PROCESAMIENTO DE IMÁGENES CON REDES NEURONALES

TRABAJO FIN DE GRADO

Curso 2022/2023

UNIVERSIDAD
COMPLUTENSE
MADRID

FACULTAD DE CIENCIAS MATEMÁTICAS

GRADO EN INGENIERÍA MATEMÁTICA

AUTOR

DANIEL CAMPOS SÁNCHEZ

DIRECTORES

D. ANTONIO LÓPEZ MONTES (UCM)
Dª. MARÍA TERESA BENAVENT MERCHÁN (UCM)
D. ANTONIO MARTINEZ RAYA (UNED/UPM)
D. NICOLÁS ANTEQUERA RODRIGUEZ (UCM)
D. JOSÉ ÁNGEL GONZÁLEZ PRIETO (UCM)

Madrid, 15 de febrero de 2023

RESUMEN

Este trabajo reúne las pautas necesarias para entender los métodos y algoritmos matemáticos que las redes neuronales emplean para procesar imágenes.

En primer lugar, se introduce el concepto de neurona artificial, así como un ejemplo para ilustrar sus partes y su funcionamiento a nivel práctico para el tratamiento de imágenes.

En la segunda parte del trabajo se introduce el concepto de red neuronal artificial, así como la importancia de las funciones de activación y el problema de clasificación. También se detalla matemáticamente el proceso de aprendizaje de las redes neuronales artificiales, introduciendo los algoritmos de *Backpropagation* y gradiente descendente.

En la tercera parte del trabajo, se introducen las redes neuronales convolucionales, debido a su importancia para el procesamiento de imágenes con redes neuronales. Se detalla su arquitectura y se explica la función de los diferentes tipos de capas que las componen.

En la parte final, se aplica la teoría matemática de las redes neuronales en el procesamiento de imágenes con tres casos prácticos utilizando MATLAB.

PALABRAS CLAVE

Neurona artificial, Red neuronal artificial, Función de activación, Problema de clasificación, *Backpropagation*, Gradiente descendente, Redes neuronales convolucionales.

ABSTRACT

This project gathers the necessary guidelines to understand the mathematical methods and algorithms that neural networks use to process images.

Firstly, the concept of artificial neuron is introduced, along with an example to illustrate its parts and practical level operation for image processing.

In the second part of the project, the concept of artificial neural network is introduced, as well as the importance of activation functions and the classification problem. The mathematical process of learning in artificial neural networks is also detailed, introducing the Backpropagation and gradient descent algorithms.

In the third part of the project, Convolutional Neural Networks are introduced due to their importance in image processing with neural networks. Their architecture is detailed and the function of the different types of layers that make them up is explained.

In the final part, the mathematical theory of neural networks is applied to image processing with three practical cases using MATLAB.

KEYWORDS

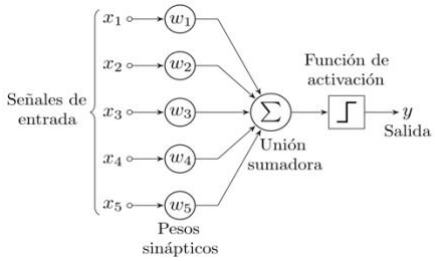
Artificial neuron, Artificial neural network, Activation function, Classification problem, Backpropagation, Gradient descendent, Convolutional neural networks.

ÍNDICE

ÍNDICE DE FIGURAS Y TABLAS

CAPÍTULO 1. INTRODUCCIÓN

Las redes neuronales artificiales surgen en los años 40 del siglo XX, como un intento de imitar el funcionamiento de las neuronas del cerebro humano[1]. Se muestra en la Tabla 1 los avances históricos en el campo de las redes neuronales:

Autor	**Época/año**	**Introducción**	**Observaciones**
Santiago Ramón y Cajal.	Finales del siglo XIX.	Doctrina de la neurona.	Santiago Ramón y Cajal, médico y científico español, Premio Nobel de medicina en 1906. Desarrolló la doctrina de la neurona, basada en que el tejido cerebral está compuesto por células individuales, las neuronas.[2]
Warren McCulloch y Walter Pitts.	1943	Primer modelo matemático simple de una red neuronal[3].	Dieron lugar al surgimiento de la teoría de las redes neuronales.
Frank Rosenblatt.	1958	Perceptrón[4].	
Bernard Widrow y Ted Hoff.	1960	ADALINE[N].	El ADALINE es un modelo de red neuronal artificial basado en el perceptrón, con la diferencia de que su objetivo es minimizar el error cuadrático medio entre la salida real y la salida esperada.

Marvin Minsky y Seymour Papert.	1969	"Perceptrons: An Introduction to Computational Geometry"[5].	En este libro, publicado en 1969 por Minsky y Paper, se presentaban las limitaciones del Percetrón de Rossenblatt. Esto causó gran frustración entre la comunidad científica y trajo consigo un periodo en el cual se dejó de hacer nuevas propuestas[6].
Paul Werbos.	1974	Método de aprendizaje *Backpropagation*.	La introducción del *Backpropagation* ha tenido un impacto significativo en el campo de la inteligencia artificial y la investigación en redes neuronales[1].
Kunihiko Fukushima.	1975 y 1980	Cognitrón[7] y Neocognitrón[8].	La introducción del Cognitrón y el Neocognitrón conformarán el origen de las redes neuronales convolucionales.

Tabla 1: introducción histórica a las redes neuronales artificiales

Desde la época de los 80 y hasta nuestros días, la redes neuronales se convirtieron en una herramienta importante en la investigación y aplicación de la visión por ordenador y de la inteligencia artificial en general. Hoy en día, las redes neuronales son una tecnología clave en una amplia variedad de aplicaciones, desde el reconocimiento de imágenes y voz hasta la traducción automática y la detección de fraudes. La investigación continúa en este campo, con el objetivo de desarrollar redes neuronales más precisas y eficaces.

1.1 La cognición visual

Figura 1.1[A] : imágenes que reconoce nuestro cerebro gracias a la percepción visual

Los seres humanos, al observar la figura 1.1, pueden reconocer tres animales en ella, un gato, un perro y un pájaro respectivamente. Esto se debe a un proceso cognitivo que lleva a cabo nuestro cerebro denominado **cognición o percepción visual**.

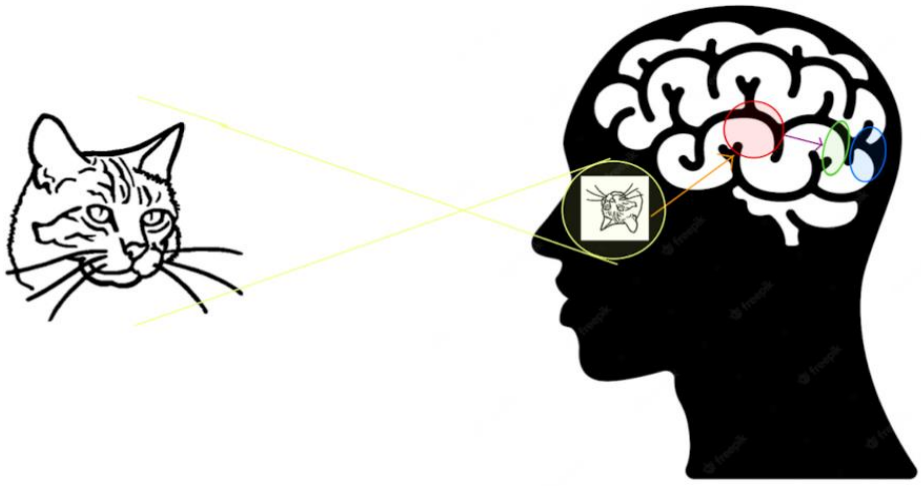


Figura 1.2[A][X] : representación del proceso de la percepción visual

En primer lugar, se debe saber que los objetos y las imágenes reflejan radiaciones luminosas de distinta frecuencia e intensidad que penetran en el interior del globo ocular a través de la pupila. Después esta señal luminosa pasa por la córnea, el cristalino y la cámara interior acuosa hasta llegar a la retina (círculo amarillo en figura 1.2), donde se proyectan las imágenes de forma invertida. Aquí, se transforma la luz en energía electroquímica que se transmite al cerebro, más concretamente al tálamo (círculo rojo en figura 1.2) a través del nervio óptico (flecha naranja en figura 1.2). Finalmente, gracias al núcleo geniculado lateral (flecha morada en figura 1.2) estos impulsos nerviosos llegan

a la corteza visual del cerebro (círculos verde y azul en figura 1.2) , situada en el lóbulo occipital, donde se produce la propia percepción[9].

Cada área de la corteza visual (círculos azul y verde en figura 1.2) extrae diferentes tipos de información de la entrada visual, desde la orientación o el contraste hasta el color o la forma de los objetos de la imagen. Es decir, un área de la corteza visual procesa formas simples como pueden ser líneas y orientaciones, a continuación otro área procesará conjuntos de esas líneas, después otro área procesará colores y texturas… Los autores de este artículo[10] resumen así este hecho:

"Hay una organización jerárquica entre las áreas visuales. Todas las áreas están altamente interconectadas, pero cada una de ellas está especializada en una parte del análisis de la información. La cognición visual es el resultado de interacciones recurrentes entre las distintas áreas visuales".

1.2 Neurona biológica vs neurona artificial

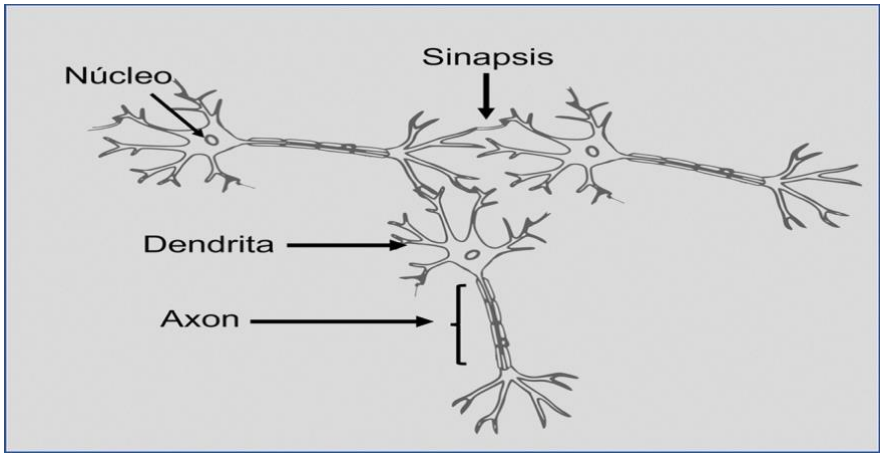


Figura 1.3[B] : neuronas biológicas

En primer lugar, para entender cómo funciona una neurona del cerebro humano y las conexiones entre ellas, se definen sus partes:

- **Dendritas**: es la parte de la neurona encargada de recoger la información, es decir, aquellos impulsos eléctricos y químicos procedentes de la neurona anterior.
- **Núcleo**: es la parte de la neurona donde se procesa la información recibida.
- **Axón**: se trata de una larga cola de la neurona encargada de transmitir la información a la siguiente neurona.

- **Sinapsis**: espacio entre dos neuronas, encargado de regular la transmisión de la información, o de los impulsos mediante unos elementos bioquímicos llamados neurotransmisores.

"Los neurotransmisores liberados en la sinapsis al final de las ramificaciones axonales, es decir, en las ramas en las que se divide la neurona al final del axón, tienen un efecto directo sobre la transmisión del impulso eléctrico el cual la siguiente neurona recibirá en sus dendritas. La neurona receptora captará estas señales y las combinará consiguiendo así un cierto nivel de activación. En función de este nivel de activación, la neurona emitirá señales eléctricas con una intensidad determinada mediante impulsos a través del axón[11]".

Las neuronas artificiales tratan de modelar matemáticamente el funcionamiento de una neurona del cerebro humano.

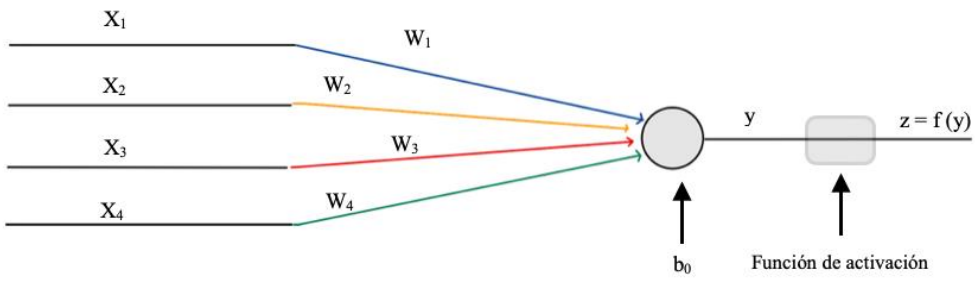


Figura 1.4[X] : ejemplo de neurona artificial

En el ejemplo de la figura 1.4 de neurona artificial se pueden observar los elementos que las componen y que se definen a continuación:

- **Entrada**: consta de un vector $\vec{X} = [X_1, X_2, X_3, X_4] \in \mathbb{R}^4$ que trata de modelar las dendritas.
- **Pesos sinápticos**: se trata de un vector $\overrightarrow{W} = [W_1, W_2, W_3, W_4] \in \mathbb{R}^4$ el cual trata de modelar la sinapsis.
- **Sesgo o bias**: se trata de un escalar $b_0 \in \mathbb{R}$.
- **Salida lineal**: consta de un escalar $y \in \mathbb{R}$ que se obtiene de la siguiente forma:

$$y = \sum_{i=1}^{4} W_i X_i + b_0 = W_1X_1 + W_2X_2 + W_3X_3 + W_4X_4 + b_0. \qquad (1.1)$$

- **Salida no lineal**: se trata de un escalar $z \in \mathbb{R}$ el cual se obtiene aplicando una función no lineal f (se explicará su importancia más adelante) a la salida lineal (1.1), la cual trata de modelar el nivel de activación, por ello se denominará **función de activación**, de tal manera que:

$$z = f(y) = f\left(\sum_{i=1}^{4} W_i X_i + b_0\right). \qquad (1.2)$$

Generalizando el ejemplo de la figura 1.4 a una neurona artificial con *n* entradas, se generalizan los anteriores conceptos de tal manera que:

- La entrada constará de un vector $\vec{X} = [X_1, \dots, X_n] \in \mathbb{R}^n$.
- Los pesos sinápticos se modelarán como un vector $\vec{W} = [W_1, \dots, W_n] \in \mathbb{R}^n$.
- El sesgo o bias será un escalar $b_0 \in \mathbb{R}$.
- La salida lineal se conformará como un escalar $y \in \mathbb{R}$ que se obtiene realizando la siguiente operación:

$$y = \sum_{i=1}^{n} W_i X_i + b_0 = W_1 X_1 + \dots + W_n X_n + b_0. \qquad (1.3)$$

- La salida no lineal que definirá como un escalar $z \in \mathbb{R}$ el cual se obtiene aplicando una función de activación f a la operación (1.3), de tal manera que:

$$z = f(y) = f\left(\sum_{i=1}^{n} W_i X_i + b_0\right). \qquad (1.4)$$

Se ilustrará a continuación, a través de un ejemplo con una imagen y datos cómo es el funcionamiento de una neurona artificial. Para este ejemplo se supone que se tiene una imagen que consta de cuatro píxeles como se ve en la figura 1.5:

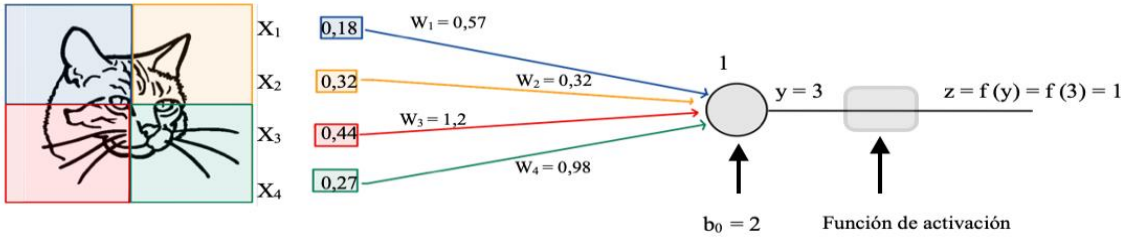


Figura 1.5[X] : ejemplo neurona artificial

1) La entrada consta de una imagen dividida en cuatro píxeles con valores de luminosidad 0.18, 0.32, 0.44 y 0.27, por ello, la entrada a la neurona será un vector $\vec{X} \in \mathbb{R}^4$, en este caso $\vec{X} = [0.18, 0.32, 0.44, 0.27]$

2) Se tendrá también un vector $\overrightarrow{W} \in \mathbb{R}^4$, el cual en este caso será $\overrightarrow{W} = [0.57, 0.32, 1.2, 0.98]$.
3) El sesgo será un parámetro, $b_0 \in \mathbb{R}$, el cual se tomará en este caso como $b_0 = 2$. El sesgo controla en qué rango numérico debe estar la salida lineal de la neurona para así facilitar o no la activación de la neurona a través de la función de activación. Un valor de sesgo alto permitirá que la activación de la neurona requiera una salida menor, por el contrario si se define un sesgo pequeño, se necesitará una salida mayor para que la neurona se active.

4) Para este ejemplo se utilizará una función de activación definida de la siguiente manera:

$$z = f(y) = \begin{cases} 0, & y < 2 \\ 1, & y \geq 2 \end{cases} \qquad (1.5)$$

De modo que si la salida lineal de la neurona es menor que 2, la neurona no se activará y su salida tomará el valor 0. Por el contrario si la salida lineal de la neurona toma un valor mayor o igual que 2, la neurona se activará y la salida tomará el valor 1.

5) Tal y como se definió en la ecuación (1.1) se puede calcular el valor de la salida no lineal de tal manera que:

$$y = \sum_{i=1}^{4} W_i X_i + b_0 = 0.57 \times 0.18 + 0.32 \times 0.32 + \\ +1.2 \times 0.44 + 0.98 \times 0.27 + 2 = 3.$$

6) Y por tanto el valor de la salida no lineal, atendiendo a la función (1.5) será:

$$z = f(y) = f(3) = 1.$$

De modo que, la neurona se activa y genera una salida con valor 1.

7) Por último, cabe destacar que si se hubiera tomado un valor para el sesgo $b_0 = 0.5$, se obtendría una salida lineal $y = 1.5$, en consecuencia $z = f(1.5) = 0$ y por tanto se obtendría que la neurona no se activaría generando así una salida con valor 0.

CAPITULO 2. REDES NEURONALES ARTIFICIALES

2.1. Introducción

Las **redes neuronales artificiales** son un modelo de inteligencia artificial que consiste en una combinación de neuronas artificiales organizadas en una estructura interconectada. Cada neurona recibe una entrada para posteriormente generar una salida a través de la función de activación. Esta salida se combina con la entrada de neuronas sucesivas hasta el final de la red.

Estas neuronas artificiales conectadas entre sí están agrupadas en diferentes niveles los cuales se denominarán capas:

"Una **capa** es un conjunto de neuronas cuyas entradas provienen de una capa anterior (o de los datos de entrada en el caso de la primera capa) y cuyas salidas son la entrada de una capa posterior[12]"

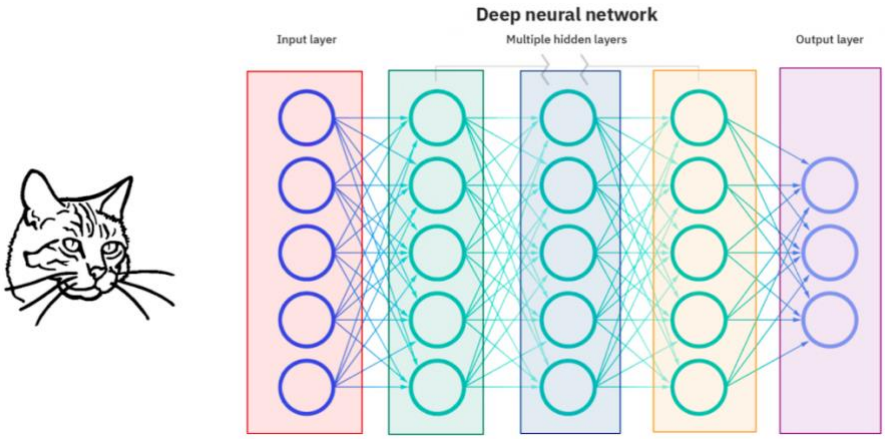


Figura 2.1[C][X] : ejemplo capas red neuronal

Existen tres tipos de capas dentro de una red neuronal:

- **Capa de entrada**: formada por neuronas que introducen los datos a procesar en la red.
- **Capas ocultas**: constituida por neuronas cuyas entradas vienen de neuronas de capas anteriores y cuyas salidas van a neuronas de capas posteriores.
- **Capa de salida**: se trata de neuronas cuya salida conformará la salida de la red neuronal.

2.2 Funciones de activación

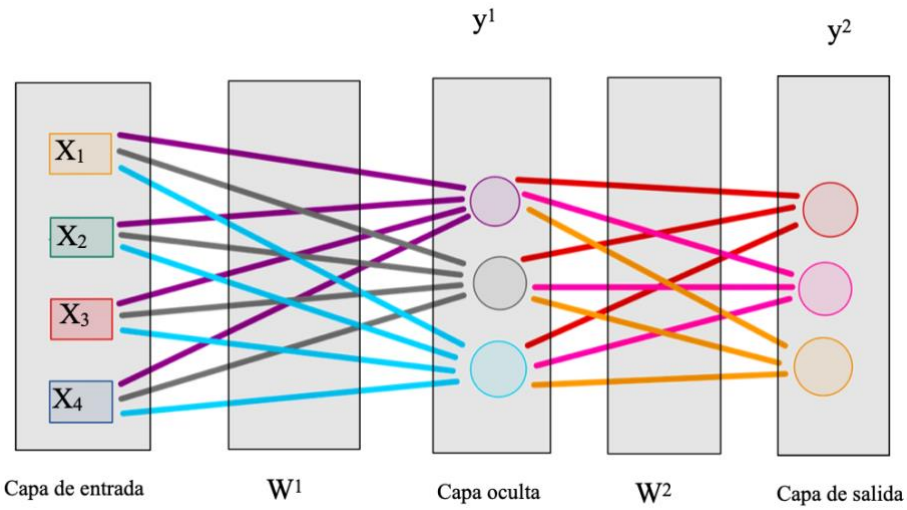


Figura 2.2[X] : red neuronal formada por tres capas

Para ilustrar el papel de las funciones de activación en una red neuronal se tomará la red de la figura 2.2. Siendo:

- $\vec{X}= [X_1, X_2, X_3, X_4] \in \mathbb{R}^4$ el vector de entrada a la red.
- $W^1 \in \mathcal{M}_{3\times4}(\mathbb{R})$ la matriz de pesos sinápticos que une la capa de entrada con la capa oculta, donde cada columna representa los valores de los pesos sinápticos que asocian cada valor de entrada con cada una las neuronas de la capa oculta.
- $W^2 \in \mathcal{M}_3(\mathbb{R})$ la matriz de pesos sinápticos que une la capa oculta con la capa de salida.
- $\vec{b}^i \in \mathbb{R}^3 \; \forall i = 1,2$ el sesgo o bias dado en forma de vector cuyas filas serán los valores del bias de las neuronas de cada capa. $i = 1$ si b pertenece a la capa oculta, $i = 2$ si b pertenece a la capa de salida.
- $\vec{y}^i \in \mathbb{R}^3 \; \forall i = 1,2$ el vector de salida lineal que representa el la información de salida de las neuronas de la última capa. $i = 1$ si y pertenece a la capa oculta, $i = 2$ si y pertenece a la capa de salida.

Supongamos que las neuronas de dicha red únicamente tienen salidas lineales tal y como se estableció en la ecuación (1.3), de tal modo que:

1) Para la salida de una neurona de la capa oculta se obtendría que:

$$\vec{y}^{1}_{(3\times1)} = W^{1}_{(3\times4)} \cdot \vec{X}_{(4\times1)} + \vec{b}^{1}_{(3\times1)}. \quad (2.1)$$

2) Y en consecuencia, para la salida de la siguiente neurona, la cual se encuentra en la capa de salida se tendría que:

$$\vec{y}^{2}_{(3\times1)} = W^{2}_{(3\times3)} \cdot \vec{y}^{1}_{(3\times1)} + \vec{b}^{2}_{(3\times1)} = W^{2}_{(3\times3)}\left[W^{1}_{(3\times4)} \cdot \vec{X}_{(4\times1)} + \vec{b}^{1}_{(3\times1)}\right] + \vec{b}^{2}_{(3\times1)}. \quad (2.2)$$

3) Operando:

$$\vec{y}^{2}_{(3\times1)} = W^{2}_{(3\times3)} \cdot W^{1}_{(3\times4)} \cdot \vec{X}_{(4\times1)} + W^{2}_{(3\times3)} \cdot \vec{b}^{1}_{(3\times1)} + \vec{b}^{2}_{(3\times1)}. \quad (2.3)$$

4) Por tanto, denotando como W^{3}al producto de W^{1} y W^{2}:

$$\vec{y}^{2}_{(3\times1)} = W^{3}_{(3\times4)} \cdot \vec{X}_{(4\times1)} + \left(W^{2} \cdot \vec{b}^{1}\right)_{(3\times1)} + \vec{b}^{2}_{(3\times1)}. \quad (2.4)$$

5) Y operando de nuevo, denotando a la suma del producto de W^{2} y $\vec{b}^{1}$ más $\vec{b}^{2}$ como $\vec{b}^{3}$:

$$\vec{y}^{2}_{(3\times1)} = W^{3}_{(3\times4)} \cdot \vec{X}_{(4\times1)} + \vec{b}^{3}_{(3\times1)}. \quad (2.5)$$

Por tanto comparando las ecuaciones (2.1) y (2.5) se concluye que para la salida lineal $\vec{y}^{2}$ de la capa de salida se realiza exactamente la misma operación que para salida de la capa oculta $\vec{y}^{1}$. Toda la operación que realiza la red neuronal de tres capas se reduce a una sola operación lineal. En conclusión, si todas las salidas de las capas fueran lineales, cualquier número de capas consecutivas serían equivalentes a una sola capa lineal, lo que limitaría la capacidad de la red para "aprender" (concepto que se verá más adelante) relaciones no lineales en los datos. Es decir, las funciones de activación no lineales permiten a las redes neuronales "aprender" a identificar patrones o tendencias en los datos que podrían pasar desapercibidos con un enfoque lineal. Un mayor número de capas en la red permitirá que la red aprenda a identificar más patrones en las imágenes de entrada. Además, las neuronas de las capas ocultas de la red responderán con una salida activa si la entrada recibida contiene algún patrón el cual han aprendido a reconocer, y no se activarán si la entrada no contiene dicho patrón.

A continuación se expondrán algunas de las funciones de activación más comunes.

2.2.1 Función Sigmoide y tangente hiperbólica

Históricamente la función sigmoide ha sido la más utilizada como función de activación para redes neuronales, consiste en que la salida y de una neurona pasa a través de la función:

$$\sigma(y) = \frac{1}{1 + e^{-y}} \quad (2.6)$$

Obteniendo así una salida no lineal.

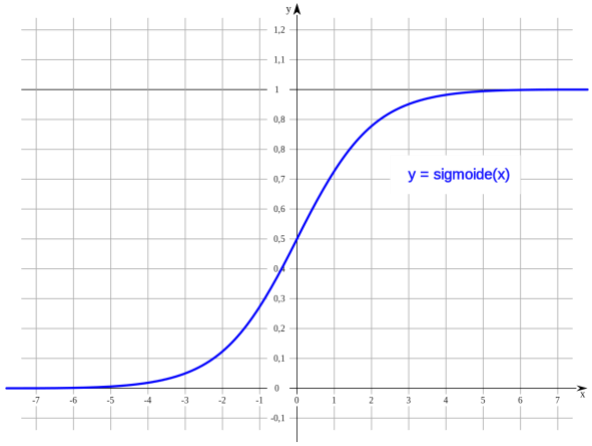
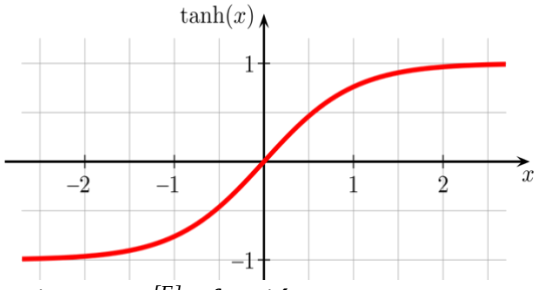


Figura 2.3[D] : función sigmoide

En 2010, Xavier Glorot y Yoshua Bengio, de la universidad de Montréal[15], realizaron un estudio concluyendo que el uso de funciones de activación sigmoidales no es óptimo para el aprendizaje de la red neuronal, ya que sus valores se acumulan rápidamente en 0 y en 1, ofreciendo así gradientes significativamente pequeños y relentizando así el aprendizaje por gradiente descendente (se verá este algoritmo más adelante).

Por estas razones propusieron el uso de la tangente hiperbólica ante las funciones sigmoidales, debido a que aunque el problema de acumulación seguía presente en esta función, la ventaja de ser simétrica facilitaba en gran medida el entrenamiento de la red.[16]



$$\tanh(y) = \frac{e^{-y} - e^{y}}{e^{-y} + e^{y}} \quad (2.7)$$

Figura 2.4[E] : función tangente hiperbólica

2.2.2 Función ReLU

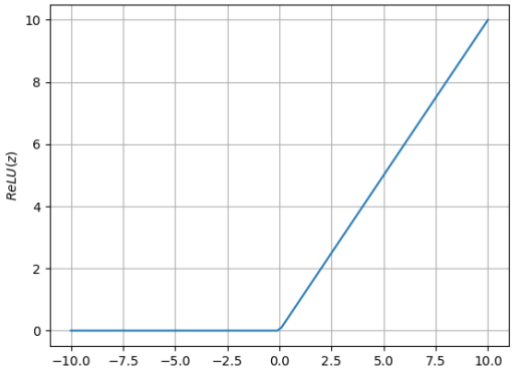


Figura 2.5[F] : función ReLU

La función ReLU (rectified linear unit) es una función de activación que esquiva los problemas de acumulación y además es más fácil de implementar computacionalmente debido a que no utiliza operaciones matemáticas complejas como la función exponencial. Actualmente se trata de la función de activación más utilizada en redes neuronales convolucionales.

$$f_{ReLU}(y) = \begin{cases} y & si\ y \geq 0 \\ 0 & si\ y < 0 \end{cases} = \max(0, y) \qquad (2.8)$$

2.3 Problema de clasificación de imágenes mediante una red neuronal.

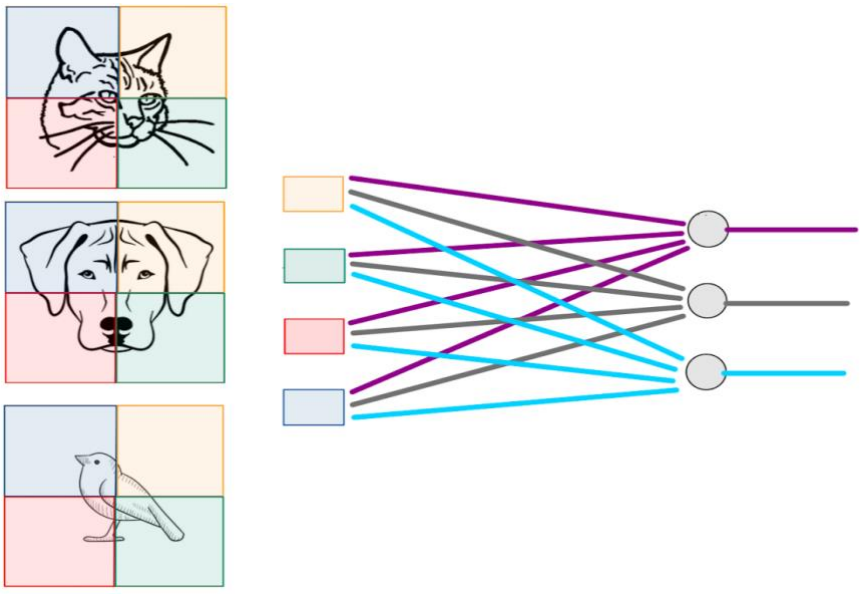


Figura 2.6[X] : ejemplo red neuronal simple para clasificación

Según la Real Academia Española, una **clase** se define como un conjunto de elementos con caracteres comunes. Un **problema de clasificación de imágenes** planteado como un problema con redes neuronales, es tal que dadas unas imágenes de entrada, la red clasifique estas imágenes de tal manera que las pueda agrupar en clases.

Para este ejemplo se utilizará la red neuronal de la figura 2.6, que únicamente cuenta con dos capas, una de entrada, y una de salida. Además se utilizarán tres imágenes de entrada de tres clases diferentes, en este caso: gato, perro y pájaro. Para la salida, la red cuenta con tres neuronas. Generalmente, para los problemas de clasificación que cuentan con K clases de imágenes se utilizan K neuronas de salida. La primera, de color morado, representará la salida de la clase gato, la del medio, de color gris, representará la salida de la clase perro y la última, de color azul, representará la salida de la clase pájaro.

Se definen a continuación los componentes que se usarán en el ejemplo:

- $\vec{X}^i \in \mathbb{R}^4 \ \forall i = 1,2,3$ los vectores de entrada a la red, formados por los valores numéricos de los píxeles de las imágenes que se procesarán. Si $i = 1$ será el vector numérico de la imagen de clase gato, si $i = 2$, el vector de la imagen de clase perro y si $i = 3$, el vector de la imagen de clase pájaro.
- $W \in \mathcal{M}_{3\times4}(\mathbb{R})$ la matriz de pesos sinápticos.
- $\vec{b} \in \mathbb{R}^3$ los valores de los sesgos de cada neurona en forma de vector.
- $\vec{y}^i \in \mathbb{R}^3 \ \forall i = 1,2,3$ los vectores de salida de la red cuyos elementos serán las salidas de cada una de las neuronas. Si $i = 1$ será el vector salida que genera la imagen de clase gato, si $i = 2$, la salida de la imagen de clase perro y si $i = 3$, la salida de la imagen de clase pájaro. De tal manera que:

$$\vec{y}^i = W \cdot \vec{X}^i + \vec{b}. \qquad (2.9)$$

Se realizan las operaciones de la red neuronal para las tres imágenes de entrada de tal manera que:

a) Para la imagen del gato:

0.57	0.32	1.20	0.98
0.15	-0.45	0.78	0.33
1.22	-0.15	0.66	-0.33

W

0.18
0.32
0.44
0.27

$\vec{X}^1$ $+$

2.00
2.21
2.31

$\vec{b}$ $=$

3.00
2.53
2.68

$\vec{y}^1$

b) Para la imagen de perro:

0.57	0.32	1.20	0.98
0.15	-0.45	0.78	0.33
1.22	-0.15	0.66	-0.33

W

0.73
0.17
0.25
0.13

$\vec{X}^2$ + | |
|---|
| 2.00 |
| 2.21 |
| 2.31 |

$\vec{b}$ = | |
|---|
| 2.90 |
| 2.48 |
| 3.30 |

$\vec{y}^2$

c) Para la imagen de pájaro:

0.57	0.32	1.20	0.98
0.15	-0.45	0.78	0.33
1.22	-0.15	0.66	-0.33

W

0.98
0.12
0.34
0.07

$\vec{X}^3$ + | |
|---|
| 2.00 |
| 2.21 |
| 2.31 |

$\vec{b}$ = | |
|---|
| 3.07 |
| 2.59 |
| 3.69 |

$\vec{y}^3$

Analizando las salidas, se observa que para la salida $\vec{y}^1$ la neurona que clasifica las imágenes de tipo gato devuelve el mayor de los valores, 3.00, mientras que la neurona que clasifica las imágenes de clase gato y la neurona de clase pájaro devuelven 2.53 y 2.68 respectivamente. Por este motivo se concluye que la red clasifica bien la entrada de la imagen de clase gato. De igual manera ocurre para la entrada de la imagen de clase pájaro, ya que la salida $\vec{y}^3$ obtiene el mayor de los valores, 3.69, en la neurona que clasifica las imágenes de clase pájaro. Por el contrario, en la salida $\vec{y}^2$ se observa que para la entrada de la imagen de clase perro, la neurona que clasifica las imágenes de clase perro obtiene el menor de los valores, 2.48, por tanto se concluye que la red no está clasificando bien todas las clases de imágenes.

Para concluir el problema de clasificación de imágenes se debe utilizar una función de activación de tipo SOFTMAX.

2.3.1 Función SOFTMAX

En los problemas de clasificación resulta conveniente interpretar las salidas de las neuronas como estimaciones de la probabilidad de que una imagen de entrada pertenezca a una de las clases del problema. Para hacer que la red neuronal represente una distribución de probabilidad definida sobre las diferentes clases del problema se introduce la función **SOFTMAX**:

$$z_j^i = \frac{e^{y_j^i}}{\sum_j e^{y_j^i}} \ . \qquad (2.10)$$

Donde:

- y_j^i es la j-ésima componente del vector $\vec{y}^i$.
- z_j^i es el resultado de aplicar la función SOFTMAX a la componente j-ésima del vector $\vec{y}^i$.

Volviendo al ejemplo con el que se está tratando, se obtiene que:

Dado $\vec{y}^1 = \begin{bmatrix} 3.00 \\ 2.53 \\ 2.68 \end{bmatrix}$, se le aplica la función SOFTMAX (2.9) de tal manera que:

$$y_1^1 = 3.00 \implies e^{3.00} = 20.08.$$
$$y_2^1 = 2.53 \implies e^{2.53} = 12.55.$$
$$y_3^1 = 2.68 \implies e^{2.68} = 14.58.$$

Y por tanto normalizando con la suma de estos valores se calcula que:

$$z_1^1 = \frac{20.08}{20.08 + 12.55 + 14.58} = 0.43.$$
$$z_2^1 = \frac{12.55}{20.08 + 12.55 + 14.58} = 0.27.$$
$$z_3^1 = \frac{14.58}{20.08 + 12.55 + 14.58} = 0.30.$$

De esta manera se concluye que:

$$\vec{z}^1 = SOFTMAX(\vec{y}^1) = SOFTMAX\left(\begin{bmatrix} 3.00 \\ 2.53 \\ 2.68 \end{bmatrix}\right) = \begin{bmatrix} 0.43 \\ 0.27 \\ 0.30 \end{bmatrix}.$$

Siendo $\vec{z}^i \in \mathbb{R}^3 \ \forall i = 1,2,3$ el vector obtenido al aplicar la función SOFTMAX al vector $\vec{y}^i \in \mathbb{R}^3$.

De tal manera que se obtendrían los siguientes valores de $\vec{z}$:

$$\vec{z}^1 = \begin{bmatrix} 0.43 \\ 0.27 \\ 0.30 \end{bmatrix}, \vec{z}^2 = \begin{bmatrix} 0.32 \\ 0.21 \\ 0.47 \end{bmatrix}, \vec{z}^3 = \begin{bmatrix} 0.29 \\ 0.18 \\ 0.53 \end{bmatrix}. \qquad (2.11)$$

Atendiendo a los valores de $\vec{z}$ dados en (2.11) se puede observar que en $\vec{z}^1$, la red devuelve una probabilidad del 43% de que la imagen de entrada del gato sea de clase gato, un 27% de que sea de clase perro y un 30% de que sea de clase pájaro. Por lo que la red clasifica bien la imagen de clase gato, al igual que lo hace con la de clase pájaro, visible en $\vec{z}^3$, donde se aprecia que la red clasifica correctamente la imagen de clase pájaro con una probabilidad del 53%. Por el contrario, cuando la imagen de entrada se trata de la imagen de clase perro, la red neuronal devuelve un 21% de probabilidad de que ésta sea de clase perro y un 47% de que ésta pertenezca a la clase pájaro. Por tanto se puede concluir que en este caso la red no está clasificando correctamente.

2.4 Entrenamiento y aprendizaje de la red neuronal artificial

Del ejemplo anterior surge la siguiente pregunta ¿cómo de buenos son los parámetros W y $\vec{b}$ para clasificar las imágenes? Teniendo en cuenta que el vector $\vec{X}$ está definido por el valor de los píxeles de la imagen, W y $\vec{b}$ son los parámetros que se pueden ajustar para mejorar la salida de la red. A esto se le denominará que la red neuronal "entrene" o "aprenda" a clasificar bien las imágenes de entrada.

2.4.1 Función de pérdida y función de coste.

Para entrenar una red neuronal se introduce la llamada **función de pérdida**, la cual representa cómo de buenos son los resultados arrojados con los parámetros W y $\vec{b}$ actuales[R]. Se representa como:

$$L^i = -\sum_{j=1}^{\#Clases} H_j \ln\left(z_j^i\right). \qquad (2.12)$$

Donde:

- L^ies el valor de la función de pérdida de la salida $\vec{z}^i$.
- $\#clases$ es el número de clases del problema ($\#clases = dim_{\mathbb{R}}(\vec{z}^i)$).
- $\vec{H} \in \mathbb{R}^{\#clases}$ se denominará al **vector de valores reales para una entrada**, es decir, será un vector de ceros con únicamente un uno en la posición en la que se encuentre la neurona que clasifique las imágenes de dicha entrada.
- H_j serán las componentes del vector $\vec{H}$.
- z_j^iserán las componentes del vector devuelto por la función SOFTMAX.

Aplicando esto al ejemplo con el que se venía tratando se obtiene que para la entrada de la imagen del gato, atendiendo a (2.11):

$\vec{z}^1 = \begin{bmatrix} 0.43 \\ 0.27 \\ 0.30 \end{bmatrix}$ y será $\vec{H} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ ya que se quiere que si la entrada es una imagen de clase gato, la red devuelva un 1 en la posición en la que se encuentra la salida de la neurona de clase gato, es decir en $j = 1$.

Aplicando ahora la función de pérdida se obtiene que la pérdida de la primera salida será:

$$L^1 = -\sum_{j=1}^{3} H_j \ln\left(z_j^1\right) = -(1 \times \ln(0.43) + 0 \times \ln\,(0.27 + 0 \times \ln\,(0.30)) = 0.84.$$

Y por tanto, aplicando la función de pérdida a las tres salidas obtenidas en el ejemplo se obtiene que:

$$L^1 = 0.84.$$
$$L^2 = 1.56.$$
$$L^3 = 0.63.$$

De tal manera que L^1 será la pérdida de la entrada de la imagen de clase gato, L^2la pérdida de la entrada de la imagen de clase perro, lo cual tiene sentido que sea la mayor ya que la red no la clasificaba correctamente, y L^3la pérdida de la entrada de la imagen de clase pájaro.

Se utiliza la función $-\ln(z_j^i)$ debido a que:

Figura 2.7[X] : gráfica de -ln(z_j^i)

Si z_j^i es un valor cercano a uno, quiere decir, que existe mucha probabilidad de que la red haya predicho bien a qué clase pertenece la imagen de entrada. Por tanto los valores W y $\vec{b}$ estarán bien ajustados, devolviendo así esta función una pérdida pequeña. Por ejemplo, si $z_j^i = 1$, la pérdida será 0. Si por el contrario z_j^i devuelve un valor más cercano a cero, quiere decir que la red ha predicho mal la clase de la imagen de entrada. En consecuencia, ofrecerá una pérdida mayor.

A continuación se introducirá la **función de coste**. Se denotará como:

$$J(W, \vec{b}) = \frac{1}{m}\sum_{i=1}^{m} L^i. \qquad (2.13)$$

Donde *m* será el número de imágenes introducidas en la red en el proceso de entrenamiento, de tal manera que realmente realiza la media aritmética de las pérdidas de las salidas que generan las distintas entradas. De hecho, atendiendo a la ecuación (2.11) se tiene que:

$$J(W, \vec{b}) = \frac{1}{m}\sum_{i=1}^{m} L^i = \frac{1}{m}\sum_{i=1}^{m}\left(-\sum_{j=1}^{\#Clases} H_j \ln(z_j^i)\right). \qquad (2.14)$$

Atendiendo a la ecuación (2.10) se obtiene que :

$$J(W, \vec{b}) = \frac{1}{m}\sum_{i=1}^{m} L^i = \frac{1}{m}\sum_{i=1}^{m}\left(-\sum_{j=1}^{\#Clases} H_j \ln\left(\frac{e^{y_j^i}}{\sum_j e^{y_j^i}}\right)\right). \qquad (2.15)$$

Y por último atendiendo a la ecuación (2.9) se obtiene:

$$J(W,\vec{b}) = \frac{1}{m}\sum_{i=1}^{m} L^i = \frac{1}{m}\sum_{i=1}^{m}\left(- \sum_{j=1}^{\#Clases} H_j \ln\left(\frac{e^{(W\cdot\vec{X}^i+\vec{b})_j}}{\sum_j e^{(W\cdot\vec{X}^i+\vec{b})_j}}\right)\right). \quad (2.16)$$

La función de coste agrupa todas estas pérdidas en un único valor. Depende de dos parámetros, W y $\vec{b}$, ya que, como se vio al principio del capítulo, son los únicos parámetros de la red los cuales pueden ajustarse.

2.4.2 Gradiente descendente y *Backpropagation*.

Se denominará entrenamiento **o aprendizaje** de la red al proceso que consiste en ajustar los parámetros W y $\vec{b}$, minimizando así la función de coste con el objetivo de que los resultados de la capa de salida se ajusten lo más posible a los resultados deseados.

El método más utilizado para ajustar estos parámetros es el algoritmo de gradiente descendente, que consiste en minimizar la función de coste. Este proceso se realiza calculando su variación con respecto de los parámetros que se quieren ajustar, para posteriormente, ir ajustando los parámetros en la dirección del gradiente.

Cabe destacar que el descenso del gradiente se aplica individualmente a todas las neuronas de la red por tanto podemos definir:

- $J(W,\vec{b})$ la función de coste.
- $W \in \mathcal{M}_{n\times m}(\mathbb{R})$ la matriz de pesos sinápticos.
- $\vec{b} \in \mathbb{R}^n$ el sesgo o bias.
- α la **razón de aprendizaje**, la cual define el tamaño del paso que se quiera dar en cada iteración del algoritmo.
- $N = \begin{bmatrix}\vec{b}\\ W\end{bmatrix}$ serán los parámetros de la neurona que quieres ajustarse.

El proceso que sigue es:

1) En primer lugar se calcula la variación de la función de coste J respecto de los parámetros que se quieran ajustar, en este caso con respecto a W y $\vec{b}$, se denotará por g al vector compuesto por dichas variaciones, es decir:

$$g = \nabla N = \begin{bmatrix} \frac{\partial J}{\partial \vec{b}} \\ \frac{\partial J}{\partial W} \end{bmatrix}. \qquad (2.17)$$

2) Se realiza un ajuste de los parámetros de forma iterativa de tal manera que:

$$N = N - \alpha(g). \qquad (2.18)$$

3) Se realiza esta iteración hasta que se encuentren los valores de W y $\vec{b}$ tal que la función de coste sea mínima.

A continuación se introduce el algoritmo *Backpropagation*, cuya finalidad es facilitar el cálculo de estas variaciones retropagando la variación de la función de coste con respecto a la salida. Para ilustrar cómo funciona este algoritmo, se aplicará en dos neuronas consecutivas pertenecientes a la capa de entrada y la capa de salida, respectivamente.

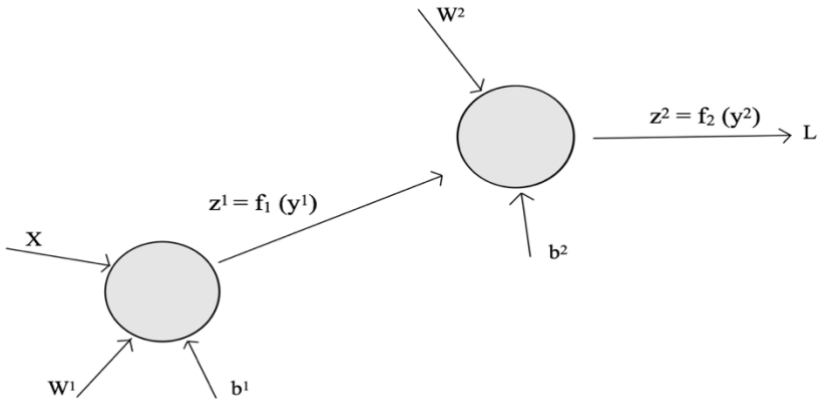


Figura 2.8[X] : dos neuronas consecutivas para Backpropagation

Donde:

- $\vec{X} \in \mathbb{R}^n$ será el vector de entrada de la red.
- $W^i \in \mathcal{M}_{n\times m}(\mathbb{R})\ \forall i = 1{,}2$ las matrices de pesos sinápticos de las capas 1 y 2 respectivamente.
- $\vec{b}^i \in \mathbb{R}^n\ \forall i = 1{,}2$ el sesgo o bias de la capa i.
- $\vec{y}^i \in \mathbb{R}^n\ \forall i = 1{,}2$ la salida lineal de la capa i.
- $\vec{z}^i \in \mathbb{R}^n\ \forall i = 1{,}2$ la salida no lineal de la capa i.
- f_i la función de activación de la capa i.

- $J = L$ la función de coste. Nótese que, en este caso, son iguales debido a que solo hay una neurona de salida.

Sea:

$$\vec{y}^1 = W^1 \cdot \vec{X} + \vec{b}^1. \qquad (2.19)$$

Y sea:

$$\vec{z}^1 = f_1(\vec{y}^1) = f_1\left(W^1 \cdot \vec{X} + \vec{b}^1\right). \qquad (2.20)$$

Además:

$$\vec{y}^2 = W^2 \cdot \vec{z}^1 + \vec{b}^2. \qquad (2.21)$$

Por tanto se tiene que:

$$\vec{z}^2 = f_2(\vec{y}^2) = f_2\left(W^2 \cdot \vec{z}^1 + \vec{b}^2\right). \qquad (2.22)$$

Por lo que se concluye que:

$$\vec{z}^2 = f_2(\vec{y}^2) = f_2\left(W^2 \cdot \vec{z}^1 + \vec{b}^2\right) = f_2\left(W^2 \cdot f_1\left(W^1 \cdot \vec{X} + \vec{b}^1\right) + \vec{b}^2\right). \qquad (2.23)$$

En primer lugar se debe calcular cuánto varía la función pérdida L si varía la salida de la neurona $\vec{z}^2$ de tal manera que se calculará:

$$\frac{\partial L}{\partial f_2}. \qquad (2.24)$$

A continuación se calcula la variación de la función de pérdida con respecto a los parámetros W^2 y b^2. Aplicando la regla de la cadena en la ecuación (2.22) se obtiene:

$$\frac{\partial L}{\partial W^2} = \frac{\partial L}{\partial f_2} \cdot \frac{\partial f_2}{\partial \vec{y}^2} \cdot \frac{\partial \vec{y}^2}{\partial W^2}. \qquad (2.25)$$

$$\frac{\partial L}{\partial \vec{b}^2} = \frac{\partial L}{\partial f_2} \cdot \frac{\partial f_2}{\partial \vec{y}^2} \cdot \frac{\partial \vec{y}^2}{\partial \vec{b}^2}. \qquad (2.26)$$

Se aplica el descenso del gradiente tal como se indica en la ecuación (2.18) obteniendo así:

$$W^2 = W^2 - \alpha\left(\frac{\partial L}{\partial W^2}\right) \quad \text{y de igual manera} \quad \vec{b}^2 = \vec{b}^2 - \alpha\left(\frac{\partial L}{\partial \vec{b}^2}\right).$$

A continuación interesa también saber cuánto varía la función de pérdida en función de los parámetros W^1 y $\vec{b}^1$, o lo que es lo mismo, cuánto varía la función de pérdida si varían los parámetros W^1 y $\vec{b}^1$. Para ello, se hará uso de la regla de la cadena aplicada a la ecuación (2.23), retropagando así la variación función de pérdida de la última capa a la capa anterior de tal manera que:

$$\frac{\partial L}{\partial W^1} = \frac{\partial L}{\partial f_2} \cdot \frac{\partial f_2}{\partial \vec{y}^2} \cdot \frac{\partial \vec{y}^2}{\partial f_1} \cdot \frac{\partial f_1}{\partial W^1}. \qquad (2.27)$$

$$\frac{\partial L}{\partial \vec{b}^1} = \frac{\partial L}{\partial f_2} \cdot \frac{\partial f_2}{\partial \vec{y}^2} \cdot \frac{\partial \vec{y}^2}{\partial f_1} \cdot \frac{\partial f_1}{\partial \vec{b}^1}. \qquad (2.28)$$

Y se puede aplicar de nuevo el descenso del gradiente:

$$W^1 = W^1 - \alpha\left(\frac{\partial L}{\partial W^1}\right) \quad \text{y de igual manera} \quad \vec{b}^1 = \vec{b}^1 - \alpha\left(\frac{\partial L}{\partial \vec{b}^1}\right).$$

Generalizando el ejemplo anterior, utilizando los elementos definidos para dos neuronas y considerando que hay $C = 1, \dots, M$ capas se puede describir el algoritmo del descenso del gradiente y *Backpropagation* como:

1) Calcular la variación o la derivada de la función de coste J con respecto a las salidas de la última capa:

$$\frac{\partial J}{\partial f_m}. \qquad (2.29)$$

2) Hallar la variación o derivada de la función de coste con respecto de los parámetros de las neuronas de la capa M, retropagando la variación de la función de coste con respecto a las salidas:

a) Respecto de los pesos sinápticos:

$$\frac{\partial J}{\partial W^m} = \frac{\partial J}{\partial f_m} \cdot \frac{\partial f_m}{\partial \vec{y}^m} \cdot \frac{\partial \vec{y}^m}{\partial W^m} . \qquad (2.30)$$

b) Respecto del bias:

$$\frac{\partial J}{\partial \vec{b}^m} = \frac{\partial J}{\partial f_m} \cdot \frac{\partial f_m}{\partial \vec{y}^m} \cdot \frac{\partial \vec{y}^m}{\partial \vec{b}^m} . \qquad (2.31)$$

3) Retropagar la variación de la función de coste de la última capa a capas anteriores, para así poder hallar las variaciones de la función de pérdida con respecto a los parámetros de capas anteriores:

$$\frac{\partial J}{\partial W^{m-1}} = \frac{\partial J}{\partial f_m} \cdot \frac{\partial f_m}{\partial \vec{y}^m} \cdot \frac{\partial \vec{y}^m}{\partial f_{m-1}} \cdot \frac{\partial f_{m-1}}{\partial W^{m-1}} . \qquad (2.32)$$

$$\frac{\partial J}{\partial \vec{b}^{m-1}} = \frac{\partial J}{\partial f_m} \cdot \frac{\partial f_m}{\partial \vec{y}^m} \cdot \frac{\partial \vec{y}^m}{\partial f_{m-1}} \cdot \frac{\partial f_{m-1}}{\partial \vec{b}^{m-1}} . \qquad (2.33)$$

4) Realizar el descenso del gradiente en cada neurona i de cada capa C de tal manera que:

$$g_i^c = \begin{bmatrix} \frac{\partial J}{\partial b_i^C} \\ \frac{\partial J}{\partial W_i^C} \end{bmatrix}. \qquad (2.34)$$

Y sea $N_i^C = \begin{bmatrix} b_i^C \\ W_i^C \end{bmatrix}$ los parámetros de entrada a la neurona, se repite de manera iterativa el proceso visto en el descenso del gradiente, de tal manera que:

$$N_i^C = N_i^C - \alpha(g_i^c) . \qquad (2.35)$$

CAPITULO 3. REDES NEURONALES CONVOLUCIONALES

Los orígenes de las redes neuronales convolucionales se fundamentan en el cognitrón[7] y en el neocognitrón[8], introducidos por Kunihiko Fukushima en 1975, bioinspirados en los estudios de David Hubel y Torsten Wiesel[13], por los que recibirían un premio Nobel en 1981. Este modelo introducido por Fukushima fue más tarde mejorado por Yann LeCun en 1998. Finalmente en el año 2012 este modelo fue perfeccionado por Dan Ciressan y otros y fueron implementadas para una unidad de procesamiento gráfico.[P]

Las redes neuronales convolucionales son un tipo de red neuronal artificial las cuales reciben su nombre debido a que utilizan la operación matemática de convolución, que tratan de imitar la corteza visual humana tal y como se vio en el apartado [1.1] de este trabajo. Son las redes más utilizadas para procesar señales. Normalmente se hace referencia a este tipo de redes con el acrónimo inglés CNN (Convolutional Neural Network).

"Una señal, es en principio, cualquier cantidad física detectable mediante la cual se pueden transmitir mensajes, es decir, podemos interpretar una señal como una variable que contiene información relevante sobre algún tema de interés para nosotros"[14].

En el caso de señales unidimensionales puede tratarse de una señal de audio, donde las entradas corresponden a muestras consecutivas en el tiempo. En el caso de señales bidimensionales, las entradas pueden corresponder a los píxeles de una imagen en blanco y negro.

En ocasiones, las señales pueden tener más de dos dimensiones, como pueden ser las imágenes a color, en las cuales se tendrán imágenes individuales para distintos canales como rojo, verde y azul.

La aplicación de las CNN es muy diversa, pero destacan aquellas que conllevan procesar imágenes. Por ejemplo, como hace la cámara de un coche autónomo, la cual dada una imagen instantánea puede reconocer los elementos que aparecen en ella como pueden ser otros coches, señales de tráfico, peatones o distinguir entre acera y calzada (detección de objetos). O también para reconocer qué tipos de objetos aparecen en una imagen (reconocimiento de objetos). Además, la salida de una red convolucional también puede ser numérica, por ejemplo para resolver problemas de clasificación (clasificación de imágenes) utilizando una capa con función de activación SOFTMAX (como se vio en el apartado [2.3.1]).[14]

En el siguiente capítulo se realizará un ejemplo de estas tres aplicaciones de redes neuronales convolucionales en MATLAB.

3.1 Arquitectura de las redes neuronales convolucionales.

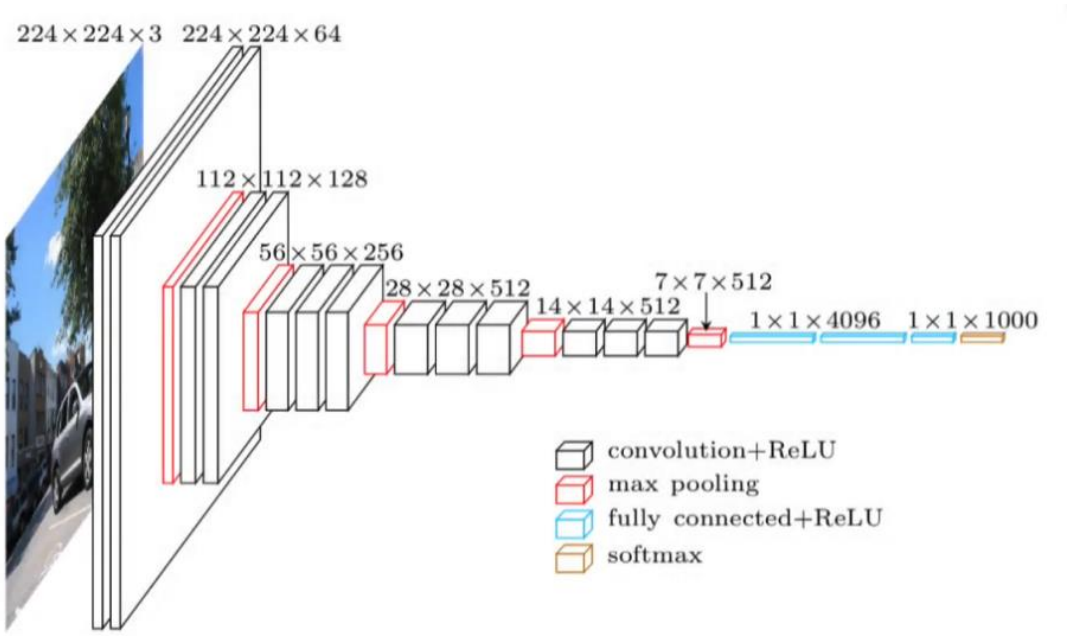


Figura 3.1[G]: arquitectura red neuronal convolucional

Una red convolucional está formada por una secuencia de capas.

- Capas convolucionales: capas que utilizan la operación matemática de convolución. Estas capas se conectan con la entrada de la red.
- Capas de pooling: su función principal será reducir la dimensión de la salida de las capas convolucionales.
- Capas fully connected: habitualmente colocadas al final de la red, reciben su nombre porque todas las neuronas de esta capa están conectadas a todas las neuronas de la capa anterior.

Como puede apreciarse en la figura 3.1, es habitual la utilización de funciones de activación entre las capas, así como la función SOFTMAX al final de la red si se trata de un problema de clasificación.

3.1.1 Convolución y capas convolucionales.

Para entender el proceso de convolución aplicado a imágenes, se comenzará con una imagen de entrada en blanco y negro, cuyos píxeles se pueden representar numéricamente en la matriz $I \in \mathcal{M}_{h\times n}(\mathbb{R})$ siendo $h \times n$ el tamaño de la imagen. Se toma en concreto para el ejemplo la matriz $I \in \mathcal{M}_4(\mathbb{R})$:

$$I = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 \\ -2 & 0 & 0 & -2 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

Se define un **filtro o kernel** como una matriz $K \in \mathcal{M}_k(\mathbb{R})$ la cual se irá colocando sobre la matriz de entrada tal y como se puede apreciar en la figura 3.2 y se realizará lo que se conoce como una operación de producto punto, es decir, se multiplicará valor por valor los elementos coincidentes de la matriz de entrada y del filtro para posteriormente realizar la suma de todos estos productos de tal manera que de cada convolución quede como resultado un único valor numérico. Nótese que en dimensión dos, donde la entrada está representada en forma de matriz se realiza el producto punto de 9 elementos.

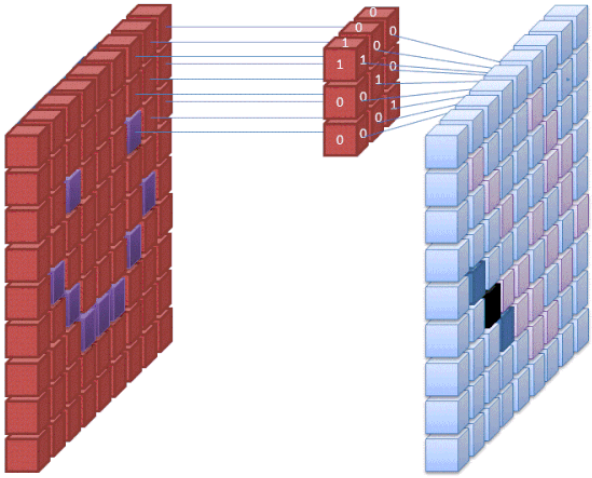


Figura 3.2[H] : ejemplo de convolución

Sea la matriz del filtro $K \in \mathcal{M}_3(\mathbb{R})$:

$$K = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Se define el **paso o zancada** que se denotará por $s \in \mathbb{N}$, el cual indica el salto que dará la matriz del filtro sobre la matriz de la imagen de entrada para ir realizando la operación de convolución, en este ejemplo se define un paso $s = 1$, de tal manera que tras haber realizado la primera convolución, la matriz del filtro se desplazará una unidad de píxel.

Además, se introduce el denominado **zero padding**, el cual consiste en rellenar la entrada con ceros alrededor de la matriz inicial con el objetivo de obtener una salida del tamaño deseado, se denotará por $p \in \mathbb{N}$. Por ejemplo, añadiendo un zero padding de $p = 1$ se obtendría una matriz de entrada $I_{zp} \in \mathcal{M}_{h+2p \times n+2p}(\mathbb{R})$.

Sea en este caso $I_{zp} \in \mathcal{M}_6(\mathbb{R})$:

$$I_{zp} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 1 & 1 & 1 & 0 \\ 0 & 1 & 2 & 2 & 2 & 0 \\ 0 & -2 & 0 & 0 & -2 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Se realizará la convolución, que se denotará con el signo $*$, de las matrices $I \in \mathcal{M}_4(\mathbb{R})$ y $K \in \mathcal{M}_3(\mathbb{R})$ de tal manera que:

$$I * K = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 \\ -2 & 0 & 0 & -2 \\ 1 & 0 & 1 & 0 \end{pmatrix} * \begin{pmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{pmatrix} =$$

$$= \begin{pmatrix} \begin{pmatrix} 2 \times 0 & +1 \times 1 & +1 \times 0 \\ +1 \times 1 & +2 \times 4 & +2 \times 1 \\ -2 \times 0 & +0 \times 1 & +0 \times 0 \end{pmatrix} & \begin{pmatrix} 1 \times 0 & +1 \times 1 & +1 \times 0 \\ +2 \times 1 & +2 \times 4 & +2 \times 1 \\ +0 \times 0 & +0 \times 1 & -2 \times 0 \end{pmatrix} \\ \begin{pmatrix} 1 \times 0 & +2 \times 1 & +2 \times 0 \\ -2 \times 1 & +0 \times 4 & +0 \times 1 \\ +1 \times 0 & +0 \times 1 & +1 \times 0 \end{pmatrix} & \begin{pmatrix} 2 \times 0 & +2 \times 1 & +2 \times 0 \\ +0 \times 1 & +0 \times 4 & -2 \times 1 \\ +0 \times 0 & +1 \times 1 & +0 \times 0 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 12 & 13 \\ 0 & 1 \end{pmatrix}$$

La matriz que resulta de aplicar la convolución se denotará por $C \in \mathcal{M}_{h_0 \times n_0}(\mathbb{R})$, en este caso $C \in \mathcal{M}_2(\mathbb{R})$.

Nótese que a esta matriz de salida se le podrá aplicar una función de activación de tipo ReLU a cada uno de sus elementos tal y como se podía ver en la figura 3.1.

Sea la la matriz de entrada $I \in \mathcal{M}_{h\times n}(\mathbb{R})$ y la matriz del filtro $K \in \mathcal{M}_k(\mathbb{R})$, sea $p \in \mathbb{N}$ el parámetro de zero padding y $s \in \mathbb{N}$ el paso o zancada y por último $C \in \mathcal{M}_{h_0\times n_0}(\mathbb{R})$ la matriz de salida, se obtendrá que:

$$h_0 = \frac{h-k+2p}{s} + 1. \qquad (3.1)$$

$$n_0 = \frac{n-k+2p}{s} + 1. \qquad (3.2)$$

Utilizando estas ecuaciones se calcula que para el ejemplo inicial se obtendrá una salida de tamaño:

$$h_0 = \frac{4-3+2(0)}{1} + 1 = 2.$$

$$n_0 = \frac{4-3+2(0)}{1} + 1 = 2.$$

De hecho, habiendo realizado los cáculos anteriores, la matriz de salida es en efecto $C \in \mathcal{M}_2(\mathbb{R})$, en cambio, si se realiza esta operación con la matriz de entrada $I_{zp} \in \mathcal{M}_6(\mathbb{R})$ se obtiene un tamaño de salida:

$$h_0 = \frac{4-3+2(1)}{1} + 1 = 4.$$

$$n_0 = \frac{4-3+2(1)}{1} + 1 = 4.$$

Concluyendo así que la matriz resultante de realizar la convolución $I_{zp} * K$ sería una matriz $C_{zp} \in \mathcal{M}_4(\mathbb{R})$.

En concreto este filtro utilizado K se denomina filtro de bordes, ya que su función es detectar los bordes entre colores diferentes de la imagen con el objetivo de detectar los contornos de los objetos. Realiza la siguiente modificación de la imagen:

Figura 3.3[I] : filtro de bordes

A continuación se verá como aplicar la convolución a imágenes a color. Una imagen a color cuenta con tres canales de entrada, verde azul y rojo. Por tanto, ahora la entrada no podrá representarse en una matriz, sino que se representará como una multimatriz de dimensión $(h \times n \times 3)$ y en consecuencia, se deberá aplicar un filtro de dimensión $(k \times k \times 3)$, como se aprecia en la siguiente imagen:

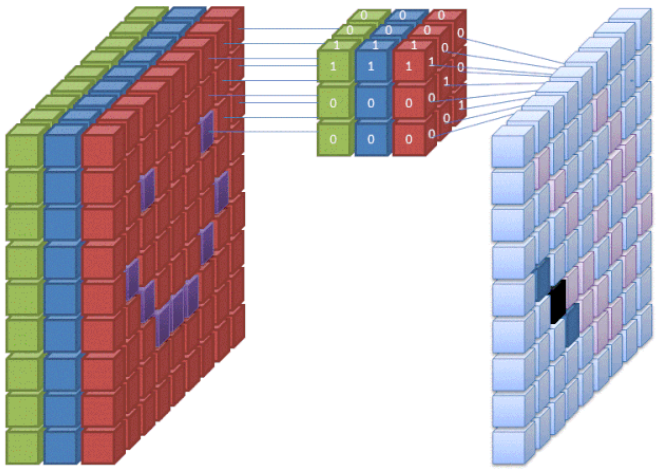


Figura 3.4[J] : convolución en imagen a color

La diferencia con respecto a las entradas en dos dimensiones será que ahora se realizará un producto punto de 27 elementos para realizar la convolución en vez de utilizar 9 elementos como se hacía en dimensión dos. Nótese que el resultado de cada convolución sigue siendo un único valor numérico. De nuevo a cada elemento de salida se le aplicará una función de activación no lineal de tipo ReLU y por ello al conjunto de elementos de salida se le denominará mapa de activación.

La dimensión, en este caso, del mapa de activación se podrá calcular utilizando de nuevo las ecuaciones (3.1) y (3.2) utilizadas anteriormente ya que el filtro deberá tener la misma profundidad que la entrada.

Por último se verá qué ocurre si se utiliza más de un filtro para una sola entrada.

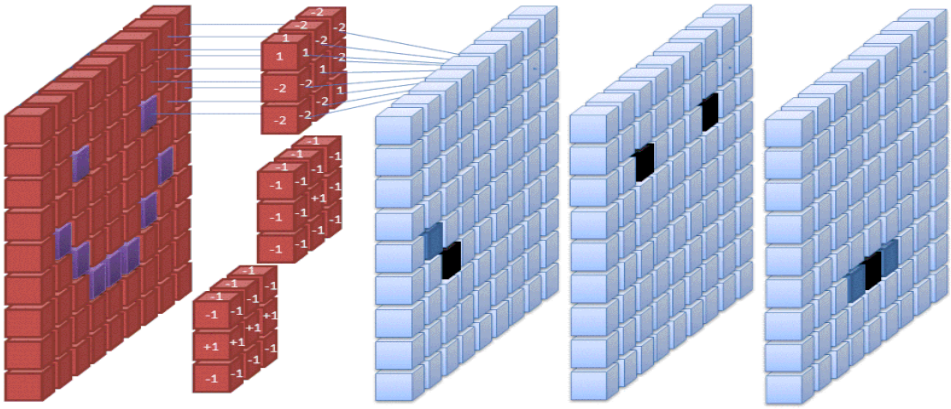


Figura 3.5[K] : aplicación de tres filtros de convolución

En este caso, la aplicación de más de un filtro, aunque la entrada y el filtro sean de dos dimensiones y se puedan representar en forma de matriz, daría como resultado una multimatriz. Las dimensiones de dicha multimatriz se podrían calcular utilizando las ecuaciones (3.1) y (3.2) para el alto y ancho de la salida, y siendo $n_f \in \mathbb{N}$ el número de filtros utilizados, la dimensión de la multimatriz de salida vendría dada por: $(h_0 \times n_o \times n_f)$. De nuevo se le aplica una función de activación y por tanto en este caso la salida vendría dada por n_f mapas de activación [14].

3.1.2 Capas de Pooling

Como se puede notar en la figura 3.1, las capas de pooling se colocan entre las capas convolucionales con el objetivo de reducir el coste computacional de una red neuronal utilizada para procesar imágenes.

Una capa de pooling lo que hace es reducir el tamaño de su entrada para que las capas posteriores puedan trabajar con datos de entrada reducidos y así reducir también el coste computacional de la red neuronal.

El pooling más utilizado se basa en combinar una serie de píxeles de la entrada utilizando una función máximo, por ello se denomina **max pooling**. Dada una matriz $I \in \mathcal{M}_{h\times n}(\mathbb{R})$, se le aplica un max pooling de $(m \times m)$, de tal manera que se divide la matriz I en submatrices de tamaño $(m \times m)$ de las cuales únicamente se quedará con el máximo valor de cada submatriz. Por ejemplo, volviendo a la matriz $I \in \mathcal{M}_4(\mathbb{R})$ del apartado anterior, se le puede aplicar un max pooling de (2×2) de tal manera que:

$$I = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \Longrightarrow I_{mp} = \begin{pmatrix} 2 & 2 \\ 1 & 1 \end{pmatrix}$$

Existen otros tipos de pooling, como el **average pooling** el cual se basa en realizar la media aritmética de los elementos de las submatrices, o el **L2 pooling**, el cual consiste en realizar la norma L2 de los elementos seleccionados[14].

3.1.3 Capas fully connected

Las capas fully connected se colocan al final de la arquitectura de la red neuronal convolucional. Es una capa de la red neuronal que está totalmente conectada con las neuronas que conforman la capa anterior. Esto significa que cada neurona de la capa fully connected está conectada con todas las neuronas de la capa anterior y cada conexión tiene un peso sináptico asociado. La salida de cada neurona de la capa fully connected se obtiene a través de una combinación lineal de las entradas y los pesos, seguida de una función de activación tal como se vio en el apartado 2.3 de este trabajo. [17]

Por ejemplo, si la capa anterior cuenta con 100 neuronas, y la capa fully connected cuenta con 10 neuronas, habrá un total de $100 \times 10 = 1000$ conexiones con sus respectivos pesos sinápticos entre las capas. La salida de las 10 neuronas de la capa fully

connected será una combinación lineal entre las salidas de las 100 neuronas de la capa anterior con los pesos sinápticos de las conexiones, seguida por una función de activación como puede ser la función SOFTMAX si se trata de un problema de clasificación con 10 clases de imágenes.

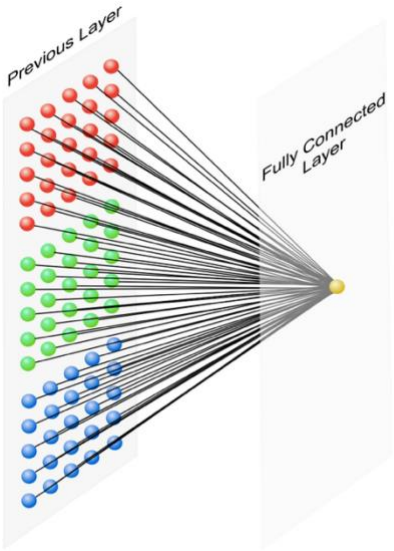


Figura 3.6[L] : capa fully connected con una neurona

3.2 Aplicaciones básicas

A continuación se verá un ejemplo de cada una de las aplicaciones de redes neuronales convolucionales para procesar imágenes más utilizadas actualmente, en primer lugar, se creará desde cero una red neuronal que nos permita clasificar imágenes, en este caso lo haremos con imágenes de números del 0 al 9. En segundo lugar se utilizará una red preentrenada que ofrece MATLAB para detectar objetos dentro de una imagen, lo cual se hará de manera que se simulará lo que ve un coche autónomo que va conduciendo por la calle y será capaz de reconocer y diferenciar todo lo que ve por su cámara frontal. Por último se hará uso de la red preentrenada Googlenet para un reconocimiento de objetos a través de la webcam del ordenador, donde se colocarán objetos aleatorios delante de la webcam del ordenador para que la red devuelva de qué objeto se trata.

3.2.1 Ejemplo 1. Clasificación de imágenes

Se creará la red neuronal desde cero, programando en MATLAB, la cual hará una clasificación de las imágenes de entrada que serán una serie de imágenes de números que van desde el cero hasta el nueve, en estas imágenes se representan números de distintos tamaños, formatos, grosores y orientaciones en blanco y negro[T]. Para empezar se cargará la base de datos ofrecida por MATLAB y a través de un bucle se le pedirá a MATLAB que enseñe veinte imágenes aleatorias de la base de datos:

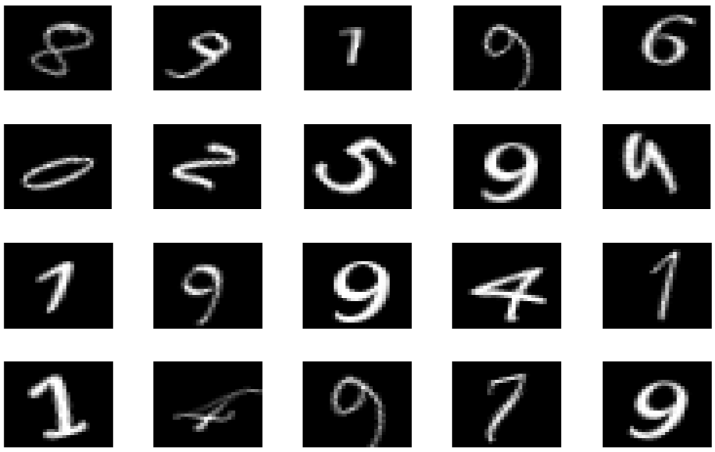


Figura 3.7: ejemplo de veinte imágenes de la base de datos

A continuación, se calculará el número de imágenes que contiene la base de datos, además también será conveniente calcular el número de imágenes que contiene cada clase, en este caso, el número de imágenes de cada número. Se hará en una tabla donde Label denota la clase y Count el número de imágenes de esa clase:

Label	Count
0	1000
1	1000
2	1000
3	1000
4	1000
5	1000
6	1000
7	1000

8	1000
9	1000

Tabla 2: imágenes de la base de datos

Cada clase contiene 1000 imágenes de cada número, es decir, diez mil imágenes en total. A continuación interesa saber cual es el tamaño de estas imágenes recordando que la capa de entrada debe ser del mismo tamaño que las imágenes que se procesarán. En este caso el tamaño resulta ser de 28×28 píxeles.

El siguiente paso será definir un conjunto de entrenamiento y otro de validación, el conjunto de entrenamiento será el utilizado para que la red neuronal entrene o aprenda, y el conjunto de validación será el que la red neuronal clasifique. En este caso se define un conjunto de entrenamiento de 750 imágenes, y las restantes conformarán el conjunto de validación.

Se define la arquitectura de la red neuronal convolucional:

1) Se define la capa de entrada, la cual debe ser del mismo tamaño que las imágenes, es decir, 28×28 .
2) Se aplica la primera capa convolucional con un filtro de 3×8. (Véase [3.1.1])
3) Se define una función de activación no lineal para la primera salida de tipo ReLU. (Véase [2.2])
4) Se aplica un max pooling 2×2 a la salida no lineal de la primera capa convolucional. (Véase [3.1.2])
5) Se aplica una segunda capa convolucional esta vez con un filtro 3×16 + ReLu.
6) Se aplica de nuevo un max pooling 2×2 para reducir el tamaño de la salida de la segunda capa convolucional.
7) Finalmente se aplica una tercera capa convolucional con un filtro 3×32 + ReLU.
8) Para poder clasificar se utiliza una capa fully connected de 10 neuronas. (Véase [3.1.3])
9) Se aplica una función SOFTMAX debido a que se buscan probabilidades para la clasificación. (Véase [2.3.1])

10) Por último se aplica una capa de clasificación la cual será la capa de salida.

Se debe especificar el entrenamiento de la red, lo cual se hará con el algoritmo del Descenso del gradiente (véase [2.4.2]), aplicado en MATLAB con "sgdm", con un $\alpha = 0{,}01$, es decir, con una razón de aprendizaje de 0,01. Además se establece un número máximo de ciclos de entrenamiento completo, en este caso cuatro.

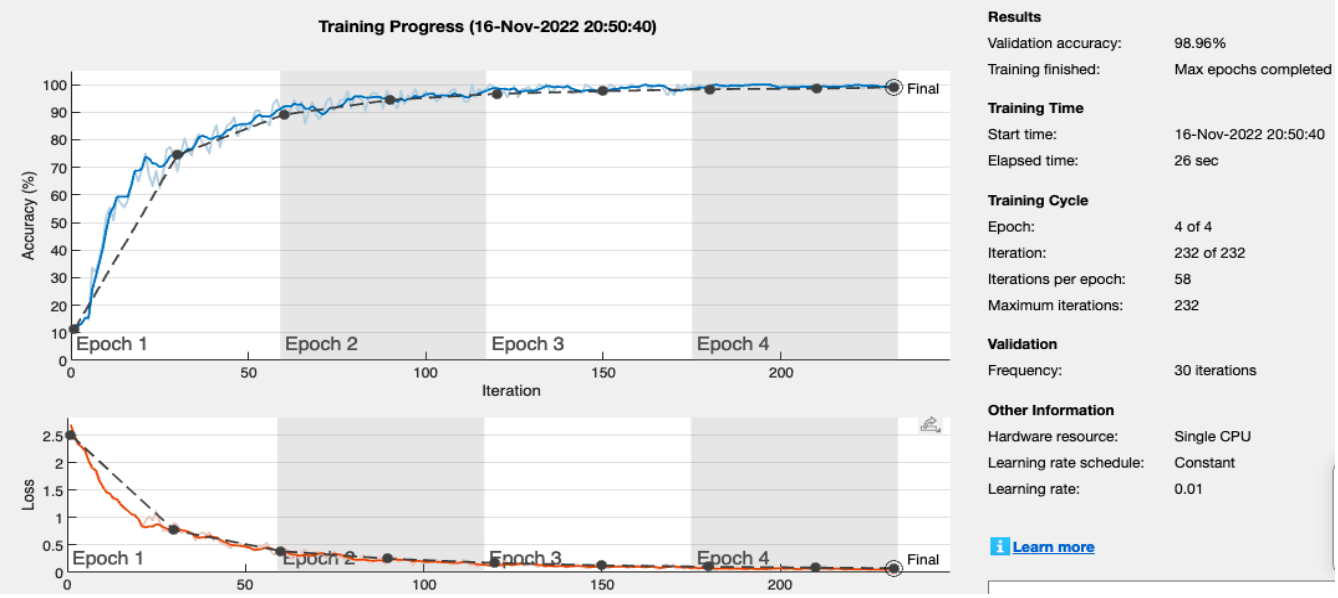


Figura 3.8 : gráfico de entrenamiento de la red

La figura 3.8 conforma el gráfico de cómo se ha desempeñado el entrenamiento de la red neuronal programada en MATLAB, el entrenamiento ha finalizado debido a que se ha realizado cuatro ciclos de entrenamiento completo, tal y como se le pidió en el código.

Cada ciclo de entrenamiento ha realizado 58 iteraciones para entrenar la red neuronal convolucional, de tal manera que en el gráfico de arriba se puede ver cómo va mejorando la precisión conforme se va aumentando el número de iteraciones de entrenamiento. Una vez entrenada y determinada la red, se aplica a los datos de validación y el resultado es que la red obtiene un 98,96% de acierto con los resultados de los datos de validación.

Por el contrario, en el gráfico inferior de la figura 3.8, se puede ver cómo disminuye la función de pérdida en función de las iteraciones, de tal manera que, gracias al entrenamiento, esta pérdida en las iteraciones finales se acerca a cero.

Cabe destacar que MATLAB ha tardado 26 segundos en realizar este entrenamiento de la red neuronal programada.

Una vez construida y entrenada la red neuronal convolucional, se hará uso de ella para clasificar tres imágenes pertenecientes al conjunto de validación. Estas imágenes serán las siguientes:

Figura 3.9: imágenes a procesar en la red neuronal

Introducidas las imágenes individualmente y de manera manual en la red. Devuelve a través de la consola de comandos que:

```
La clase a la que pertenece la imagen introducida es: 0
La clase a la que pertenece la imagen introducida es: 3
La clase a la que pertenece la imagen introducida es: 7
```

Figura 3.10: salida de la red neuronal convolucional

El código de MATLAB utilizado para este ejemplo puede verse en el anexo: Código de MATLAB 3.2.1. Clasificación de imágenes.

3.2.2 Ejemplo 2. Identificación de objetos.

Una de las aplicaciones más importantes de las redes neuronales convolucionales en el procesamiento de imágenes es la identificación de objetos. [U]. Uno de los usos más comunes es el funcionamiento de los coches autónomos. Son capaces de realizar su cometido gracias a la redes neuronales convolucionales y más concretamente a la identificación de objetos, ya que analizando las señales a través de su cámara frontal, son capaces de distinguir los coches, peatones, bicicletas… Así como diferenciar la calzada de la acera.

Para realizar esta implementación en MATLAB se hará uso de la red pre-entrenada Deeplabv3+. Además se hará uso de la base de datos CamVid (Cambridge-driving Labeled Video Database), ofrecida por MATLAB para sus usuarios. Esta base de datos (propiedad de la Universidad de Cambridge) consta de fotografías realizadas por cámaras coches autónomos en las calles de Cambridge.

La red preentrenada Deeplabv3+ utiliza la llamada **Depthwise Separable Convolution**, un tipo de convolución donde se aplica un único filtro para cada canal de entrada. Además utiliza el denominado **Spatial Pyramid Pooling (SSP)**, un tipo de pooling que se aplica en una etapa final de la red, más concretamente entre las capas convolucionales y las capas fully connected para así evitar la necesidad de reducir la imagen al principio de la red. Por último utiliza capas fully connected como las estudiadas[O].

En primer lugar se debe descargar la red preentrenada del servidor de Mathworks, desarrollador y distribuidor de MATLAB. Así como la base de datos del servidor web de la Universidad de Cambridge, donde se encuentran las imágenes.

A continuación, se carga la red neuronal y se le pide que indique los objetos que identificará en las imágenes. Se agrupan en las siguientes clases:

```
classes = 11×1 string
    "Sky"
    "Building"
    "Pole"
    "Road"
    "Pavement"
    "Tree"
    "SignSymbol"
    "Fence"
    "Car"
    "Pedestrian"
    "Bicyclist"
```

Para continuar, es necesario cargar la base de datos donde se encuentran las imágenes descargadas, aprovechando para pedir que devuelva como salida la imagen 559 de la base de datos, que será la figura 3.11.

Se procesará la imagen con la red neuronal pre-enrenada pidiendo que devuelva la misma imagen segmentada, es decir identificando cada objeto que aparece, lo cual se hará con un mapa a color de tal manera que generará la figura 3.12.

Figura 3.11 : imagen de la base de datos Camvid

Figura 3.12 : figura 3.11 aplicando la segmentación de imágenes

Para finalizar, puede interesar saber con qué frecuencia aparece cada objeto en la imagen, de tal manera que se analizarán las estadísticas para que devuelva en un histograma la frecuencia con la que los objetos aparecen en la imagen.

Como se puede observar en el histograma de la figura 3.13, la calzada ocupa cerca del 30% de la imagen, mientras que los edificios un 25% y el cielo algo más del 16%. Los siguientes objetos que aparecen con mayor frecuencia serían los árboles, cerca del 13%, la acera 7% y los coches 6%. Por último los objetos que aparecen con menor frecuencia serían postes, señales de tráfico, vallas de edificios, peatones y bicicletas, con una frecuencia menor al 5%.

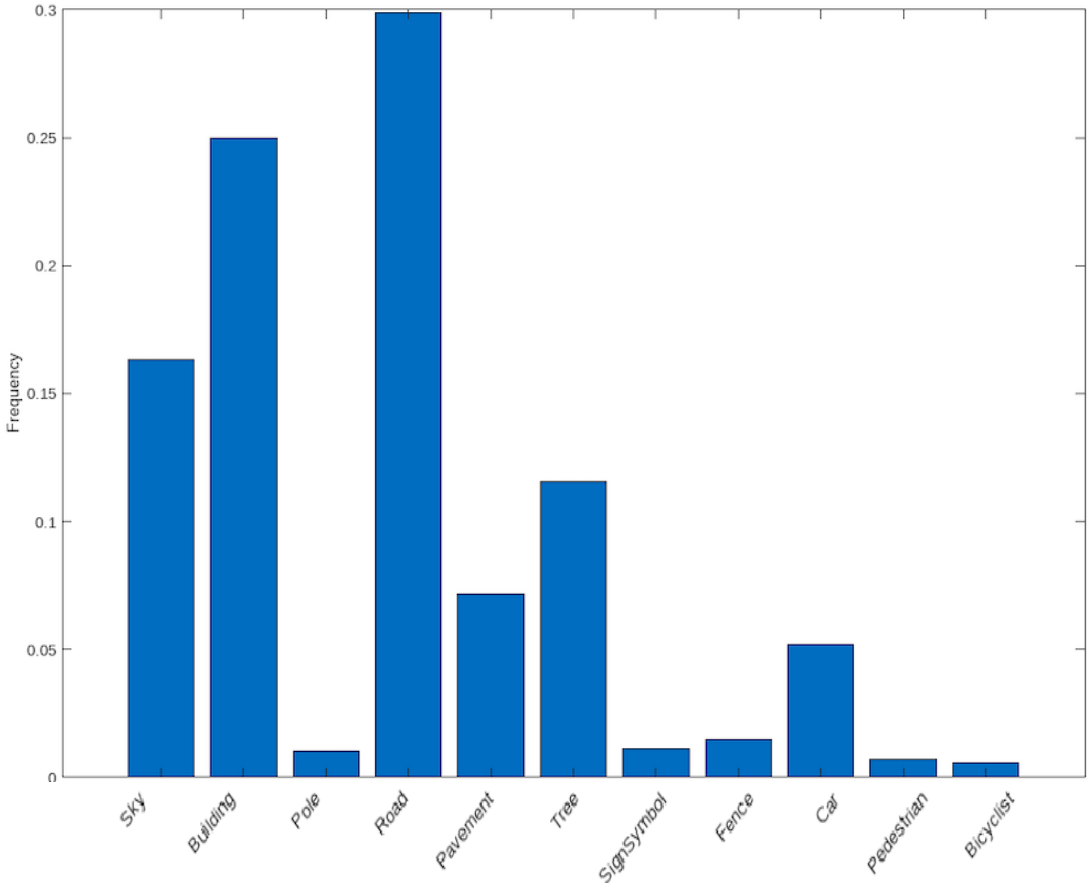


Figura 3.13 : gráfico de frecuencias de la figura 3.11

El código de MATLAB utilizado para este ejemplo puede verse en el anexo: Código de MATLAB 3.2.2. Identificación de objetos.

3.2.3 Ejemplo 3. Reconocimiento de objetos.

En esta aplicación de las redes neuronales convolucionales se verá como la red es capaz de reconocer objetos aleatorios a través de la webcam del ordenador a través de

una red convolucional denominada GoogleNet. GoogleNet es una red convolucional profunda formada por 22 capas diseñada por la empresa Google que se presentó a los usuarios en el año 2014 de la mano del ingeniero Christian Szegedy y sus colaboradores[18]. Para acceder a GoogleNet en MATLAB, se pude utilizar la red pre-entrenada GoogleNet que se encuentra en el paquete de Deep Learning Toolbox.

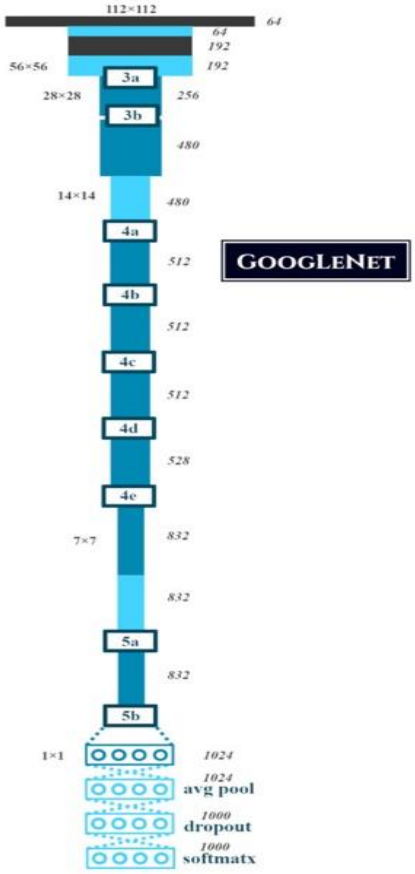


Figura 3.14[18]: arquitectura de Googlenet

GoogleNet se compone de tres secciones diferentes:

- La sección convolucional la cual se compone de dos capas convolucionales como las vistas anteriormente seguidas cada una de ellas por una capa de max pooling.
- En la segunda sección se colocan sucesivamente tres bloques de dos (3a,3b), cinco (4a,4b,4c,4d,4e) y dos (5ª,5b) módulos de incepción respectivamente. Los **módulos de incepción** contienen capas convolucionales de filtros (1×1) todas ellas con la función de activación ReLU. Estos tres bloques están separados entre sí por una capa de max pooling.
- En la sección de salida se suceden una capa de average pooling, otra de **apagado o dropout**, la cual desactiva un número de neuronas de forma aleatoria y por último, una última capa con función SOFTMAX , la cual realiza la clasificación.[18]

Para implementar esta red en el reconocimiento de objetos[V], se colocarán objetos delante de la webcam del ordenador, y la red nos devolverá, con cierta probabilidad qué es el objeto que ha captado a través de la cámara.

En primer lugar, se cargará la cámara seleccionando la del propio ordenador, y se seleccionará la red pre-entrenada con la que se quiere trabajar, en este caso Googlenet.

Para continuar se realizará la fotografía, se acudirá a la red neuronal seleccionada para realizar la clasificación y además se le pedirá que devuelva un histograma con las cinco predicciones principales.

Esto se realizará con dos objetos aleatorios obteniendo así las siguientes salidas:

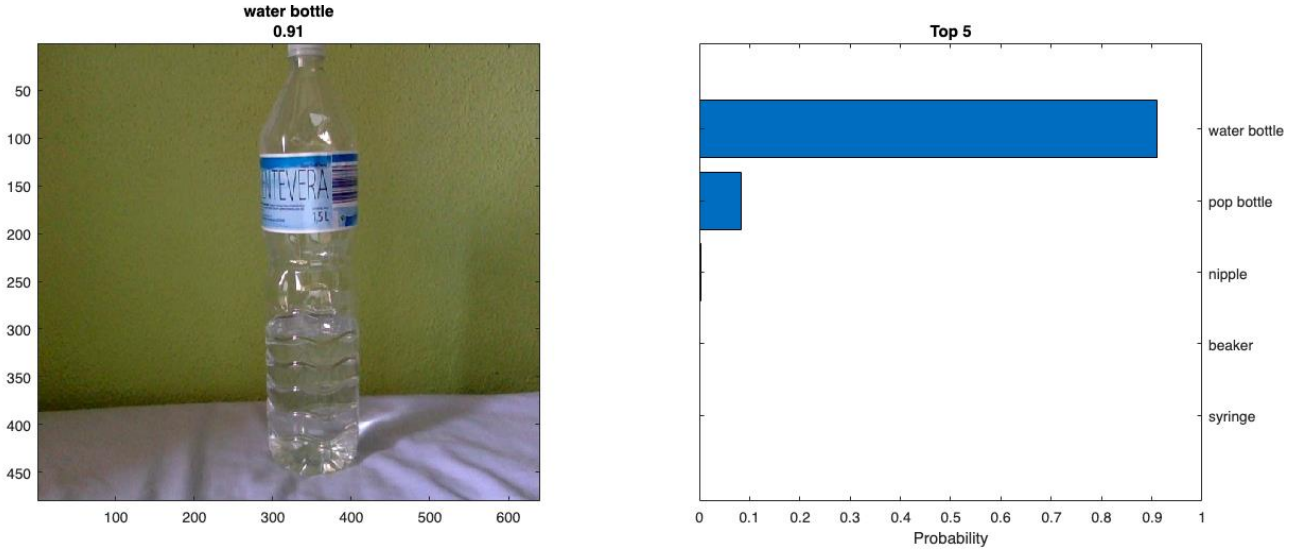


Figura 3.15 : clasificación a través de webcam de botella de agua

En este caso, se colocó una botella de agua, la cual clasifica de manera correcta con un 91% de seguridad. Además a la derecha podemos ver el gráfico de las cinco predicciones más cercanas a la que la red neuronal ha devuelto en la salida junto con sus probabilidades.

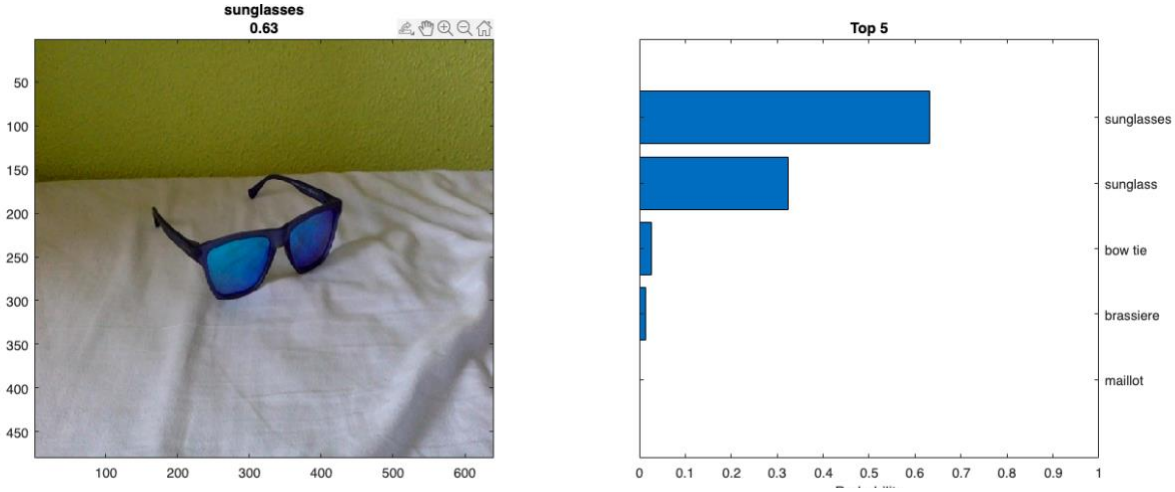


Figura 3.16 : clasificación a través de webcam de gafas de sol

En este caso, se colocó delante de la webcam del ordenador unas gafas de sol, las cuales clasifica de manera correcta con un 63% de seguridad. Además podemos ver el gráfico a la derecha de las cinco predicciones más cercanas a la que la red neuronal ha devuelto en la salida junto con sus probabilidades.

El código de MATLAB utilizado para este ejemplo puede verse en el anexo: Código de MATLAB 3.2.1. Clasificación de imágenes.

CONCLUSIONES

En este trabajo se ha profundizado en los métodos y algoritmos matemáticos que conforman el funcionamiento de las redes neuronales para el procesamiento de imágenes. Así como se han proporcionado numerosos ejemplos para entenderlo tanto teórica como prácticamente.

Además de estudiar todas las matemáticas que hay detrás de las redes neuronales y de las redes neuronales convolucionales, se ha llevado toda esa teoría a la práctica en el último apartado. Se ha conseguido programar y entrenar una red neuronal convolucional desde cero para la clasificación de imágenes, utilizando únicamente elementos explicados a lo largo del trabajo.

También se han visto otras aplicaciones del procesamiento de imágenes con redes neuronales convolucionales, como son la identificación o el reconocimiento de objetos, a través de redes convolucionales pre-entrenadas que ofrece MATLAB para el aprovechamiento de sus usuarios.

En conclusión, las redes neuronales convolucionales son una herramienta muy poderosa para resolver problemas relacionados con el procesamiento de imágenes y visión artificial. A pesar de que este trabajo se haya centrado en esta aplicación, cabe destacar otras aplicaciones de las redes neuronales convolucionales para futuras líneas de investigación:

- Reconocimiento de patrones: Las CNN se utilizan en problemas de reconocimiento de patrones, como la identificación de escritura a mano, la identificación de dígitos y la identificación de lenguaje.
- Generación de imágenes: Las CNN se utilizan en la generación de imágenes, como la creación de imágenes a partir de descripciones de texto o la creación de imágenes de alta resolución a partir de imágenes de baja resolución.
- Medicina: Las CNN se utilizan en la diagnóstica de enfermedades a partir de imágenes médicas, como tomografías y radiografías, así como en la identificación de patrones en datos médicos.
- Juegos: Las CNN se utilizan en el aprendizaje de juegos, como el aprendizaje de estrategias de juego y la generación de personajes y paisajes en juegos.

REFERENCIAS

[1] Caicedo Bravo, E. F., & Lopez Sotelo, J. A. (2009). *Una aproximación práctica a las redes neuronales artificiales.* Universidad del valle Programa editorial.

[2] R.Llinás, R. (2003). The contribution of Santiago Ramon y Cajal to functional neuroscience. *nature reviews neuroscience*, 77-80.

[3] McCulloch, W., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of mathematical biophisysics, 5*.

[4] Ramirez, F. (22 de Julio de 2018). *archive.org.* Obtenido de Wayback Machine: https://web.archive.org/web/20180722124753/https:/data-speaks.luca-d3.com/2018/07/historia-de-la-ia-frank-rosenblatt-y-el.html

[5] Minsky, M., & Papert, S. (1972 (2nd edition with corrections, first edition 1969)). *Perceptrons: An Introduction to Computational Geometry.* Cambridge MA: The MIT Press.

[6] McCorduck, P. (2004). *Machines Who Think (2nd ed.).* (A. Peters, Ed.) Natick: CRC Press.

[7] Fukushima, K. (1975). *Cognitron: A self-organizing multilayered neural network.* Biological Cybernetics.

[8] Fukushima, K. (1980). *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.* (Vol. 4). Biological Cybernetics.

[9] Torrades, S., & Perez-Sust, P. (2008). Sistema visual. La percepción del mundo que nos rodea. *Offarm, 6*(27), 98-99.

[10] Torrades, S., & Perez-Sust, P. (2008). Sistema visual. La percepción del mundo que nos rodea. *Offarm, 6*(27), 101.

[11] Aguilar Medina, D. (2008). Redes neuronales. *RITS*(1), 88-92.

[12] Garcia-Olalla Olivera, O. (16 de septiembre de 2019). *xeridia.com.* Obtenido de https://www.xeridia.com/blog/redes-neuronales-artificiales-que-son-y-como-se-entrenan-parte-i

[13] Hubel, D., & Wiesel, T. (1962). Receptive fields, binocular interaction and functional architecture in the cat´s visual cortex. *The Journal of physiology, 160*(1), 106.

[14] Berzal , F. (2018). *Redes Neuronales & Deep Learning.* Granada. 599-630

[15] Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *JMLR Proceedings, 9*, 249-256.

[16] Berzal, F. (2018). *Redes Neuronales & Deep Learning.* Granada. 197-199

[17] *datascientest.com.* (s.f.). Obtenido de DataScientest: https://datascientest.com/es/convolutional-neural-network-es

[18] Rodriguez Abril, R. (s.f.). *lamaquinaoraculo.com.* Obtenido de La Máquina Oráculo: https://lamaquinaoraculo.com/computacion/googlenet/

BIBLIOGRAFÍA

Aguilar Medina, D. (2008). Redes neuronales. *RITS*(1), 88-92.

Berzal , F. (2018). *Redes Neuronales & Deep Learning.* Granada.

Berzal, F. (2018). *Redes Neuronales & Deep Learning.* Granada.

Caicedo Bravo, E. F., & Lopez Sotelo, J. A. (2009). *Una aproximación práctica a las redes neuronales artificiales.* Universidad del valle Programa editorial.

datascientest.com. (s.f.). Obtenido de DataScientest: https://datascientest.com/es/convolutional-neural-network-es

Fukushima, K. (1975). *Cognitron: A self-organizing multilayered neural network.* Biological Cybernetics.

Fukushima, K. (1980). *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.* (Vol. 4). Biological Cybernetics.

Garcia-Olalla Olivera, O. (16 de septiembre de 2019). *xeridia.com.* Obtenido de https://www.xeridia.com/blog/redes-neuronales-artificiales-que-son-y-como-se-entrenan-parte-i

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *JMLR Proceedings, 9*, 249-256.

Hubel, D., & Wiesel, T. (1962). Receptive fields, binocular interaction and functional architecture in the cat´s visual cortex. *The Journal of physiology, 160*(1), 106.

McCorduck, P. (2004). *Machines Who Think (2nd ed.).* (A. Peters, Ed.) Natick: CRC Press.

McCulloch, W., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of mathematical biophisysics, 5*.

Minsky, M., & Papert, S. (1972 (2nd edition with corrections, first edition 1969)). *Perceptrons: An Introduction to Computational Geometry.* Cambridge MA: The MIT Press.

R.Llinás, R. (2003). The contribution of Santiago Ramon y Cajal to functional neuroscience. *nature reviews neuroscience*, 77-80.

Ramirez, F. (22 de Julio de 2018). *archive.org.* Obtenido de Wayback Machine: https://web.archive.org/web/20180722124753/https:/data-speaks.luca-d3.com/2018/07/historia-de-la-ia-frank-rosenblatt-y-el.html

Rodriguez Abril, R. (s.f.). *lamaquinaoraculo.com.* Obtenido de La Máquina Oráculo: https://lamaquinaoraculo.com/computacion/googlenet/

Torrades, S., & Perez-Sust, P. (2008). Sistema visual. La percepción del mundo que nos rodea. *Offarm*, *6*(27), 98-99.

Torrades, S., & Perez-Sust, P. (2008). Sistema visual. La percepción del mundo que nos rodea. *Offarm*, *6*(27), 101.

WEBGRAFÍA

[A] https://bit.ly/3INeyjs

Consultado en noviembre 2022

[B] https://bit.ly/3ZOsSP5

Consultado en noviembre 2022

[C] http://bit.ly/3iBjRYE

Consultado en noviembre 2022

[D] http://bit.ly/3kh3j8W

Consultado en diciembre 2022

[E] http://bit.ly/3kiBYTz

Consultado en diciembre 2022

[F] http://bit.ly/3ZzPnXJ

Consultado en diciembre 2022

[G] http://bit.ly/3XzbR9B

Consultado en diciembre 2022

[H] http://bit.ly/3iCXuSC

Consultado en enero 2023 (Cecbur, CC BY-SA 4.0)

[I] http://bit.ly/3kbGPWJ

Consultado en enero 2023

[J] http://bit.ly/3CPIsjs

Consultado en enero 2023 (Cecbur, CC BY-SA 4.0)

[K] https://bit.ly/3CLZgHS

Consultado en enero 2023 (Cecbur, CC BY-SA 4.0)

[L] http://bit.ly/3iHKCL5

Consultado en enero 2023

[M] http://bit.ly/3Wi7FK1

Consultado en diciembre 2022

[N] http://bit.ly/3DcStrh

Consultado en diciembre 2022

[O] http://bit.ly/3E5QTHQ

Consultado en enero 2023

[P] http://bit.ly/3w39ShR

Consultado en enero 2023

[Q] http://bit.ly/3GCN0uy

Consultado en diciembre 2022

[R] https://bit.ly/3iIUm80

Consultado en diciembre 2022

[S] https://bit.ly/3iIONq3

Consultado en diciembre 2022

[T] http://bit.ly/3W7AHw0

Consultado en noviembre 2022

[U] http://bit.ly/3ZEq1Yz

Consultado en noviembre 2022

[V] http://bit.ly/3w5VEwF

Consultado en noviembre 2022

[X] Figuras de creación propia.

ANEXOS

Código de MATLAB 3.2.1. Clasificación de imágenes.

```
clear all
clc
clf

%Cargar base de datos
digitDatasetPath = fullfile(matlabroot,"toolbox","nnet","nndemos",...
    "nndatasets","DigitDataset");
imds = imageDatastore(digitDatasetPath,...
    "IncludeSubfolders",true,"LabelSource","foldernames");

%Mostrar ejemplos de la base de datos
figure;
perm = randperm (10000,20);
for i = 1:20
    subplot(4,5,i)
    imshow(imds.Files{perm(i)});
end

%Calcular el número de imágenes que tiene cada categoría
labelCount = countEachLabel(imds)

%Definir el tamaño de cada imagen
img = readimage(imds,1);
size(img)

%Especificar conjuntos de entrenamiento y validación
numTrainFiles = 750;
[imdsTrain, imdsValidation] = splitEachLabel(imds,numTrainFiles,"randomized");

%Definir la arquitectura de la red neuronal convolucional
layer = [
  imageInputLayer([28 28 1])%tamaño de la imagen

  convolution2dLayer(3,8,"Padding","same")
  batchNormalizationLayer
  reluLayer

  maxPooling2dLayer(2,"Stride",2)

  convolution2dLayer(3,16,"Padding","same")
  batchNormalizationLayer
  reluLayer
```

```
  maxPooling2dLayer(2,"Stride",2)

  convolution2dLayer(3,32,"Padding","same")
  batchNormalizationLayer
  reluLayer

  fullyConnectedLayer(10)
  softmaxLayer
  classificationLayer]

%Especificar el entrenamiento de la red
option = trainingOptions("sgdm",...
  "InitialLearnRate",0.01,...
  "MaxEpochs",4,...
  "Shuffle","every-epoch",...
  "ValidationData",imdsValidation,...
  "ValidationFrequency",30,...
  "Verbose",false,...
  "Plots","training-progress");

%Entrenar a la red
net = trainNetwork(imdsTrain,layer,option)

%% Sacar figuras del conjunto de validación
figure;
I=imds.Files{4};
F=imds.Files{3513};
G=imds.Files{7324};
subplot(1,3,1);
imshow(I);
subplot(1,3,2);
imshow(F);
subplot(1,3,3)
imshow(G);
%% Introducir imágenes para que la red neuronal las clasifique
% Cargar base de datos
digitDatasetPath = fullfile(matlabroot,"toolbox","nnet","nndemos",...
"nndatasets","DigitDataset");
imds = imageDatastore(digitDatasetPath,...
"IncludeSubfolders",true,"LabelSource","foldernames");

% Especificar conjuntos de entrenamiento y validación
numTrainFiles = 750;
[imdsTrain, imdsValidation] = splitEachLabel(imds,numTrainFiles,"randomized");

% Cargar la red neuronal previamente entrenada
load("TrainedNetwork.mat", "net");
```

```
% Cargar la imagen
TestImage = imread("imagen.png");

% Hacer que la imagen tenga tamaño [28 28 1]
grayTestImage = rgb2gray(TestImage);
finalTestImage = imresize(grayTestImage, [28 28]);
imshow(finalTestImage)

% Clasificar la imagen de prueba
predictedClass = classify(net, finalTestImage);

% Mostrar la clase predicha
fprintf("La clase a la que pertenece la imagen introducida es: %s\n", predictedClass);
```

Código de MATLAB 3.2.2. Identificación de objetos.

```
clear all
clc
clf

%%Descargar la red preentrenada de segmentación de imágenes
pretrainedURL = 'https://ssd.mathworks.com/supportfiles/vision/data/deeplabv3plusResnet18CamVid.zip';
pretrainedFolder = fullfile(tempdir,'pretrainedNetwork');
pretrainedNetworkZip = fullfile(pretrainedFolder,'deeplabv3plusResnet18CamVid.zip');
if ~exist(pretrainedNetworkZip,'file')
    mkdir(pretrainedFolder);
    disp('Downloading pretrained network (58 MB)...');
    websave(pretrainedNetworkZip,pretrainedURL);
end

%Cargar la red preentrenada para su futuro uso
unzip(pretrainedNetworkZip, pretrainedFolder)
pretrainedNetwork = fullfile(pretrainedFolder,'deeplabv3plusResnet18CamVid.mat');
data = load(pretrainedNetwork);
net = data.net;

% Descargar el Dataset de imágenes
imageURL = 'http://web4.cs.ucl.ac.uk/staff/g.brostow/MotionSegRecData/files/701_StillsRaw_full.zip';
labelURL = 'http://web4.cs.ucl.ac.uk/staff/g.brostow/MotionSegRecData/data/LabeledApproved_full.zip';

outputFolder = fullfile(tempdir,'CamVid');
labelsZip = fullfile(outputFolder,'labels.zip');
imagesZip = fullfile(outputFolder,'images.zip');

if ~exist(labelsZip, 'file') || ~exist(imagesZip,'file')
    mkdir(outputFolder)
```

```
    disp('Downloading 16 MB CamVid dataset labels...');
    websave(labelsZip, labelURL);
    unzip(labelsZip, fullfile(outputFolder,'labels'));

    disp('Downloading 557 MB CamVid dataset images...');
    websave(imagesZip, imageURL);
    unzip(imagesZip, fullfile(outputFolder,'images'));
end

%Listar las clases que nuestra red identificará en la imagen
classes = string(net.Layers(end).Classes)

%Cargar las imágenes y le pedimos que nos enseñe la 559
imgDir = fullfile(outputFolder,'images','701_StillsRaw_full');
imds = imageDatastore(imgDir);
I = readimage(imds,559);
I = histeq(I);
imshow(I)

%Realizar el mapa a color de la segmentación de imágenes
labelIDs = camvidPixelLabelIDs();
labelDir = fullfile(outputFolder,'labels');
pxds = pixelLabelDatastore(labelDir,classes,labelIDs);
C = readimage(pxds,559);
cmap = camvidColorMap;
B = labeloverlay(I,C,'ColorMap',cmap);
imshow(B)
pixelLabelColorbar(cmap,classes);

%Analizar las estadísticas
tbl = countEachLabel(pxds)
frequency = tbl.PixelCount/sum(tbl.PixelCount);
bar(1:numel(classes),frequency)
xticks(1:numel(classes))
xticklabels(tbl.Name)
xtickangle(45)
ylabel('Frequency')
```

Código de MATLAB 3.2.3 Reconocimiento de objetos.

```
clear all
clc
clf

%Cargar la cámara y seleccionar la red
camera = webcam;
net = googlenet;
```

```
%Clasificar instantáneas de la cámara
inputSize = net.Layers(1).InputSize(1:2)
figure
im = snapshot(camera);
image(im)
im = imresize(im,inputSize);
[label,score] = classify(net,im);
title({char(label),num2str(max(score),2)});

%Mostrar predicciones principales
h = figure;
h.Position(3) = 2*h.Position(3);
ax1 = subplot(1,2,1);
ax2 = subplot(1,2,2);
im = snapshot(camera);
image(ax1,im)
im = imresize(im,inputSize);
[label,score] = classify(net,im);
title(ax1,{char(label),num2str(max(score),2)});
[~,idx] = sort(score,'descend');
idx = idx(5:-1:1);
classes = net.Layers(end).Classes;
classNamesTop = string(classes(idx));
scoreTop = score(idx);
barh(ax2,scoreTop)
xlim(ax2,[0 1])
title(ax2,'Top 5')
xlabel(ax2,'Probability')
yticklabels(ax2,classNamesTop)
ax2.YAxisLocation = 'right';
```