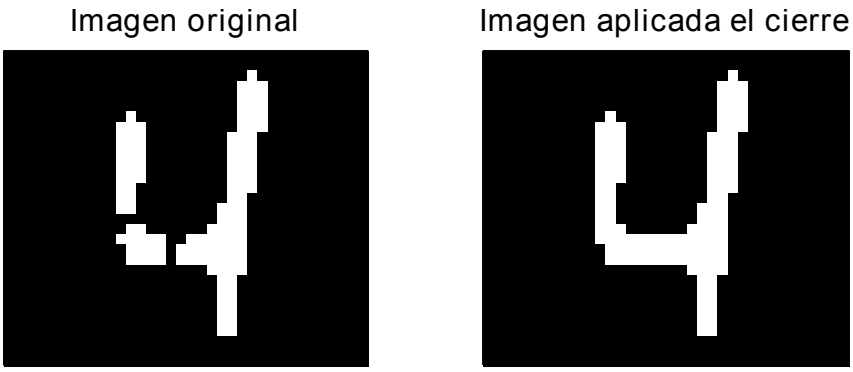


Figura 4.5: Aplicación de la operación morfológica de cierre

Después se aplica a la imagen resultante la operación morfológica de esqueletización [23], cuyo objetivo es reducir la imagen a un borde básico que se corresponde con su forma esencial, devolviendo únicamente su esqueleto en una imagen binarizada (ver Figura 4.6). Esto se realiza para que no afecte el grosor de la líneas del dígito cuando se haga la extracción de características.

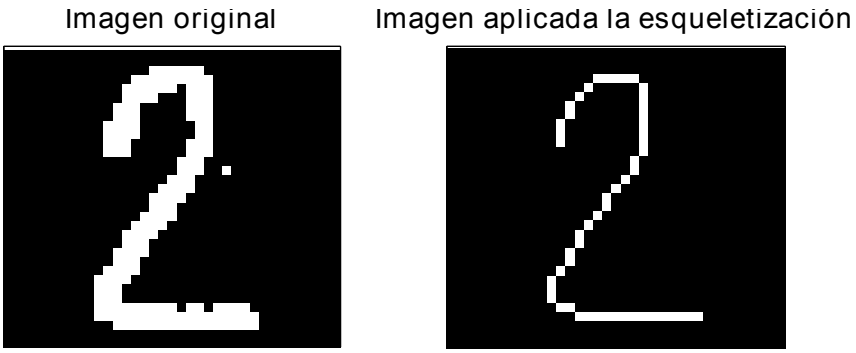


Figura 4.6: Aplicación de la operación morfológica de esqueletización

Como último paso, se recortará la imagen obteniendo otra que solo contenga el dígito que se reconocerá posteriormente. El tamaño de esta imagen se normalizará mediante el escalado de la imagen. Este método consiste en llevar a un tamaño estándar las dimensiones de la imagen, para que así el número de píxeles sea equivalente para todos los dígitos

(ver Figura 4.7). El tamaño al que se van a normalizar todas las imágenes es de 38 píxeles de alto y 45 píxeles de ancho.

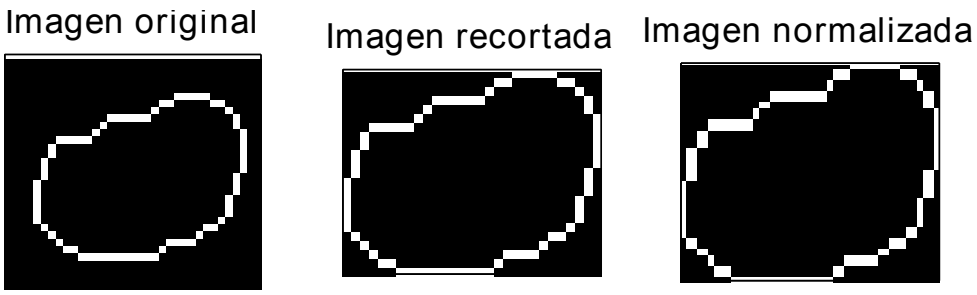


Figura 4.7: Normalización de la imagen

Con la aplicación de todas estas técnicas sobre las imágenes de los dígitos se consigue cierta uniformidad entre ellas, lo que ayuda a que en la siguiente etapa del OCR las características que se extraigan sean uniformes entre los mismos dígitos y diferentes entre dígitos distintos.

4.4. Extracción de características

Una vez realizada la segmentación y el preprocesamiento se tiene una imagen con la información susceptible de ser "reconocida".

El objetivo de esta etapa es poder distinguir entre los tipos de carácter de manera robusta. Aunque el mecanismo con que el ser humano realiza esta tarea no se conoce en su totalidad, parece que el análisis incluye un análisis estructural de cada carácter [24].

Por lo tanto, aparecen dos enfoques con respecto a la realización de este análisis estructural. En el primer enfoque, la imagen pixelada es introducida directamente al clasificador y durante el proceso de aprendizaje se analizan sus características, dando lugar a una estructura compleja. En el segundo enfoque, el análisis estructural se lleva a cabo en una etapa previa e independiente de la clasificación, en la cual se extraen las características consideradas importantes para poder distinguir entre los distintos caracteres, pudiendo ser la estructura de la red más general y menos compleja.

La opción escogida fue el segundo enfoque, ya que interesa alcanzar un diseño sencillo y adaptativo. Este paso tiene una gran dificultad ya que se deben elegir un conjunto de características que proporcionen unos resultados adecuados. Los principales requisitos que deben cumplir las características son:

- Discriminación, deben ser características que diferencien suficientemente una clase de otra, en nuestro caso un dígito de otro.
- Deben tener un valor similar para las mismas clases.
- Independencia, las características deben estar incorreladas unas de otras.
- Pequeño espacio para características, el número de características debe ser pequeño para mejorar la rapidez y facilidad de clasificación.

Además, la obtención de las características debe tener un bajo coste computacional, tanto en tiempo como en complejidad.

Las características que se han utilizado en este caso para el reconocimiento de los dígitos son:

1. La primera característica es la existencia de huecos en la imagen. Esta característica devuelve el número de huecos que contiene la imagen y la posición donde se encuentran. Con ella se consigue diferenciar entre los números que tienen huecos, como son el 0, 6, 8 y 9, y el resto que no tienen. Esta característica es muy útil si los trazos de los números están completos. En caso contrario, el resultado no sería correcto.
2. La segunda característica devuelve la proporción entre el ancho y el alto antes de aplicar la normalización a la imagen, último paso del preprocesamiento. Esta característica diferencia muy bien el número 1 del resto de dígitos, ya que su proporción en muy pequeña en comparación con el resto.

3. Para la siguiente característica se divide la imagen en 3 columnas de igual tamaño, y se devuelve el valor absoluto de la diferencia entre los píxeles que se encuentran en la primera columna de la imagen y los que se encuentran en la última. El valor devuelto será pequeño para números como el 0, 5 o el 8, y grandes para el 1, 3, 7. El resto de dígitos también tendrán un valor pero que no será tan significativo como en los anteriores.

4. Para la cuarta característica, de la misma manera que en la anterior, se divide la imagen en 3 filas de igual tamaño, y se devuelve el valor absoluto de la resta entre los píxeles que se encuentran en la primera fila de la imagen y los que se encuentran en la última. El resultado será pequeño para los números 0, 1, 3 y 8 y grande para el 4, 6, 7 y 9.

5. La quinta característica indica si la mayoría de los píxeles se encuentran alrededor de la imagen. Esta característica ayuda a diferenciar los números 0 y 8 del resto.

6. La sexta característica también divide la imagen en 3 columnas, e indica si el máximo de los píxeles se encuentran en la última columna de la imagen y es mayor que la media que se encuentra en la imagen. Esto va a permitir distinguir los dígitos 1, 4 y 7, que cumplirán estas condiciones.

7. Esta característica ayuda a diferenciar el número 4 del resto de números, ya que al realizar las primeras pruebas se observó una carencia de la red neuronal en este número. Esta característica compara si el máximo de píxeles se encuentra en la última columna. Después, compara si el siguiente máximo se encuentra en la primera columna y si el máximo de píxeles horizontales se encuentra en el centro de la imagen.

8. La octava característica devuelve la diferencia entre el número de píxeles de la mitad superior y de la mitad inferior de la imagen. Esto ayuda a diferenciar entre aquellos números que tienen muchos píxeles en la parte inferior, como el 6, ya que el resultado será grande y positivo. En cambio, aquellos números que tengan la mayoría de sus píxeles en la parte superior, como el 9 o el 7, el valor devuelto será grande pero negativo.

9. Esta característica se añadió para diferenciar el número 7 por el mismo motivo que se integró la séptima característica. Para ello, se utiliza una función que te devuelve la fila y la columna donde se encuentra el mayor número de pixeles de la imagen que le pasas como parámetro. Con ello, se compara que el máximo de píxeles horizontales se encuentre el la parte superior de la imagen, y que que el máximo de píxeles verticales no se encuentre en la zona de la izquierda de la misma. Además tiene en cuenta si la imagen lleva dibujado un trazo horizontal en el centro.

10. La décima característica devuelve la simetría vertical del dígito. El resultado será positivo para el caso del 0 y del 8.

11. La undécima característica que se extrae devuelve la simetría horizontal del número que contiene la imagen. El valor obtenido será positivo para los dígitos 0, 3, y 8.

12. La última característica devuelve la proporción de píxeles que se encuentran en el centro de la imagen con respecto al resto. Esto dará un valor alto y negativo para aquellos números que no tengan muchos píxeles en el centro, como para el caso del 0, 1, 2 o 7.

Una vez obtenidas las características se pasa a la siguiente etapa donde se va a devolver el dígito estimado.

4.5. Reconocimiento

Una vez se tienen las características más importantes de la imagen hay que determinar el dígito correspondiente por medio de una de las técnicas de minería de datos, en nuestro caso las redes neuronales.

Como ya dijimos en el capítulo 2.2, las redes neuronales intentan imitar la arquitectura del cerebro. Se componen de una serie de unidades básicas, llamadas neuronas, que reciben una entrada, y mediante unos pesos realizan diferentes operaciones para presentar una salida.

Existen muchos modelos de redes neuronales, pero vamos a utilizar uno de los más empleados el **Perceptrón Multicapa** (MLP) con aprendizaje *backpropagation*, ya que suele utilizarse en tareas de ajuste funcional y clasificación [1], como es este caso.

Este modelo se obtiene de añadir capas ocultas a un perceptrón simple (más información en [25]). Además, esta arquitectura suele entrenarse con un algoritmo denominado retropropagación o *backpropagation*(BP).

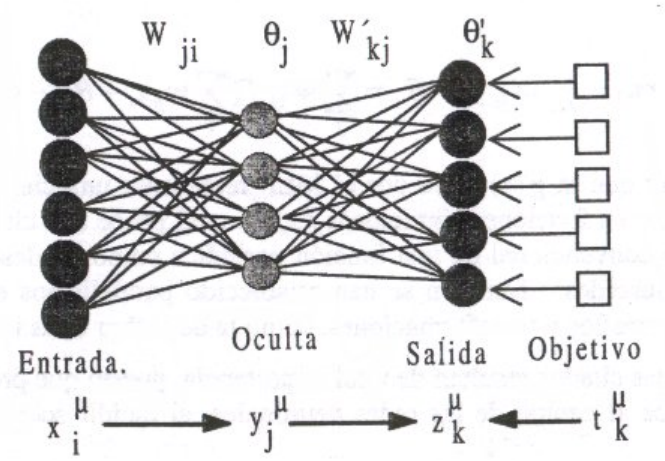


Figura 4.8: Arquitectura del MLP

Se puede expresar matemáticamente la operación de un MLP con una capa oculta y neuronas de salida lineal, denominando x_i a las entradas de la red, y_j a las salidas de la capa oculta y z_k a las de la capa final, siendo t_k las salidas objetivo (ver Figura 4.8), como:

$$z_k = \sum_j w'_{kj} y_j - \theta'_i = \sum_j w'_{kj} f(\sum_i w_{ij} x_i - \theta_j) - \theta'_i \tag{4.1}$$

Donde w_{ij} son los pesos de la capa oculta y θ_j sus umbrales, w'_{kj} los pesos de la capa de salida y θ'_k sus umbrales y $f(.)$ de tipo sigmoideo como por ejemplo:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4.2}$$

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \tanh(x) \tag{4.3}$$

La Ecuación 4.2 proporciona una salida en el intervalo [0, +1], sin embargo la Ecuación 4.3 tiene un intervalo entre [-1, +1].

En nuestro caso se utilizará la última Ecuación 4.3, ya que una de sus propiedades es su alta velocidad de procesamiento [1].

Esta es la arquitectura más común de MLP, aunque existen numerosas variantes [26].

El procedimiento para entrenar la arquitectura MLP mediante BP es el siguiente:

1. Establecer aleatoriamente los pesos y umbrales iniciales. Se debe comenzar siempre con pesos iniciales aleatorios (normalmente números pequeños, positivos y negativos), ya que si se parte de pesos y umbrales iniciales nulos el aprendizaje no progresará.

2. Para cada patrón μ del conjunto de aprendizaje:

   a) Llevar a cabo una fase de ejecución para obtener la respuesta de la red ante el patrón $\mu - simo$.

   $$E(w_{ji}, \theta_j, w'_{kj}, \theta'_k) = \frac{1}{2} \sum_{\mu} \sum_{k} [t_k^{\mu} - g(\sum_{j} w'_{kj} y_j^{\mu} - \theta'_k)]^2 \tag{4.4}$$

   cuya minimización se lleva a cabo mediante descenso por el gradiente; en esta ocasión habrá un gradiente respecto de los pesos de la capa de salida (w'_{kj}) y otro respecto de los que oculta (w_{ji}).

   $$\delta w'_{kj} = -\varepsilon \frac{\partial E}{\partial w'_{kj}} \tag{4.5}$$

   $$\delta w_{ji} = -\varepsilon \frac{\partial E}{\partial w'_{ji}} \tag{4.6}$$

   b) Calcular las señales de error asociadas $\triangle'^{\mu}_k$ y $\triangle^{\mu}_j$.

   $$\delta w'_{kj} = \varepsilon \sum_{\mu} \triangle'^{\mu}_k y_j^{\mu}, \text{ con } \triangle'^{\mu}_k = [t_k^{\mu} - g(h'^{\mu}_k)] \frac{\partial g(h'^{\mu}_k)}{\partial h'^{\mu}_k} \text{ y } h'^{\mu}_k = \sum w'_{kj} y_j^{\mu} - \theta'_k \tag{4.7}$$

$$\delta w_{ji} = \varepsilon \sum_{\mu} \triangle_j^{\mu} x_i^{\mu}, \text{ con } \triangle_j^{\mu} = (\sum_k \triangle_k'^{\mu} w'_{kj}) \frac{\partial f(h_j^{\mu})}{\partial h_j^{\mu}} \text{ y } h_j^{\mu} = \sum_i w_{ji} x_i^{\mu} - \theta_j \tag{4.8}$$

c) Calcular el incremento parcial de los pesos y umbrales debidos a cada patrón μ (Ecuaciones 4.7 y 4.8).

3. Calcular el incremento total (para todos los patrones) actual de los pesos $\delta w'_{kj}$ y δw_{ji} (Ecuaciones 4.7 y 4.8).

4. Actualizar pesos y umbrales

5. Calcular el error actual (Ecuaciones 4.5 y 4.6) y volver al punto 2 si todavía no es satisfactorio.

En el esquema presentado, que surge de manera natural del proceso de descenso por gradiente, se lleva a cabo una fase de ejecución para todos y cada uno de los patrones del conjunto de entrenamiento, se calcula la variación en los pesos debida a cada patrón, se acumulan y solamente entonces se procede a la actualización de los pesos. Este esquema se suele denominar **aprendizaje por lotes** (*batch*). Otra posibilidad consiste en actualizar los pesos sinápticos tras la presentación de cada patrón μ (en vez de presentarlos todos y luego analizar), esquema denominado **aprendizaje en serie** (on line). Recientemente se ha demostrado [27] que el aprendizaje en serie estima mejor el gradiente, permite emplear ritmos de entrenamiento mayores y suele ser más rápido.

El algoritmo BP constituye un método de gran generalidad, lo que presenta ventajas e inconvenientes. Su principal ventaja es que se puede aplicar a multitud de problemas diferentes, proporcionando con frecuencia buenas soluciones con no demasiado tiempo de desarrollo. Un inconveniente es su lentitud de convergencia, precio a pagar por disponer de un método general de ajuste funcional.

Para resolver algunos inconvenientes del BP se plantean correcciones o variantes. Algunas de ellas tratan de resolver el problema de la lenta convergencia, mientras que otros se centran en conseguir una mejor generalización. Una de las variantes más importantes

para el problema de la convergencia es incluir en el algoritmo un **término de inercia**. Consiste en añadir al cálculo de la variación de los pesos (Ecuaciones 4.7 y 4.8) un término adicional proporcional al incremento de la iteración anterior, proporcionando una cierta inercia al entrenamiento.

$$\delta w'_{kj}(t+1) = -\varepsilon \frac{\partial E}{\partial w'_{kj}}|_t + \alpha \delta w'_{kj}(t-1) \tag{4.9}$$

$$\delta w'_{ji}(t+1) = -\varepsilon \frac{\partial E}{\partial w'_{ji}}|_t + \alpha \delta w'_{ji}(t-1) \tag{4.10}$$

con α un parámetro de 0 a 1, que se suele tomar próximo a 1 ($\alpha \approx 0{,}9$).

De esta manera, si los incrementos en un determinado peso tienen siempre el mismo signo, las actualizaciones en cada iteración serán mayores; sin embargo, si los incrementos oscilan (a veces positivos, a veces negativos) el incremento efectivo tiende a cancelarse. Así, en zonas estrechas y profundas de la superficie del error (con forma de valle angosto), los pesos correspondientes a las dimensiones estrechas, que sin el término de inercia oscilarían de un lado al otro del valle, sufren pequeños incrementos, mientras que los de las direcciones que descienden directamente al fondo se ven potenciados [28]. Con esta variante se aumenta el ritmo de aprendizaje efectivo en determinadas direcciones.

Estas técnicas aceleradoras dan lugar a variantes del BP. También existen un grupo de algoritmos denominados **métodos de segundo orden**, que utilizan la segunda derivada del error. Entre ellos se encuentran los algoritmos de gradientes conjugados, Newton, Levenberg-Marquardt,etc. Se puede encontrar más información en [28] [29].

En este caso, y ya que MATLAB te permite elegir entre muchos algoritmos, se va a escoger el de Levenberg-Marquardt para el entrenamiento de la red neuronal. Este algoritmo es una optimización de los algoritmos de Newton, que se diseñó para tener que calcular la matriz Jacobiana en vez de la Hessiana y así mejorar el rendimiento y la velocidad de cálculo de los pesos. Este algoritmo utiliza un aprendizaje serie. Se ha escogido este algoritmo ya que es el método más rápido para la formación de redes

neuronales de tamaño moderado, hasta varios cientos de pesos.

Otra elección a tener en cuenta es la magnitud de los pesos iniciales, pues una correcta elección puede suponer un menor tiempo de entrenamiento. Para el uso de la función de activación tangente hiperbólica, simplemente el elegir los pesos aleatoriamente en el intervalo [-2.4/n, +2.4/n] siendo n el número de entradas en la neurona, suele dar buenos resultados [25]. Esto lo calcula directamente MATLAB sin necesidad de introducir ningún peso.

4.6. Experimentos realizados y resultados

En las secciones anteriores se ha visto qué tecnología se va a utilizar y cómo se va a implementar. A modo de resumen, se van a recapitular las principales ideas, antes de describir los experimentos que se realizaron junto con sus resultados y la justificación de la solución escogida.

El prototipado de la solución se implementó en MATLAB, ya que dispone de unas prestaciones muy potentes en cálculo, además de ofrecer una caja de herramientas exclusiva para redes neuronales, por lo que las fases de experimentación y prototipado se simplifican en gran medida.

Se implementaron una serie de funciones para aplicar las diferentes técnicas por las que debían pasar todas las imágenes: segmentación (sección 4.2) y preprocesamiento (sección 4.3), según se explicó anteriormente. Una vez hecho esto, se implementó otra función que extraía las características de las imágenes, explicadas en la sección 4.4, cuya salida es una matriz con estos valores. Después, se dividió dicha matriz en los conjuntos de entrenamiento, test y validación. En este momento se pasó a la etapa de reconocimiento (sección 4.5), donde se empezaron a realizar pruebas con el algoritmo del Perceptrón Multicapa con aprendizaje BP, modificando sus parámetros (función de activación, número de neuronas, número de características), obteniendo diferentes resultados que serán expuestos y analizados a continuación.

4.6.1. Descripción de los datos

Para realizar los experimentos se han empleado un conjunto de datos reales, proporcionados por el tutor de proyecto, obtenidos de exámenes realizados a alumnos de la Universidad Carlos III de Madrid.

Fueron un total de 342 exámenes, de los cuales 316 eran de distintos alumnos. Cada uno contiene los 9 dígitos del identificador del alumno que ha realizado el examen. Por tanto, el número de dígitos para realizar los experimentos fueron de 3078.

Estos dígitos se dividieron en tres conjuntos: conjunto de entrenamiento, conjunto de test y conjunto de validación repartiéndose en 60 %, 20 % y 20 % respectivamente.

4.6.2. Experimento 1

Para el primer experimento se escogió una red neuronal con el algoritmo de aprendizaje BP con la variante Levenberg-Marquardt. Como parámetros se eligió la función de transferencia de la Ecuación 4.3. El número de neuronas fueron 20 en la capa oculta y 10 en la capa de salida, quedando la red que se muestra en la Figura 4.9.

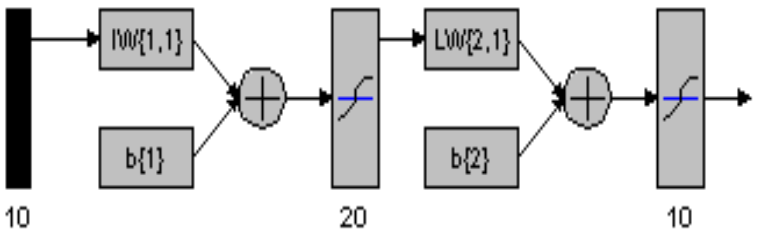


Figura 4.9: Arquitectura de la red neuronal usada en el Experimento 1

Se procesaron las imágenes y se extrajeron sus características con las funciones explicadas en secciones anteriores. En un principio las características fueron 10, sin incluir la

número 7 y la número 9, ya que no se sabía cuales de los dígitos se iban a reconocer peor.

Como se puede observar en la Figura 4.10, donde se muestra una evolución del error de aprendizaje de los datos de entrenamiento y el error de generalización del conjunto de test, ambos se empiezan a decrementar monótonamente hasta cierto punto que el error de generalización marcado por las muestras de test empieza a aumentar, es decir, empieza a haber un sobreajuste de la red neuronal.

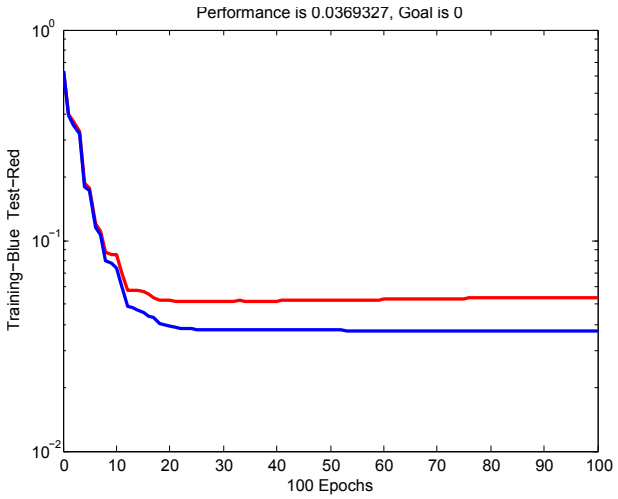


Figura 4.10: Representación del error en el entrenamiento para la red del Experimento 1

Con el entrenamiento realizado, la tasa de acierto para el conjunto de entrenamiento es del 51,87 %, para el conjunto de test es del 31,20 %. La tasa de acierto para cada número se puede observar en la Tabla 4.1. En ella se observa que los números que mejor son reconocidos son el 0, el 1 y el 8, y los que más problemas tiene para reconocer son el 3, 4 y 7.

Para eliminar el sobreajuste aparecido en el entrenamiento de la red neuronal se volvió a entrenar, pero esta vez usando el procedimiento de validación cruzada para encontrar el punto óptimo en el que se debe dejar de entrenar la red, es decir donde se encuentre el error de generalización mínimo para esta red. En la Figura 4.11 se puede observar cómo el entrenamiento ha sido detenido en la época 18, ya que es donde se consigue el mínimo error de generalización para el conjunto de validación.

Números	Conjunto de entrenamiento (%)	Conjunto de test (%)
0	94,81	84,62
1	85,71	77,14
2	48,75	29,02
3	40,86	25,29
4	42,88	29,29
5	76,89	54,29
6	67,86	54,29
7	53,16	40,00
8	84,00	68,15
9	63,87	41,61
Total	51,87	31,20

Tabla 4.1: *Tasa de acierto de la red neuronal del Experimento 1*

Una vez entrenada la red neuronal con el procedimiento de validación cruzada se obtuvo una tasa de acierto para el conjunto de entrenamiento del 49,80 %, para el conjunto de test un 35,80 % y para el conjunto de validación del 36,50 %.

Comparándolo con los datos anteriores, el conjunto de entrenamiento tiene una tasa un poco menor, pero en cambio los conjuntos de test y validación mejoran su tasa de acierto. En la Tabla 4.2 se puede ver también la evolución de los dígitos y su tasa de acierto.

Números	Conjunto de entrenamiento (%)	Conjunto de test (%)	Conjunto de validación (%)
0	86,90	88,00	74,19
1	80,25	72,00	76,67
2	41,02	32,50	32,00
3	33,64	25,00	30,77
4	31,02	37,14	41,86
5	54,03	41,90	43,64
6	59,00	48,30	43,33
7	50,85	38,89	32,62
8	57,05	35,83	40,00
9	50,36	36,67	43,33
Total	49,80	35,80	36,50

Tabla 4.2: *Tasa de acierto de la red neuronal utilizando el procedimiento de validación cruzada del Experimento 1*

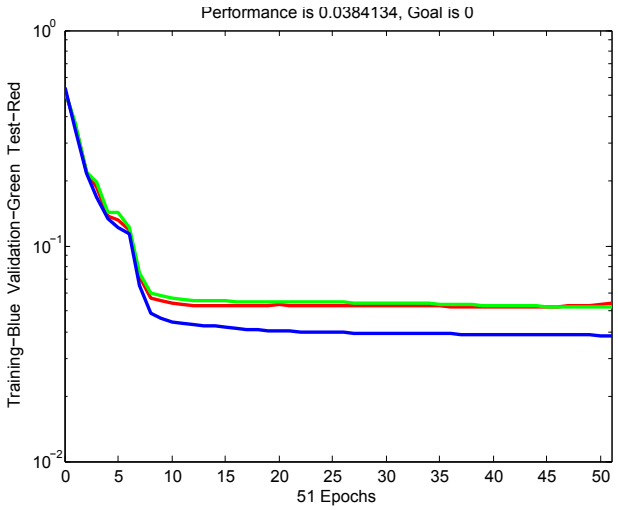


Figura 4.11: Representación del error en el entrenamiento utilizando el procedimiento de validación cruzada para la red del Experimento 1

Como esta tasa de acierto estaba muy por debajo de la que se obtenía con el sistema Eyegrade (72,86 %) se decidió realizar más experimentos aumentando el número de capas de la red neuronal y el número de neuronas que tienen cada una de ellas.

4.6.3. Experimento 2

En el intento de mejorar los resultados del experimento anterior se decidió aumentar el número de neuronas de la capa oculta de 20 a 40 neuronas. El resto de parámetros se siguieron manteniendo.

El entrenamiento de la nueva red neuronal se realizó con las mismas características que se usaron en el anterior. También se decidió aplicar el procedimiento de validación cruzada (ver Figura 4.12). En este caso también se puede ver cómo el error de aprendizaje y generalización van disminuyendo y a partir de un cierto punto el error de generalización de los conjuntos de test y validación no mejora y se empieza a producir el sobreajuste, momento en que se detiene el entrenamiento.

El resultado de la tasa de acierto que se obtuvo en este caso para el conjunto de

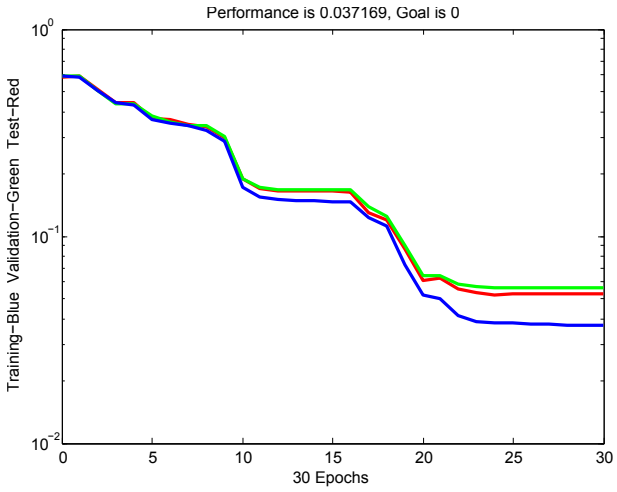


Figura 4.12: Representación del error en el entrenamiento utilizando el procedimiento de validación cruzada para la red del Experimento 2

entrenamiento fue 65,73 %, para el conjunto de test de 39,60 % y para el conjunto de validación de 38,40 %.

Estos resultados son mejores en el conjunto de entrenamiento, pero en los conjuntos de validación y test apenas se nota la mejoría, por lo que se decidió realizar una nueva prueba introduciendo una nueva capa oculta.

4.6.4. Experimento 3

Como se ha dicho anteriormente, en este caso se va a introducir una nueva capa oculta a la red neuronal. Por tanto, la red neuronal para este experimento constará de los 10 valores de características extraídos de las imágenes, dos capas ocultas con 40 y 20 neuronas respectivamente y la salida que contará con 10 valores, uno por cada dígito (ver Figura 4.13).

En este caso, la tasa de acierto para el conjunto de entrenamiento es 70,40 %, para el conjunto de test es 49,80 % y para el conjunto de validación es 47,80 %. Estos valores son mejores que los experimentos realizados anteriormente, además se ve en la Figura 4.14 como el error de generalización llega a un mínimo mayor que en los casos anteriores.

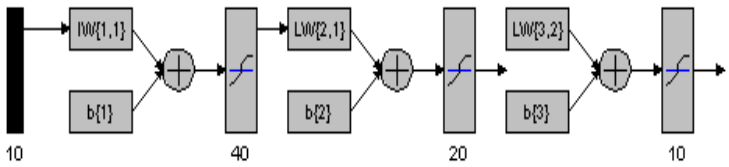


Figura 4.13: Arquitectura de la red neuronal empleada en el Experimento 3

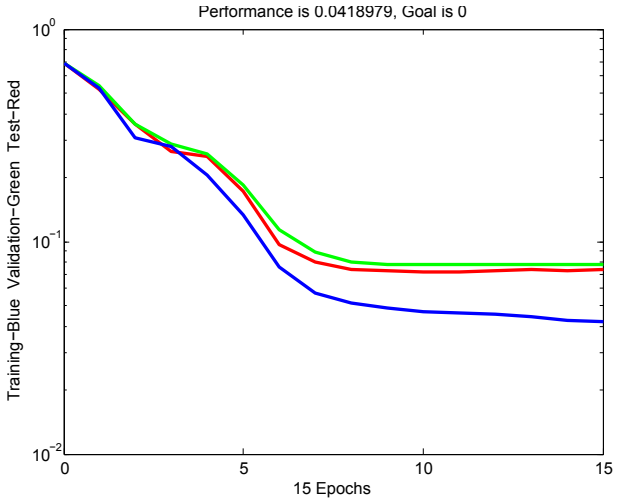


Figura 4.14: Representación del error en el entrenamiento utilizando el procedimiento de validación cruzada para la red del Experimento 3

También, se puede observar la tasa de acierto para cada número mostrada en la Tabla 4.3.

Números	Conjunto de entrenamiento (%)	Conjunto de test (%)	Conjunto de validación (%)
0	93,18	70,21	68,71
1	83,57	69,68	66,35
2	60,47	55,56	47,41
3	47,06	23,53	43,15
4	55,00	42,86	45,95
5	84,78	64,00	76,21
6	70,56	68,57	58,24
7	64,55	35,48	37,54
8	78,13	50,00	35,48
9	78,00	75,71	71,67
Total	70,40	49,80	47,80

Tabla 4.3: *Tasa de acierto de la red neuronal utilizando el procedimiento de validación cruzada del Experimento 3*

Una cosa que llama la atención en dicha tabla es la diferencia de la tasa de acierto existente entre los dígitos 0 y 1 y el resto de números. Esto se debe a que hay un mayor número de muestras de estos dígitos que del resto, debido a que los identificadores de los alumnos empiezan todos por "100". Se puede ver la proporción de muestras en la Tabla 4.4.

Dígito	0	1	2	3	4	5	6	7	8	9	Total
Número de muestras	983	482	282	216	142	142	148	302	256	125	3078
% muestras	31,93	15,65	9,16	7,01	4,61	4,61	4,80	10,00	8,31	4,06	100,00

Tabla 4.4: *Proporción de dígitos en las muestras de entrenamiento*

Esto es un problema, ya que la tasa de error varía en caso de que todos los dígitos tuvieran la misma proporción en las muestras que se le está dando a la red neuronal para que entrene. Por tanto, se buscó el dígito que tuviera menos muestras, es decir el 9 con 125 muestras, y se hizo una restricción en los demás dígitos seleccionando solo 125 muestras de cada uno de ellos. Además la división de los conjuntos de entrenamiento, test y validación también tuvo que ser ecuánime, por lo que a partir de este momento

el conjunto de entrenamiento cuenta con 75 muestras de cada número y los conjuntos de test y validación con 25 muestras de cada número cada uno de ellos.

Se volvió a entrenar la red con los mismos parámetros que la anterior, pero con 1250 muestras en vez de con 3078, tal y como se ha explicado.

4.6.5. Experimento 4

En este experimento, se utiliza la misma red neuronal que en el experimento anterior, pero el número de muestras con las que se entrena son 1250 para solucionar el problema de que los dígitos 0 y 1 fueran reconocidos con una alta tasa de acierto muy superior al resto de lo números.

Una vez entrenada la red neuronal con la restricción de las muestras usadas se obtuvo una tasa de acierto para el conjunto de entrenamiento del 73,73 %, para el conjunto de test del 51,20 % y para el conjunto de validación del 53,60 %. Se puede observar la evolución del entrenamiento en la Figura 4.15.

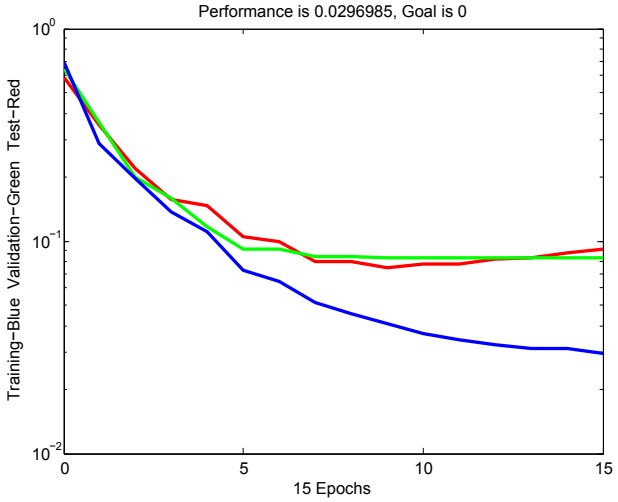


Figura 4.15: Representación del error en el entrenamiento utilizando el procedimiento de validación cruzada para la red del Experimento 4

Con la restricción de las muestras se observa como se aumenta la tasa de acierto de algunos de los dígitos igualándose al orden de los números 0 y 1 o incluso superándolos.

Números	Conjunto de entrenamiento (%)	Conjunto de test (%)	Conjunto de validación (%)
0	90,12	84,62	68,57
1	81,36	50,00	70,83
2	70,93	35,71	52,38
3	50,00	31,82	27,27
4	60,49	32,35	33,33
5	87,88	54,55	65,22
6	75,95	73,08	56,67
7	61,05	42,11	33,33
8	80,30	45,45	56,25
9	91,53	66,67	87,50
Total	73,73	51,20	53,60

Tabla 4.5: *Tasa de acierto de la red neuronal utilizando el procedimiento de validación cruzada del Experimento 4*

Por tanto, se ha conseguido el objetivo. Los siguientes experimentos se realizarán con la restricción de las 1250 muestras, 125 por cada número.

Aún así se observa que hay algunos dígitos que sí mejoraran su tasa podrían aumentar la tasa de acierto general, como por ejemplo el 4 y el 7 que tienen una de las tasas de acierto más baja. Por eso se deciden incorporar dos nuevas características con la que la red neuronal tendrá más información y podrá clasificar mejor estos números.

4.6.6. Experimento 5

En este experimento, se sigue utilizando la misma red neuronal que en los dos anteriores, es decir, con dos capas ocultas con 40 y 20 neuronas y una salida con 10 valores, pero con 12 valores de entrada. Se añaden las características 7 y 9 explicadas en la sección 4.4 para mejorar el acierto de los dígitos 4 y 7 (ver Figura 4.16).

Al entrenar la red neuronal con las dos nuevas características añadidas se obtuvo una tasa de acierto para el conjunto de entrenamiento del 83,43 %, para el conjunto de test del 65,80 % y para el conjunto de validación del 68,80 % (ver Figura 4.17).

Como se observa en la Tabla 4.6, la tasa de acierto de los números 4 y 7 han aumentado

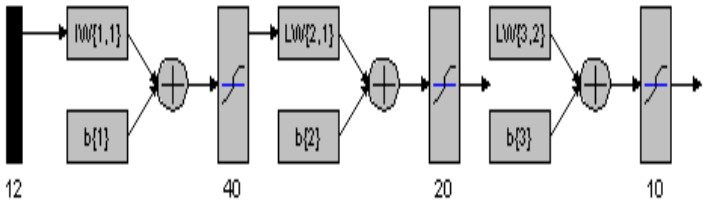


Figura 4.16: Arquitectura de la red neuronal empleada en el Experimento 5

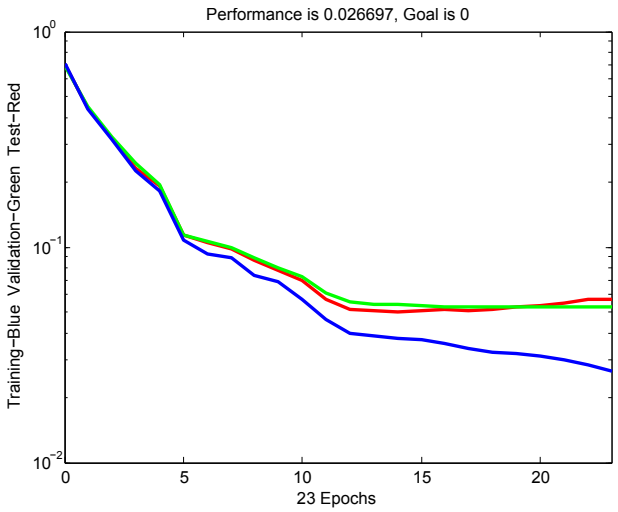


Figura 4.17: Representación del error en el entrenamiento utilizando el procedimiento de validación cruzada para la red del Experimento 5

Números	Conjunto de entrenamiento (%)	Conjunto de test (%)	Conjunto de validación (%)
0	100,0	85,10	88,75
1	95,94	76,25	69,04
2	90,47	71,29	69,71
3	65,38	54,00	56,00
4	73,75	59,80	72,00
5	91,17	66,11	57,14
6	75,00	60,53	69,31
7	88,57	51,00	55,71
8	89,85	56,95	61,74
9	87,14	67,00	72,98
Total	83,46	65,80	68,80

Tabla 4.6: *Tasa de acierto de la red neuronal añadiendo dos características a las ya existentes*

considerablemente. Por tanto, se ha logrado la finalidad al añadir estas 2 características. Además, se ha mejorado la tasa de acierto, llegando al 83,46 % para el conjunto de entrenamiento.

Como el objetivo es intentar alcanzar una mejor tasa de acierto, en el siguiente experimento se modificará la red neuronal añadiendo más neuronas en las capas ocultas de las red.

4.6.7. Experimento 6

En este experimento, se añaden más neuronas a la red neuronal intentando que la tasa de acierto sea más alta que en las pruebas realizadas anteriormente para los tres conjuntos con los que se trabaja. En esta ocasión, se entrenará la red neuronal con las 12 entradas utilizadas en el anterior experimento, pero se incrementarán las neuronas de las capas ocultas a 60 y 30 respectivamente, dejando la salida con los 10 valores correspondientes a los 10 dígitos, quedando la red como la que se muestra en la Figura 4.18.

Esta red obtuvo una tasa de acierto del 86,56 % para el conjunto de entrenamiento, un 51,60 % para el conjunto de test y un 50,80 % para el conjunto de validación. Como

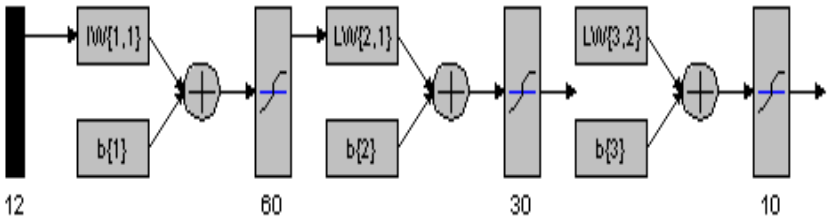


Figura 4.18: Arquitectura de la red neuronal empleada en el Experimento 6

se puede observar la tasa de acierto disminuye en todos los conjuntos, aunque el cambio es más significativo en los conjuntos de test y validación. Además, viendo la Figura 4.19 donde se muestra la evolución del entrenamiento de la red neuronal, se puede observar cómo, aunque se ha realizado la validación cruzada para el entrenamiento, se produce una mayor separación entre el error de aprendizaje (conjunto de entrenamiento) y el de generalización (test-validación). Por tanto, la red está teniendo un sobreajuste mayor que en otras ocasiones, en que aunque también sucedía esto, la tasa de acierto de los conjuntos de test y validación es superior. Por tanto, esta red neuronal no mejora la anterior.

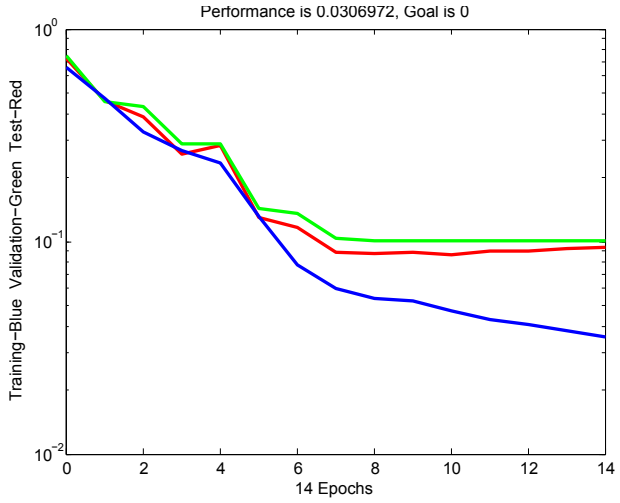


Figura 4.19: Representación del error en el entrenamiento utilizando el procedimiento de validación cruzada para la red del Experimento 6

4.6.8. Conclusión

Según los experimentos realizados y los resultados obtenidos, se puede decir que la mejor red neuronal es la que contiene 2 capas ocultas con 40 y 20 neuronas respectivamente. Además, como entradas a dicha red se va a utilizar 12 características por cada imagen de cada dígito, ya que se ha visto que los resultados del entrenamiento eran superiores que si se entrenaba la misma red con 10 características. La salida solo cabe la posibilidad de que tenga 10 valores, ya que es el número de dígitos que se pueden reconocer.

Además, es necesario tener la misma proporción de muestras para realizar el entrenamiento, ya que sino los números con mayores muestras tienen una tasa mayor de reconocimiento que el resto y esto ocasiona que el resto de números tenga una peor tasa de acierto y la tasa total del conjunto también se degrade.

Por tanto, la red escogida como mejores resultado es la de la Figura 4.16, realizada en el Experimento 5, con una tasa de acierto del 83,43 % para el conjunto de entrenamiento, 65,80 % para el conjunto de test y 68,80 % para el conjunto de validación. Este porcentaje no es mayor que el obtenido en el sistema Eyegrade (72,86 %), por tanto el objetivo de mejorar la tasa no se ha conseguido en el prototipo de la solución.

Capítulo 5

SOLUCIÓN

En este capítulo se explicará cómo se ha trasladado la solución implementada en el capítulo anterior al lenguaje de programación Python para que se pudiera integrar con el sistema Eyegrade.

5.1. Introducción

Una vez se implementó el prototipo de la solución y se concretaron los parámetros de la red neuronal que mejores resultados daban, junto con las entradas que se iban a introducir en dicha red neuronal, se trató de llevar al lenguaje de programación Python para así poderlo integrar en el sistema Eyegrade sustituyendo el módulo de reconocimiento de dígitos que tiene en la actualidad.

Para trasladar el prototipo a la solución final en el sistema Eyegrade, lo que se hizo fue reemplazar el módulo OCR que tenía el sistema por el nuevo basado en redes neuronales. Los pasos que se siguieron fueron:

- Implementar el procesamiento de imágenes para tratar los dígitos por separado.
- Desarrollar los algoritmos necesarios para extraer las características de las imágenes

que luego serán utilizadas como entradas de la red neuronal.

- Implementar la red neuronal con los pesos obtenidos en el prototipo realizado con anterioridad.

A lo largo del capítulo se van a describir los pasos seguidos con una sección final en la que se expondrán los resultados en el sistema Eyegrade y se compararán con lo que había anteriormente.

5.2. Procesamiento de las imágenes

La segmentación de la imagen se hace de la misma manera que se hacía en el prototipo de la solución. El sistema Eyegrade devuelve, tanto las esquinas exteriores de la celda de cada número como las interiores. Cogiendo estos datos recortamos la imagen, pasándola antes a una matriz que gracias a la biblioteca *NumPy* [17] permite recortarla escogiendo la parte que interesa y así tener tantas imágenes como dígitos contenga la imagen inicial para procesarlos de manera individual.

El procesamiento de la imagen para eliminar todas las imperfecciones y extraer la información que importa también se hizo con la importación de bibliotecas de Python.

La aplicación del filtro de mediana, se hizo gracias al módulo *ImageFilter* [30], que tiene un método que hace la mediana de cualquier imagen.

Las operaciones morfológicas se realizaron con una de las más potentes bibliotecas que contiene Python, *OpenCV* [19]. Esta biblioteca permitió hacer la erosión y la dilatación de la imagen, pasándola antes al formato de *OpenCV*, *IPL*.

La operación de esqueletización se realizó con la importación de otra biblioteca *pymorph* [18], que pasando la imagen a binaria permitió obtener el esqueleto de la imagen.

También, se implementó gracias a la biblioteca *NumPy*, el recorte de la imagen cogiendo solo la parte del dígito escrito, como se había hecho en el primer paso. La posterior

normalización se realizó con la biblioteca *PIL* [16] y su método para redimensionar una imagen.

Estos procesamientos de la imagen no la dejan exactamente igual a la que se obtenía en MATLAB, sobretodo la diferencia se produce en la esqueletización. Esto puede que afecte en la clasificación de las imágenes ya que el entrenamiento y los pesos que se van a utilizar en la red neuronal son los obtenidos de MATLAB, pero este factor se podrá analizar más adelante (ver sección 5.6).

5.3. Extracción de características

En esta parte se trasladaron los algoritmos desarrollados en MATLAB para la extracción de características al lenguaje Python.

Gracias a la biblioteca *NumPy*, se transformaron todas las imágenes de los dígitos en matrices, por lo que se pudieron implementar todas las características, ya que la mayoría de ellas eran operaciones matriciales.

A continuación, se expone una breve descripción de la implementación de las características, indicando las bibliotecas necesarias para el desarrollo de cada uno de ellos:

1. Existencia de huecos en la imagen. Esta característica es la que tuvo más complejidad de desarrollo. Se necesitó la importación del paquete *ndimage* [31] de la biblioteca *Scipy* [32], el cual tiene una función que busca los huecos de la imagen que se le pasa por parámetro, y otra que devuelve las coordenadas donde se sitúa un hueco dentro de la imagen. Previamente, se hizo el negativo de la imagen, ya que dichas funciones consideran que el trazo es un píxel negro y el fondo es blanco, y las imágenes que se utilizan son al contrario como ya se ha explicado anteriormente en el capitulo 4.3.

2. Proporción entre el ancho y el alto. Se obtuvo el ancho y el alto de la imagen y se dividió ambos valores. Se tuvo que transformar estos valores de enteros a decimales para que la división fuera decimal, ya que sino el resultado era un entero y se perdía

mucha precisión, además de que el resultado no era el mismo que en el prototipo de la solución, en el que no era necesario expecificar el formato.

3. Diferencia entre los píxeles verticales. Al tener la imagen como una matriz y gracias a la biblioteca *NumPy*, se pudo dividir fácilmente cada imagen en tres columnas y sumar en cada una de ellas cuantos píxeles contenían para devolver el valor absoluto de la diferencia entre la primera y la última columna.

4. Diferencia entre los píxeles horizontales. Se utilizó el mismo método que en la anterior pero dividiendo la imagen en tres filas, y del mismo modo se obtuvo la diferencia del número de píxeles extre la primera y última fila.

5. Existencia de píxeles alrededor de la imagen. Esta característica también se ayuda de la biblioteca *NumPy*. Gracias a la suma de píxeles por filas y columnas se detecta si la mayoría de estos están alrededor de la imagen.

6. Máximo de píxeles en la última columna. Se divide la imagen en tres columnas, como se hizo en la característica 3 y se detecta si hay una existencia mayor de píxeles en la última columna comparándola con las otras dos.

7. Diferencia del número 4. La primera parte del desarrollo de esta característica es igual que la anterior, pero además se comprueba que el segundo máximo de píxeles verticales está en la primera columna y que el máximo de píxeles horizontales esta la parte del centro de la imagen.

8. Diferencia entre las mitades horizontales de la imagen. Se divide la imagen horizontalmente en dos partes. Se suman los píxeles de cada una de ellas y se devuelve la diferencia entre ellos.

9. Diferencia del número 7. Se detecta dónde está el máximo de píxeles horizontales y verticales y de comprueba que los horizontales está en la parte superior y que los verticales no se encuentran en la zona de la izquierda de la imagen. Además se le da una mayor valor si el siguiente máximo de píxeles horizontales se sitúa en el centro de la imagen aproximadamente.

10. Simetría vertical del dígito. Se divide la imagen en dos columnas y se comprueba si existe simetría entre ambas partes.

11. Simetría horizontal del dígito. Se realiza de la misma manera que se ha hecho para la característica anterior pero dividiendo la imagen en dos filas.

12. Existencia de píxeles en el centro de la imagen. Al igual que en la característica 5 se devuelve la diferencia de la suma de los píxeles que se encuentran alrededor de la imagen con respecto a los que se encuentra en el centro.

Como se explicará posteriormente (sección 5.6), después de entrenar y obtener los resultados con las 12 características, no se alcanzó la tasa de acierto que se consiguió en el prototipo de la solución, por lo que se decidió implementar más características que fueran sencillas, con el objetivo de intentar mejorar los resultados. Se decidió incluir 9 características más que resultaban de dividir cada imagen en 9 partes, 3 en horizontal y otros 3 en vertical, con la ayuda de la biblioteca *NumPy*, y sumar los píxeles que contenían cada una de ellas (ver Figura 5.1). Se especifican las características de la 13 a la 21 como el sumatorio de los píxeles de la partes primera, segunda, tercera, etc ,hasta la novena.

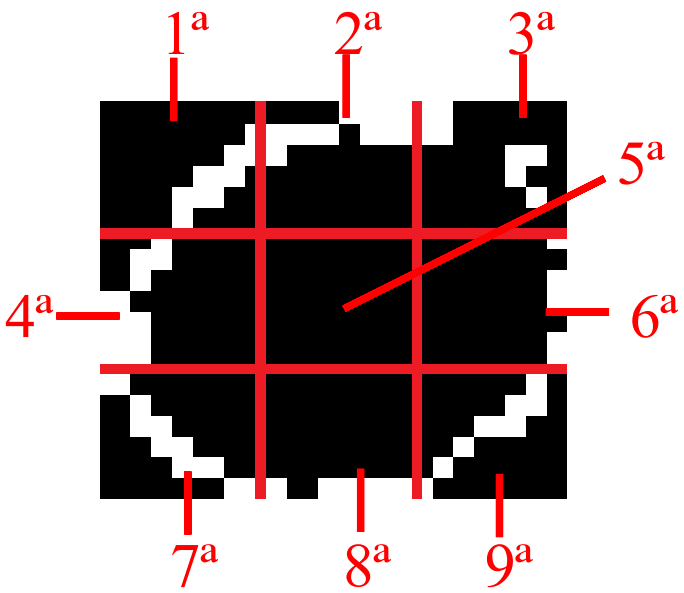


Figura 5.1: Ejemplo de una imagen dividida en las 9 partes

Por tanto, en la solución final implementada en Python se extraen 21 características.

Como se observa en la descripción de la implementación de la características, la mayoría se pudieron implementar gracias a las bibliotecas adicionales de Python, sobre todo a *NumPy*.

Como ya se ha dicho antes (sección 5.2), las etapas de procesamiento de la imagen no devuelven la misma imagen que retornaba MATLAB, y por tanto el vector de características resultante no será el mismo. Este factor habrá que evaluarlo posteriormente (sección 5.6) cuando dicho vector se introduzca en la red neuronal y se obtengan los resultados finales.

5.4. Red Neuronal

Hasta este momento, lo que se ha hecho ha sido traspasar los métodos tal y como fueron implementados en MATLAB para el prototipo de la solución al lenguaje Python para así poder integrarlo en el sistema Eyegrade. Pero ahora no se va a entrenar la red neuronal, ya que los pesos de cada capa se han extraído del prototipo de la solución, y son los que se van a utilizar en la red neuronal implementada en Python.

Por tanto, en esta etapa se crea la red neuronal leyendo los datos de los pesos y cada vez que se vaya a reconocer un dígito se pasará el vector de entradas, con las 21 características finales de la imagen, por la red neuronal para obtener el dígito reconocido.

Para implementar la red neuronal, se leen los pesos junto con sus umbrales de los archivos donde se han guardado, creando las matrices que representan las capas de la red neuronal ($IW \in \mathcal{M}_{40x21}$ para la primera capa oculta, $LW1 \in \mathcal{M}_{20x40}$ para la segunda capa oculta, y $LW2 \in \mathcal{M}_{20x10}$ para la tercera y última capa o capa de salida) y sus umbrales ($\overline{a} = (a_1, \ldots, a_{40})^T$ para los umbrales de la primera capa oculta, $\overline{b} = (b_1, \ldots, b_{20})^T$ para los de la segunda capa oculta y $\overline{c} = (c_1, \ldots, c_{10})^T$ para la capa de salida).

Una vez se tienen todas las matrices de las capas ocultas, se realiza el producto matri-

cial de la primera capa oculta, IW, con el vector que se ha obtenido de la extracción de características $\overline{x} = (x_0, \ldots, x_{21})^T$. Una vez se obtiene el vector resultado, se suma el vector de umbrales, $\overline{a} = (a_1, \ldots, a_{40})^T$. De esta forma se obtiene el vector $\overline{y} = (y_0, \ldots, y_{40})^T$

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{40} \end{pmatrix} = \begin{pmatrix} IW_{1,1} & IW_{1,2} & \ldots & IW_{1,21} \\ IW_{2,1} & IW_{2,2} & \ldots & IW_{2,21} \\ \vdots & \vdots & \ddots & \vdots \\ IW_{40,1} & IW_{40,2} & \ldots & IW_{40,21} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{21} \end{pmatrix} + \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_{40} \end{pmatrix} \tag{5.1}$$

El resultado se introduce en la función de activación (Ecuación 4.3), para normalizar el vector $\overline{z} = (z_1, \ldots, z_{40})^T$ entre los valores [-1,+1](ver Ecuación 5.2).

$$\begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_{40} \end{pmatrix} = \begin{pmatrix} tanh(y_1) \\ tanh(y_2) \\ \vdots \\ tanh(y_{40}) \end{pmatrix} \tag{5.2}$$

Una vez hecho esto, se vuelve a repetir los mismos pasos (ver Ecuaciones 5.3 y 5.4) pero utilizando los resultados del anterior paso (resultado de la Ecuación 5.2).

$$\begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_{20} \end{pmatrix} = \begin{pmatrix} LW1_{1,1} & LW1_{1,2} & \ldots & LW1_{1,40} \\ LW1_{2,1} & LW1_{2,2} & \ldots & LW1_{2,40} \\ \vdots & \vdots & \ddots & \vdots \\ LW1_{20,1} & LW1_{20,2} & \ldots & LW1_{20,40} \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_{40} \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{20} \end{pmatrix} \tag{5.3}$$

$$\begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_{20} \end{pmatrix} = \begin{pmatrix} tanh(r_1) \\ tanh(r_2) \\ \vdots \\ tanh(r_{20}) \end{pmatrix} \tag{5.4}$$

Por último se vuelve a repetir los mismos pasos (ver Ecuaciones 5.5 y 5.6) con la matriz que se corresponde a la capa de salida de la red neuronal, $LW2$, obteniendo un vector con los pesos que le corresponden a cada dígito, $\overline{v} = (v_1, \ldots, v_{10})^T$.

$$\begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_{10} \end{pmatrix} = \begin{pmatrix} LW2_{1,1} & LW2_{1,2} & \ldots & LW2_{1,20} \\ LW2_{2,1} & LW2_{2,2} & \ldots & LW2_{2,20} \\ \vdots & \vdots & \ddots & \vdots \\ LW2_{10,1} & LW2_{10,2} & \ldots & LW2_{10,20} \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_{20} \end{pmatrix} + \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_{10} \end{pmatrix} \tag{5.5}$$

$$\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{10} \end{pmatrix} = \begin{pmatrix} tanh(t_1) \\ tanh(t_2) \\ \vdots \\ tanh(t_{10}) \end{pmatrix} \tag{5.6}$$

La posición, empezando desde 0, del vector resultante de la Ecuación 5.6 que obtenga un mayor peso será el dígito que la red neuronal ha clasificado para la imagen que se le pasó al inicio del reconocedor de dígitos (OCR).

5.5. Integración con Eyegrade

Una vez implementado todos los módulos del reconocedor de dígitos (OCR), como el preprocesamiento de la imagen, extracción de sus características y la red neuronal como último paso, se debe integrar con el sistema Eyegrade.

El módulo OCR que tiene este sistema inicial, devuelve la decisión que ha tomado el módulo y un vector con los pesos que tiene cada dígito, introduciéndole la imagen y las coordenadas donde se sitúa el número que debe evaluar. Por tanto nuestro módulo debe devolver lo mismo para no modificar el código en el sistema Eyegrade.

El nuevo módulo OCR necesita igual que el anterior la imagen y las coordenadas para

recortar la imagen y obtener el número que se quiera evaluar. Por tanto, en esta parte se puede utilizar la que ya está implementada en el módulo OCR que tenía el sistema.

Con respecto a la salida que espera el sistema Eyegrade, se puede devolver la salida de la red neuronal que es un vector con los pesos de cada dígito. Como también se debe devolver dígito decidido, se implementó una función que devuelve dicho dígito. Ambos resultados son los que espera Eyegrade y devolviéndose en el mismo formato que el antiguo módulo de OCR.

Por tanto, la integración con el sistema Eyegrade se pudo realizar sin dificultad, implementando una pequeña función y devolviendo el vector resultante que retorna la red neuronal junto con el dígito decidido por ella.

5.6. Resultados

En esta sección se pretende evaluar los resultados obtenidos en la nueva implementación del reconocedor de caracteres, OCR, para el sistema Eyegrade, comparándolos con los obtenidos en el prototipo de la solución y con los resultados que se obtenían del anterior módulo de reconocimiento de dígitos de Eyegrade.

5.6.1. Comparación con el prototipo de la solución

Para hacer la comparación con el prototipo de la solución se hicieron varias evaluaciones.

Primero se evaluó que la red neuronal implementada en el nuevo módulo del sistema Eyegrade diera el mismo resultado que el que se obtenía en el prototipo de la solución. Para ello se extrajeron las características de unas imágenes de prueba en el sistema Eyegrade y se grabaron en un fichero. Estas características se introdujeron tanto en el prototipo de la solución implementado en MATLAB, como en la red neuronal implementada en Python para el sistema Eyegrade y se compararon los resultados. Se pudo observar que

para ambas redes el resultado obtenido era el mismo.

Como segunda evaluación se comprobó si ante una misma imagen las características extraídas de ella daban los mismos resultados en ambos sistemas. Este punto ya se había mencionado anteriormente (sección 5.2), dando un resultado negativo. Para comprobarlo se obtuvieron las características de unas imágenes de prueba en ambos sistemas y se compararon los resultados observando que no eran iguales. Las mayores diferencias se encuentran en las características que utilizan la imagen esqueletizada, ya que MATLAB tiene una función que se ajusta mejor al esqueleto real de la imagen que Python (ver Figura 5.2 en la que se observa las diferencias entre ambos sistemas).

Figura 5.2: Ejemplo de la aplicación a una imagen de la esqueletización en ambos sistemas

Ante la anterior evaluación, se pensó de qué manera iba a afectar este resultado al módulo del sistema final que se iba a introducir en Eyegrade. El no obtener las mismas características en ambos sistemas, suponía que los datos extraídos del prototipo de la solución (pesos de las capas ocultas y sus umbrales, ver sección 5.4) para utilizarlo en el sistema final no valdrían. La solución que se pensó ante este problema fue extraer las características de las imágenes del sistema Eyegrade y entrenar la red neuronal desde MATLAB. De esta manera, se evitaría implementar el entrenamiento de la red neuronal en Python, que es lo más complicado de realizar de las redes neuronales. Aún así, quedaba evaluar si con los nuevos valores de las características el resultado obtenido por la red neuronal iba a ser igual que el que se obtuvo en el prototipo de la solución.

Por tanto, la tercera comprobación que se hizo fue evaluar si el entrenamiento con las características obtenidas con el sistema Eyegrade daban unos resultados parecidos a los obtenidos en el prototipo de la solución. Al hacer el entrenamiento con la red neuronal que

mejores resultados obtuvo en el prototipo (ver sección 4.6) y utilizando las características 1 a 12 presentadas en la sección 5.3, se obtuvo una tasa de acierto de 45,73 % para el conjunto de entrenamiento, un 32,40 % para el conjunto de test y un 36,20 % para el conjunto de validación (ver Figura 5.3).

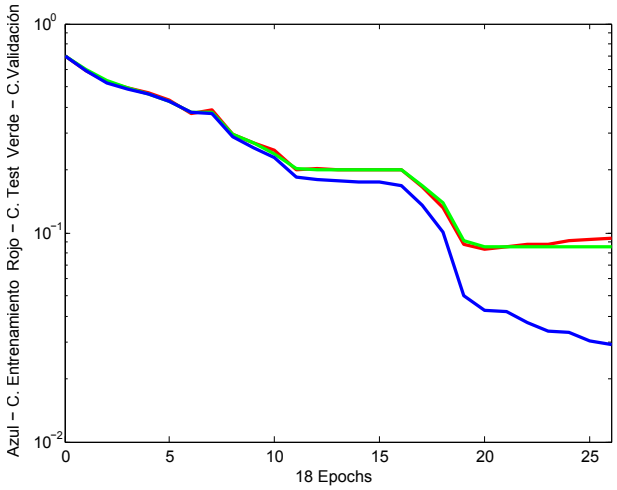


Figura 5.3: Representación del error en el entrenamiento para la red del prototipo de la solución con las 12 características extraídas desde Python

Estos resultados no son mejores que los que se obtuvieron en el Experimento 5 del prototipo de la solución (sección 4.6.6). Las tasas de acierto de todos los conjuntos son bastante menores que las que se obtuvieron en el prototipo.

Por tanto, se puede decir que las diferencias en la extracción de las características sí afectan de forma negativa al entrenamiento de la red y por consiguiente también a los resultados finales.

Se pensó una forma de paliar este inconveniente y mejorar los resultados. Una posible solución era incorporar más características. Por ello, se añadieron las características de la 13 a la 21, explicadas en la sección 5.3.

Con todas las características se volvió a entrenar la red y se volvió a evaluar el sistema. La tasa de acierto que se obtuvo con las 21 características fue un 76,01 % para el conjunto de entrenamiento, un 45,76 % para el conjunto de test y un 48,51 % para el conjunto de validación.

Estos resultados, incluyendo las 9 características, no son mejores que los obtenidos con el mejor prototipo que se obtuvo para la solución, pero son más cercanos que los resultados obtenidos con las 12 características iniciales. Finalmente, se implementaron estos últimos cambios para integrarlos en el sistema final de Eyegrade.

5.6.2. Comparación con el modulo de reconocimiento de dígitos del sistema Eyegrade

Para hacer la comparación entre el módulo de reconocimiento que posee en el sistema Eyegrade y el sistema finalmente desarrollado en este proyecto, se introdujeron las mismas imágenes en ambos módulos, y se midió la tasa de acierto.

Para el módulo del sistema de Eyegrade se alcanzó una tasa de 63,20 % para la identificación de dígitos aislados, es decir para la reconocimiento de dígitos individuales, no de los números completos que contienen más de un dígito como los identificadores de los alumnos. Para el nuevo módulo implementado con redes neuronales, con los últimos cambios de introducir 9 características más, se consiguió una tasa de 45,76 % para los mismos dígitos y mismas condiciones con las que se extrajo la tasa de acierto del módulo inicial.

Por tanto, el módulo inicial del sistema obtiene una mejor tasa de acierto que el implementado con la red neuronal, y no se consigue el objetivo de este trabajo, que era mejorar dicha tasa.

Capítulo 6

CONCLUSIONES Y LÍNEAS FUTURAS DE TRABAJO

Finalmente, en este último capítulo se recogen las conclusiones extraídas del trabajo realizado, así como las líneas futuras de trabajo para mejorarlo.

6.1. Conclusiones

Este trabajo pretendía mejorar la tasa de acierto de un módulo de reconocimiento de dígitos manuscritos que estaba implementado en el sistema Eyegrade. Para abordar este problema, se pensó en utilizar redes neuronales dadas sus ventajas y el aumento en el uso que se está haciendo de ellas, como se explicó en el capitulo 2.

Para utilizar las redes neuronales se debían introducir en ella unas imágenes de prueba. Había diferentes maneras de introducirlas, y al final se optó por extraer de cada imagen una serie de características comunes que permitieran diferenciar los números del 0 al 9.

Después de decidir como se iban a tratar las imágenes y como se iban a extraer las características, se planteó el problema de como entrenar las diferentes redes neuronales. Se buscó un entorno que permitiera experimentar con diferentes redes, entrenarlas y eva-

luar su tasa de acierto. Finalmente, el entorno escogido fue MATLAB porque posee un paquete para redes neuronales que hacen su implementación muy sencilla, además del potencial en cálculo. Una vez escogidos los parámetros y las características más adecuadas se implementaría en el lenguaje de Python para integrarlo con el sistema Eyegrade.

Se realizaron diferentes pruebas en el prototipo hecho en MATLAB, cuyos resultados se pueden observar en el capítulo 4.6. Después de obtener una tasa de acierto inferior a la obtenida por el sistema Eyegrade, se intentó mejorarla con la incorporación de dos características. Con esta mejora el mejor resultado obtenido fue alrededor de un 68 % de tasa de acierto para el conjunto de test. Este resultado seguía siendo inferior a lo obtenido en el módulo inicial del sistema Eyegrade, que tenía una tasa de alrededor del 72 %, pero era la que más se acercaba.

Una vez decidida la red neuronal y los parámetros de extracción de las imágenes se implementó tanto la extracción de características como la red neuronal para poderla utilizar en el sistema Eyegrade. Se realizaron diferentes pruebas ya que el resultado de la extracción de características no devolvía lo mismo que el prototipo de la solución. Después de ir paso a paso intentando descubrir donde estaba el posible error se encontró el punto donde se encontraba la mayor diferencia. Se daba en una de las transformaciones que se hacen en las imágenes para extraer el esqueleto de los números.

Como las características no eran las mismas que las obtenidas en el prototipo de la solución, se volvió a entrenar la red con las nuevas características y se obtuvo la tasa de acierto (se pueden ver los resultados en el capitulo 5.6). La tasa de acierto final fue aproximadamente de un 20 %, menor que en el prototipo de la solución, y por tanto menor también que la obtenida en el módulo inicial del sistema.

Por tanto, el objetivo de este trabajo que era mejorar la tasa de acierto del módulo de reconocimiento de dígitos aplicando una tecnología diferente no se ha conseguido, ya que en el prototipo de la solución se consiguió llegar a una tasa parecida a la que tenía el módulo inicial, pero al implementarlo en Python para integrarlo con el sistema la tasa ha bajado considerablemente.

6.2. Líneas Futuras de trabajo

Las posibles líneas futuras por las que se puede seguir trabajando son:

- Mejorar el algoritmo que se utiliza para obtener el esqueleto de las imágenes. Este algoritmo tendría que optimizar la figura y conseguir una línea con un grosor de un píxel que dibujara el esqueleto de la imagen.
- Analizar y extraer más características o mejorar las que ya se han implementado para diferenciar los diferentes dígitos que escriban los alumnos.

Si aún con estas mejoras no se obtiene una mejor tasa de acierto, se podría seguir avanzando de la siguiente manera:

- Probar a aumentar el número de neuronas en cada capa. Si aún así no se mejora la tasa se podría incrementar el número de capas añadiendo otra más para poder tener mayores relaciones entre las neuronas y conseguir mejorar la tasa de acierto de los dígitos final.
- En caso de que con la red neuronal de backpropagation no se obtuvieran mejores resultados en la tasa, se debería de probar a desarrollar otro tipo de neuronas como Hoffman que también son muy utilizadas en este tipo de problemas.

APÉNDICES

APÉNDICE A

PRESUPUESTO DEL PROYECTO

En este apéndice se presentan justificados los costes globales de la realización de este Proyecto Fin de Carrera.

La realización de este proyecto comenzó en el mes de marzo de 2011 y concluye en enero de 2012, por lo que se ha prolongado durante 10 meses. El trabajo se ha compatibilizado con un trabajo de media jornada durante los 4 primeros meses y otro trabajo de jornada completa durante los últimos 6 meses de duración del proyecto.

El proyecto se puede dividir en varias fases, las cuales se indican a continuación:

1. Estudio del problema a resolver y familiarización con el sistema Eyegrade: se ha llevado a cabo durante el mes de marzo y abril empleando unas 150 horas.

2. Desarrollo del software: esta parte junto con la siguiente son la parte central del proyecto. Se ha empleado aproximadamente unos 2 meses que suman alrededor de 150 horas.

3. Análisis de los datos: esta parte es la que más tiempo llevo ya que se debían de analizar los datos obtenidos y compararlos con los que se tenían. En esta fase se llevó a cabo en 5 meses sumando alrededor de 450 horas.

4. Redacción de la memoria del proyecto: corresponde a la fase final de la realización

del proyecto y se llevó a cabo durante los últimos 2 meses, sumando alrededor de 150 horas.

Los costes imputables a gastos de personal y de material, se pueden deducir de las Tablas A.1 y A.2.

Tabla A.1: *Fases del Proyecto*

Fase 1	*Estudio del problema y familiarización con el sistema Eyegrade*	150 *horas*
Fase 2	*Desarrollo del software*	150 *horas*
Fase 3	*Análisis de la base de datos*	450 *horas*
Fase 4	*Redacción de la memoria del proyecto*	150 *horas*
Total		900 ***horas***

En la Tabla A.1 se muestran las fases del proyecto y el tiempo aproximado para cada una de ellas. Así pues, se desprende que el tiempo total dedicado por el proyectando ha sido de 900 horas, de las cuales aproximadamente 60 horas han sido compartidas con el tutor del proyecto, por lo que el total asciende a 960 horas. Estableciendo unas tarifas de 60 €/hora, el coste de personal se sitúa en 21.600 €(para realizar el cómputo del presupuesto dedicado a la retribución del proyectando se ha supuesto un salario de 20€/hora, y para el caso del tutor, de 60 €/hora).

Tabla A.2: *Costes de material*

Concepto	**Precio**	**Amortización**	**Importe**
Ordenador de gama media	1.099	$\frac{1}{4}$	274,75€
Software de programación (MATLAB)	3.900	$\frac{1}{4}$	975€
Documentación			89,35€
Total			**1.339,10 €**

En la Tabla A.2 se recogen los costes de material desglosados en equipo informático, el software de programación utilizado y documentación. Se ha tenido en cuenta un coeficiente de amortización de $\frac{1}{4}$ según la vida media de cada concepto que así lo requiera. Ascienden, pues, a un total de 1.339,10 €.

A partir de estos datos, el presupuesto total es el mostrado en la Tabla A.3.

Tabla A.3: *Presupuesto*

Concepto	**Importe**
Costes personal	21.600 €
Costes material	1.339,10 €
Costes indirectos	4587,82 €
Base imponible	27.526,92 €
I.V.A. (18 %)	4.954,85 €
TOTAL	32.481,77 €

El coste total del proyecto asciende a **treinta y dos mil cuatrocientos ochenta y uno con setenta y siete** euros

Bibliografía

[1] Bonifacio Martín del Brío and Alfredo Sanz Molina. *Redes Neuronales y Sistemas Borrosos*. RA-MA Editorial, 3ª edition, 2006.

[2] Lotfi A. Zadeh. Fuzzy Logic. *IEEE Computer Magazine*, 1988.

[3] Hans Moravec. *Mind Children. The Future of Robot and Human Intelligence*. Harvard University Press, 1988.

[4] John E. Hopcroff. Máquinas de Turing. *Ciencia e Investigación*, 1984.

[5] Carver Mead and Mohammed Ismail. *Analog VLSI implementation of neural systems*. Kluwer Academic Publishers, 1989.

[6] Paul M. Churchland and P. Smith Churchland. ¿Podría pensar una máquina? *Investigación y Ciencia*, (162), Marzo 1994.

[7] David E. Rumelhart and James L. McClelland. Foundations. In *Parallel Distributed Processing*.

[8] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. Mit Press, 1969.

[9] Robert J. Marks II. Intelligence: Computational Versus Artificial. In *IEEE Transactions on Neural Networks*. 1993.

[10] Santiago Ramón y Cajal. *Textura del Sistema Nervioso del Hombre y de los vertebrados*. N. Moya, Madrid, 1899-1904.

[11] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 1943.

[12] Berndt Müller and Joachim Reinhardt. *Neural Networks: An Introduction*. Springer-Verlag, 1990.

[13] José R. Hilera and Víctor J. Martínez. *Redes neuronales artificiales. Fundamentos, modelos y aplicaciones*. RA-MA Editorial, 1995.

[14] The MathWorks Inc. MATLAB and Simulink for Technical Computing. Último acceso: 20/01/2012. http://www.mathworks.com/.

[15] Python Software Foundation. The Python Programming Language. Último acceso: 20/12/2011. http://www.python.org/.

[16] Python Software Foundation. Python Imaging Library (PIL). Último acceso: 11/11/2011. http://www.pythonware.com/products/pil/.

[17] Scientific Computing Tools For Python: Numpy. Último acceso: 04/11/2011. http://numpy.scipy.org/.

[18] Luis Pedro Coelho, Roberto A. Lotufo, and Rubens C. Machado. PYMORPH: Image Morphology in Python. Último acceso: 30/10/2011. http://packages.python.org/pymorph/.

[19] Free Software Foundation. OpenCV 2.1 Python Reference. Último acceso: 13/11/2011. http://opencv.willowgarage.com/documentation/python/index.html.

[20] Jesús Arias Fisteus. Eyegrade: Grading multiple choice exams with low-cost and portable computer-vision techniques. Último acceso: 12/10/2011. http://www.it.uc3m.es/jaf/eyegrade/.

[21] Arturo de la Escalera Hueso. *Visión por computador. Fundamentos y métodos*. Prentice Hall, 2001.

[22] Jesús Cid Sueiro. Departamento de Teoría de la Señal de la Universidad Carlos III de Madrid. IMAGine: Cursos Interactivos de Tratamiento Digital de Imagen. Operaciones espaciales. Último acceso: 19/10/2011. http://www.tsc.uc3m.es/imagine/Curso_ProcesadoBasico/index.html.

[23] Jesús Cid Sueiro. Departamento de Teoría de la Señal de la Universidad Carlos III de Madrid. IMAGine: Cursos Interactivos de Tratamiento Digital de Imagen. Operaciones morfológicas. Último acceso: 19/10/2011. http://www.tsc.uc3m.es/imagine/Curso_ProcesadoMorfologico/index.html/.

[24] Robert J. Schalkoff. *Pattern Recognition: Statistical, Structural And Neural Approaches*. Wiley India Pvt. Ltd., 2009.

[25] José C. Principe, Neil R. Euliano, and W.Curt Lefebvre. *Neural and Adaptive Systems: Fundamentals through Simulations*. Wiley, 2000.

[26] Paul J. Werbos. Backpropagation through time: What it does and how to do it. In *Proceedings of the IEEE*. 1990.

[27] D. Randall Wilson and Tony R. Martínez. The General Inefficiency of Batch Training for Gradient Descent Learning. *Neural Networks*, 2003.

[28] Chris M. Bishop. Neural networks and their applications. *Review of Scientific Instruments*, 1994.

[29] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999.

[30] Python Software Foundation. The ImageFilter Module for Python. Último acceso: 05/11/2011. http://www.pythonware.com/library/pil/handbook/imagefilter.htm.

[31] The SciPy Packages: Ndimage. Último acceso: 06/11/2011.

[32] The SciPy library. Último acceso: 21/10/2011. http://www.scipy.org/.