Universidad de Valladolid

Facultad de ciencias

Trabajo fin de grado

Grado en matemáticas

Aprendizaje estadístico en el reconocimiento de señales de tráfico

Autor: Gonzalo Bobillo Rincón
Tutor: Eusebio Arenal Gutiérrez

2024

Índice general

Resumen

Este Trabajo Fin de Grado explora los fundamentos teóricos y la aplicación práctica de métodos matemáticos de clasificación en el campo del reconocimiento de objetos, con énfasis en la detección de señales de tráfico.
Se aborda la formulación matemática del reconocimiento de objetos mediante la representación matricial de imágenes y los principios estadísticos del aprendizaje automático. El estudio examina técnicas de clasificación desde un punto de vista matemático, como el algoritmo de K-Vecinos más Cercanos, la regresión lineal, la regresión logística y las redes neuronales. Se profundiza especialmente en la teoría de redes neuronales y su proceso de aprendizaje mediante retropropagación y métodos de optimización. Además, se analiza un tipo concreto de red neuronal, las redes neuronales convolucionales (CNN). Se pondrá en práctica este conocimiento teórico mediante el entrenamiento de una CNN (YOLOv7) para el reconocimiento de señales de tráfico.
El trabajo concluye con un análisis de los resultados, demostrando la aplicación de conceptos matemáticos avanzados en el aprendizaje profundo y la visión por computadora.

Palabras clave: Reconocimiento de objetos, clasificación, *deep learning*, redes neuronales, CNN.

Abstract

This Thesis explores the theoretical foundations and practical application of mathematical classification methods in the field of object recognition, with an emphasis on traffic sign detection.
The mathematical formulation of object recognition is addressed through the matrix representation of images and the statistical principles of machine learning. The study examines classification techniques from a mathematical perspective, such as the k-Nearest Neighbors algorithm, linear regression, logistic regression, and neural networks. Special attention is given to the theory of neural networks and their learning process through backpropagation and optimization methods. Additionally, a specific type of neural network, convolutional neural networks (CNNs), is analyzed.
This theoretical knowledge is put into practice by training a CNN (YOLOv7) for traffic sign recognition. The thesis concludes with an analysis of the results, demonstrating the application of advanced mathematical concepts in deep learning and computer vision.

Keywords: Object recognition, classification, deep learning, neural networks, CNN.

Introducción

Durante la última década, el *machine learning* se ha convertido en una de las áreas más influyentes de la inteligencia artificial, encontrando aplicaciones en una amplia gama de campos, desde la medicina hasta las finanzas y la ingeniería. Esta disciplina se basa en el desarrollo de algoritmos que permiten a las máquinas aprender de los datos y mejorar su rendimiento en tareas específicas sin que realmente estén planificadas para ello. El objetivo de este trabajo es explorar diferentes técnicas de aprendizaje automático desde un punto de vista matemático, mostrando una comprensión profunda de las bases teóricas que subyacen a estos algoritmos, con una aplicación práctica en el reconocimiento de señales de tráfico.

En primer lugar, se explicará el planteamiento del problema y el marco en el que se desarrolla (capítulo 1). Luego, se abordarán los conceptos que sustentan los métodos de aprendizaje supervisado, incluyendo técnicas como la regresión lineal (2.3.1), logística (2.3.2) o las redes neuronales (capítulo 3). Se detallará cómo se modelan estos problemas matemáticamente, cuáles son las suposiciones subyacentes, y cómo se resuelven utilizando técnicas de optimización y teoría de la probabilidad. Se pondrá énfasis en el problema de clasificación.

A continuación, en el capítulo 4, como parte aplicada de este trabajo, se presenta un modelo específico para el reconocimiento de señales de tráfico en imágenes utilizando redes neuronales convolucionales (*Convolutional Neural Networks*, CNN) y se explica la metodología que se ha llevado a cabo para su implementación en un conjunto de datos real. Este problema es de gran relevancia en el campo de la visión por computador y es fundamental para el desarrollo de sistemas de conducción autónoma. Las redes neuronales convolucionales, que imitan el funcionamiento de la corteza visual del cerebro, han demostrado un rendimiento superior en tareas de clasificación y reconocimiento de imágenes, lo que las hace ideales para esta aplicación.

El desarrollo del modelo de identificación de señales de tráfico se llevará a cabo utilizando los conceptos teóricos presentados en la primera parte del trabajo, destacando la importancia de una comprensión sólida de las bases matemáticas para el diseño de algoritmos de *machine learning* efectivos.

Finalmente, se presentarán los resultados obtenidos (4.5) al aplicar el modelo a un conjunto de datos real, evaluando su precisión y eficiencia, destacando las ventajas y limitaciones del enfoque propuesto.

Capítulo 1

Planteamiento del problema

En los últimos años, el uso de sistemas avanzados de asistencia a la conducción (ADAS, *Advanced Driver Assistance Systems*) se ha generalizado en los vehículos modernos. Simultáneamente, las investigaciones en el ámbito de la conducción autónoma han avanzado notablemente, acercando cada vez más la posibilidad de vehículos que no requieren intervención humana. Ambos desarrollos comparten la necesidad esencial de identificar correctamente las señales de tráfico, lo cual es fundamental para garantizar una conducción segura.

Para abordar este desafío perteneciente al campo de la visión por computador, se han desarrollado diversas técnicas a lo largo del tiempo. Este problema puede parecer trivial, ya que el reconocimiento de objetos es una tarea inherentemente fácil, incluso para un niño. El cerebro humano está adaptado biológicamente para realizar esta tarea de forma rápida y eficaz. En contraste, desarrollar sistemas artificiales que puedan realizar esta tarea con la misma precisión y velocidad presenta un desafío considerable. Las máquinas deben lidiar con la variabilidad en las condiciones de iluminación, ángulos de visión y oclusiones de los objetos, y no poseen la misma habilidad innata para generalizar a partir de experiencias limitadas. Esto requiere el diseño de algoritmos complejos y el procesamiento de grandes volúmenes de datos para lograr resultados comparables.

1.1. Sistemas de reconocimiento de objetos en imágenes

Un sistema de reconocimiento de objetos en imágenes toma como entrada una imagen y devuelve como salida los objetos de interés y su localización.

El proceso que llevan a cabo estos sistemas se puede descomponer en tres partes principales. Veámoslo en el caso concreto de un sistema de reconocimiento de señales de tráfico, que es el tema que nos atañe en este trabajo.

1. **Localización de la región de interés**: Consiste en identificar una zona de la imagen donde puede haber una señal de tráfico. La opción más simple es realizar un barrido sobre la imagen. Se elige un tamaño de ventana y se desplaza por toda la imagen de forma sistemática, analizando cada sección o bloque de píxeles en busca de patrones que puedan indicar la presencia de un objeto. Este

método consiste en una búsqueda por fuerza bruta, por lo que tiene un alto coste computacional, ya que requiere evaluar un gran número de ventanas. Una opción alternativa es generar una serie de máscaras binarias para separar los objetos de interés del fondo. Se puede utilizar la información de color de la imagen, ya que las señales de tráfico se caracterizan por poseer colores que resaltan frente al fondo (rojo, azul, amarillo, blanco, negro) y en unas proporciones concretas. Como resultado, las regiones de interés se determinan como componentes conectados, algunos de los cuales serán señales de tráfico.

2. **Verificación de la presencia de una señal de tráfico**: Se deben filtrar aquellas regiones de interés que contengan una señal de tráfico (del tipo que sea) de aquellas que contienen otro tipo de objetos. Una posibilidad es utilizar el conocimiento de su forma y sus bordes, por ejemplo, reconociendo triángulos equiláteros, círculos, etc.

3. **Categorización del tipo de señal de tráfico**: Una vez que se ha confirmado la presencia de la señal, el paso final es el reconocimiento, utilizando una base de datos de todos los posibles modelos de señales de tráfico. Se pueden utilizar métodos de clasificación que van desde la simple coincidencia de plantillas hasta sofisticados algoritmos de aprendizaje automático como las redes neuronales para lograr un reconocimiento robusto.

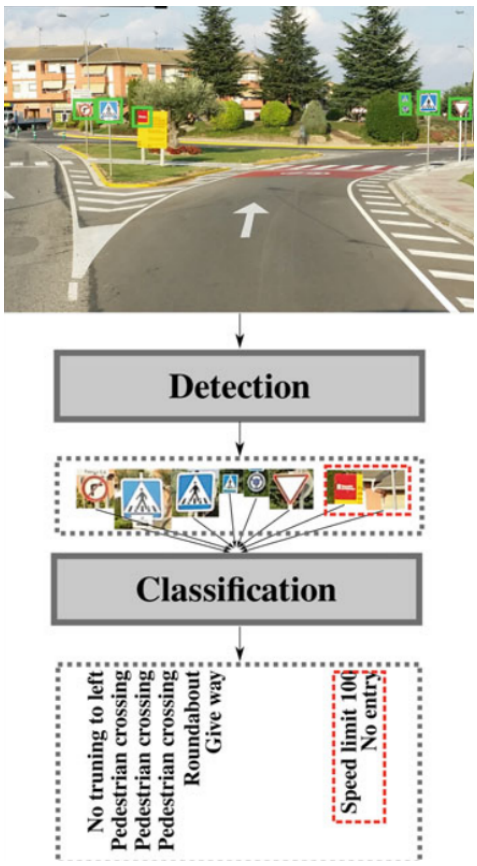

Figura 1.1: Ejemplo del proceso de reconocimiento de señales de tráfico. Usualmente se agrupan los pasos 1 y 2, denominándolo «detección». Fuente: [9].

Para cada una de estas fases existen múltiples métodos distintos, variando en complejidad, rapidez y eficiencia.

Desde una perspectiva matemática, los pasos 2 y 3 se corresponden a un problema de clasificación. La verificación es una clasificación binaria (señal contra no-señal) mientras que la categorización es una clasificación multiclase. En el capítulo 2 se profundizará sobre los métodos de clasificación.

El primer paso tiene menos contenido matemáticamente interesante, así que no se desarrollará en este trabajo. Se puede consultar en profundidad la información relativa a este paso en [15].

1.2. Representación de la imagen

Para poder aplicar un sistema de detección de objetos, se necesita primero representar digitalmente una imagen, ya que es el objeto sobre el que se aplicarán los pasos mencionados previamente.
Una imagen digital está compuesta por píxeles, donde cada píxel solo puede tomar un color. En blanco y negro, se puede representar como una matriz de tamaño w x h, donde w y h son el ancho y el alto en píxeles. Esta matriz toma valores enteros entre 0 y 255 según el color del píxel, donde 0 representa al blanco, 255 al negro y entre medias la escala de grises.

En el caso de las imágenes a color, una representación habitual es la RGB, siglas correspondientes a *Red, Green, Blue*. El color de cada píxel está definido por las intensidades de estos tres colores primarios. Por tanto, se tendrían tres matrices de dimensión w x h, correspondiente a las intensidades del rojo, verde y azul.

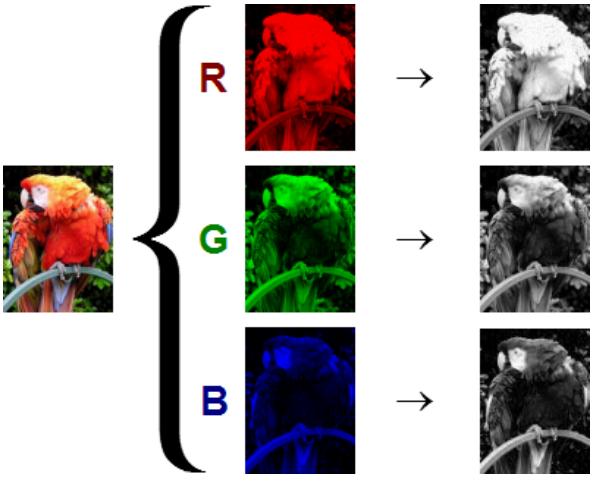

Figura 1.2: Representación RGB de una imagen.

Así, podemos definir una imagen I como un arreglo de dimensión $w \times h \times 3$ que toma valores en $[0, 255]$.

$$I \in \mathcal{M}_{w \times h \times 3}([0, 255])$$

Por tanto, el problema que trataremos es, dada una representación RGB de una imagen a color, localizar las n posibles señales de tráfico y clasificarlas de acuerdo a su

significado.

Para abordar este problema, existen múltiples métodos de clasificación de imágenes y de detección de objetos, siendo aquellos provenientes del aprendizaje automático lo más usados en la actualidad.

1.3. Aprendizaje automático

El aprendizaje automático, conocido popularmente como *machine learning*, es una rama de la Inteligencia Artificial que se encarga de construir modelos predictivos que se fundamentan en la teoría de la inferencia estadística y en la optimización para captar patrones en los datos y realizar predicciones sobre datos futuros.

La principal motivación detrás de los modelos de aprendizaje automático frente a los modelos tradicionales reside en la captura automática de patrones complejos que serían difíciles de definir mediante reglas explícitas. A diferencia de los enfoques tradicionales, que dependen de características diseñadas manualmente y suposiciones específicas sobre los datos, los modelos de *machine learning* pueden adaptarse a muchos problemas sin la necesidad de conocimiento experto y mejorar su rendimiento a medida que se les alimenta con más datos. Esto los hace más flexibles y escalables, especialmente en tareas donde los patrones pueden ser muy complejos.

Pongamos un ejemplo para ilustrar el poder del machine learning frente a los modelos clásicos. En el ámbito de la detección de fraudes en tarjetas de crédito, los métodos tradicionales se basaban en sistemas de reglas predefinidas. Estos sistemas eran diseñados por expertos para identificar patrones de fraude mediante reglas fijas, como detectar transacciones en ubicaciones geográficas inusuales o compras de grandes cantidades en poco tiempo. Aunque útiles en algunos casos, estos métodos tenían limitaciones significativas: eran rígidos y no podían adaptarse a nuevos patrones de fraude.

A mediados de la década de 1990, el uso de *machine learning* revolucionó este área. Se comenzaron a usar algoritmos como las redes neuronales y los árboles de decisión, que aprenden de los datos históricos de transacciones y fraudes, permitiendo una detección más dinámica y precisa. Estos modelos no dependen de reglas fijas y se adaptan continuamente a nuevos patrones emergentes de fraude. Como resultado, los sistemas basados en aprendizaje automático lograron una mayor precisión en la detección de fraudes, reduciendo tanto falsos positivos como falsos negativos, y mejorando significativamente la capacidad para identificar comportamientos fraudulentos complejos, según muestra un estudio de Ghosh de 1994 [7] mediante el uso de redes neuronales.

En resumen, mientras que los métodos tradicionales ofrecían un enfoque estático, el aprendizaje estadístico ha demostrado ser superior al proporcionar una detección más adaptativa y precisa, encontrando patrones complejos en los datos que serían difíciles de reconocer para un humano.

1.3.1. Tipos de aprendizaje estadístico

Los problemas de *machine learning* se pueden dividir en aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.

- En el aprendizaje **supervisado** tenemos un conjunto de datos $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ también llamado conjunto de entrenamiento, donde el objetivo es, dado un nuevo vector x, predecir su salida asociada y.

- En el aprendizaje **no supervisado** solo tenemos un conjunto de vectores $D = \{x_1, x_2, \ldots, x_n\}$ sin sus salidas asociadas. El objetivo es analizar y obtener información sobre la estructura de los datos, por ejemplo, agrupándolos en *clusters*.

- El aprendizaje **por refuerzo** se utiliza en entornos dinámicos donde una serie de acciones conllevan a una recompensa o un castigo, por ejemplo, un sistema que está aprendiendo a conducir un automóvil. El sistema comienza a conducir y varios segundos después choca con un obstáculo. Sin embargo, no hay información que cuantifique cómo de buena ha sido cada acción dentro de la serie. Simplemente, el sistema es castigado porque chocó contra el obstáculo. Ahora el sistema debe determinar qué acciones no fueron correctas y modificar su conducta en base a esa experiencia. Cuando el coche no choque, se recompensarán esas acciones para reforzar la conducta positiva.

En este trabajo, nos centraremos en el aprendizaje supervisado, ya que queremos predecir salidas a partir de imágenes. El aprendizaje supervisado se puede dividir en dos tipos de problemas, clasificación o regresión.

La principal diferencia es que en los problemas de **regresión**, la salida es cuantitativa (por ejemplo, el conjunto de los números reales). Mientras que en los problemas de **clasificación**, la salida es categórica, o sea, son etiquetas (por ejemplo, el conjunto $\{$*manzanas*, *peras*, *plátanos*$\}$).

Sin embargo, la idea principal detrás de la resolución de ambos problemas, pasa por encontrar una función que minimice el error en las predicciones. Este error se mide de forma natural en la regresión, donde se pueden aplicar distancias en $\mathbb{R}$, pero se debe definir más cuidadosamente en el problema de clasificación: ¿qué distancia definimos en el conjunto $\{$*manzanas*, *peras*, *plátanos*$\}$? Estas cuestiones se abordarán en el capítulo 2.

Capítulo 2

Métodos matemáticos de clasificación

2.1. Clasificación y regresión

Definamos el problema de regresión matemáticamente.

Sean X e Y variables aleatorias. La variable X representa las características de entrada y toma valores en un conjunto $\mathcal{X}$ denominado conjunto de características, mientras que Y representa las etiquetas o salidas correspondientes, tomando valores en un conjunto no discreto $\mathcal{Y}$ generalmente representado por $\mathbb{R}$. Un conjunto de entrenamiento $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ consiste en n muestras independientes e idénticamente distribuidas (i.i.d.) del par (X, Y), donde cada $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ proviene de la distribución conjunta $P(X, Y)$.

El objetivo general es encontrar la relación entre las variables X e Y a partir de las muestras del conjunto D, para expresarlo de forma general como:

$$Y = f(X) + \epsilon$$

donde $f : \mathcal{X} \to \mathcal{Y}$ es una función desconocida y ϵ representa un término de error aleatorio con esperanza nula.

En el contexto de la clasificación, $\mathcal{Y}$ es un conjunto finito de etiquetas, típicamente $\mathcal{Y} = \{1, 2, \ldots, K\}$ para un problema de K clases.

La clasificación se puede reformular como un caso concreto de regresión si en lugar de predecir directamente una etiqueta de clase, modelamos las probabilidades $p_k(x) = P(Y = k|X = x)$ de que la muestra $x \in \mathcal{X}$ pertenezca a una clase k, teniendo en cuenta que $\sum_{k=1}^{K} p_k = 1$. Entonces, la clasificación se transforma en un problema de regresión en el que predices un función continua real que representa la probabilidad para cada clase.

Después de obtener las probabilidades, la clase final de una muestra x se asigna tomando la que tiene la probabilidad máxima: $\arg\max_k p_k(x)$.

Por tanto,

- la regresión trata de encontrar una función $f : \mathcal{X} \to \mathbb{R}$ que se ajuste a los datos.
- la clasificación busca una función $g : \mathcal{X} \to \mathbb{R}^K$ que modele la distribución de probabilidades condicionadas para cada clase, para luego aplicar la composición que da lugar al clasificador $h(x) = \arg\max_k \circ g(x)$.

El objetivo del problema de clasificación es encontrar un clasificador $h : \mathcal{X} \to \mathcal{Y}$ que modele lo mejor posible la relación de las variables X e Y. Es decir, dado $x \in \mathcal{X}$, el valor $\hat{y} = h(x)$ debería corresponder con la verdadera etiqueta asociada a x, que llamaremos y.
Para poder elegir un buen clasificador h, debemos cuantificar el error de las predicciones. Esto se logra mediante una **función de pérdida** $\mathcal{L}(\hat{y}, y) : \mathcal{Y} \times \mathcal{Y} \longrightarrow \mathbb{R}^+$ que mide la distancia entre la predicción $\hat{y}$ y la salida esperada y. La función de pérdida más simple y natural es la función de **pérdida 0/1**. Se define como

$$\mathcal{L}_{0/1}(\hat{y}, y) = \begin{cases} 1 & \text{si } \hat{y} \neq y \\ 0 & \text{si } \hat{y} = y \end{cases} \tag{2.1}$$

donde cualquier predicción fallida penaliza igual, es decir, la distancia entre todos los elementos de $\mathcal{Y}$ es 1.

La **función riesgo** R asociada a un clasificador h se define como la esperanza de la función de pérdida, siendo (x, y) observaciones provenientes de la distribución conjunta $P(X, Y)$.

$$R(h) = E(\mathcal{L}(\hat{y}, y))$$

Esta función mide el desempeño del clasificador, cuanto más se acerque a 0, mejor realiza su cometido de aproximar la variable Y a partir de X.

El **clasificador de Bayes** proporciona la solución óptima al problema de clasificación. Se define como:

$$h^*(x) = \arg\max_{y \in \mathcal{Y}} P(Y = y | X = x)$$

donde $P(Y = y|X = x)$ es la probabilidad condicional de que la etiqueta sea y dado que se observa x.

Proposición: *En un problema de clasificación multiclase, el clasificador de Bayes h^* minimiza la función de riesgo, usando la función pérdida $\mathcal{L}_{0/1}$.*

Demostración. Sea h un clasificador cualquiera y sea h^* el clasificador de Bayes. Comenzamos desarrollando la probabilidad de que el clasificador h se equivoque

$$P(Y \neq h(x) \mid X = x) = 1 - P(Y = h(x) \mid X = x)$$

$$= 1 - \sum_{k=1}^{K} P(Y = k, h(x) = k \mid X = x).$$

Así, para $x \in \mathcal{X}$ fijo,

$$P(Y \neq h(x) \mid X = x) = 1 - \sum_{k=1}^{K} P(Y = k \mid X = x) I_{\{h(x)=k\}}. \tag{2.2}$$

Aplicando la definición de h^*,

$$P(Y = h^*(x) \mid X = x) = \max_{k \in \{1,\dots,K\}} P(Y = k \mid X = x) \geq \sum_{k=1}^{K} P(Y = k \mid X = x) I_{\{h(x)=k\}}.$$

Y utilizando esta desigualdad en la expresión 2.2:

$$P(h(x) \neq Y \mid X = x) = 1 - \sum_{k=1}^{K} P(Y = k \mid X = x) I_{\{h(x)=k\}} \geq P(y \neq h^*(x) \mid X = x).$$

Por tanto, la probabilidad de que el clasificador h se equivoque en su predicción es mayor o igual a la probabilidad de que se equivoque el clasificador de Bayes h^*. Con la función de pérdida $\mathcal{L}_{0/1}$, la función de riesgo se puede escribir como

$$R(h) = E(\mathcal{L}(h(x), y)) = P(Y \neq h(x) \mid X = x)$$

Por consiguiente,

$$R(h^*) \leq R(h) \quad \forall h$$

□

En la práctica, las probabilidades condicionales $P(Y = y|X = x)$ son desconocidas, lo que imposibilita la construcción explícita del clasificador de Bayes. Sin embargo, podemos realizar una aproximación a partir de estimaciones de estas probabilidades $P(Y = y|X = x)$ que deben ser calculadas a partir de los datos del conjunto finito de entrenamiento $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$,.

La estimación de $P(Y|X)$ puede hacerse de varias maneras:

- Estimación directa: Se modela $P(Y = y|X = x)$ directamente, es decir, asignando la probabilidad de que una observación x tenga una etiqueta y según los datos que disponemos en el conjunto de entrenamiento (por ejemplo, mediante el método de la regresión logística (sección 2.3.2) o las redes neuronales (sección 3.2).

- Enfoque generativo: Se estima $P(X = x|Y = y)$ asumiendo una distribución de las características X sobre Y (por ejemplo, una distribución normal) y también se estima la probabilidad marginal $P(Y = y)$ para luego aplicar el teorema de Bayes:

  $$P(Y = y|X = x) = \frac{P(X = x|Y = y)P(Y = y)}{\sum_{y' \in \mathcal{Y}} P(X = x|Y = y')P(Y = y')}$$

  Este enfoque es usado en modelos generativos, como el clasificador de Bayes ingenuo y los modelos basados en distribuciones gaussianas.

El clasificador de Bayes, aunque no resulta útil en la práctica, proporciona un marco teórico fundamental para comprender y desarrollar algoritmos de clasificación, y sirve como referencia para evaluar el rendimiento de otros clasificadores.

Tampoco podemos construir la función riesgo R ya que implica conocer la distribución exacta $P(X, Y)$, por ello, tenemos que emplear la conocida estimación de la esperanza $E(X) = \frac{1}{n}\sum_i^n x_i$. Aplicando este estimador, obtenemos la función **riesgo empírico**:

$$R_n(h) = \sum_{i=1}^{n} \frac{1}{n} \mathcal{L}(\hat{y}_i, y_i)$$

donde $\hat{y}_i = h(y_i)$ para todo i y $\mathcal{L}$ es la función de pérdida.

2.2. Modelos paramétricos y no paramétricos

Los modelos estadísticos pueden ser categorizados en paramétricos y no paramétricos.

Un método paramétrico asume que los datos siguen una distribución de probabilidad conocida salvo un conjunto finito de parámetros a determinar. La tarea de aprendizaje implica estimar estos parámetros a partir de los datos.

Formalmente, para un método **paramétrico**:

- Existe un espacio de parámetros $\Theta \subset \mathbb{R}^d$
- El clasificador $h_\theta : X \to Y$ está completamente determinado por $\theta \in \Theta$.
- El aprendizaje consiste en estimar θ^* que minimiza el riesgo empírico para un conjunto de entrenamiento: $\theta^* = \arg\min_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^n \mathcal{L}(h_\theta(x_i), y_i)$

Sin embargo, en un método **no paramétrico**, el clasificador no es conocida salvo un conjunto finito de parámetros. En la práctica, estimar esta función equivale a estimar infinitos parámetros.

A continuación se estudiará como ejemplo el modelo no paramétrico K-Vecinos más Cercanos. El resto de métodos desarrollados en este trabajo son paramétricos: regresión lineal, regresión logística y redes neuronales.

2.2.1. K-Vecinos más Cercanos

El método K-Vecinos más Cercanos (*K-Nearest Neighbors* o *KNN*) es un ejemplo de modelo no paramétrico. *KNN* sirve tanto para clasificación como para regresión. En el caso de la clasificación, este método se considera una aproximación empírica al clasificador de Bayes. En lugar de estimar las probabilidades posteriores de forma directa, *KNN* utiliza las frecuencias observadas en el «vecindario» de una muestra para aproximar estas probabilidades.

Sea $(\mathcal{X}, d)$ un espacio métrico, donde $\mathcal{X}$ es el espacio de características y $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$ es una función de distancia. Sea $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ el conjunto de entrenamiento, donde $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$ (espacio de etiquetas). Para un nuevo punto $x \in X$, denotamos $N_K(x)$ al conjunto de los K puntos más cercanos a x (vecinos) en el conjunto D.
La predicción se realiza de la siguiente manera:

a) Para clasificación, mediante la moda:

$$f_K(x) = \text{moda}\{y_i : (x_i, y_i) \in N_K(x)\}$$

Si hay empate en el valor de la moda, se debe aplicar alguna estrategia complementaria, como reducir el valor de K o decantarse por la clase del vecino más cercano.

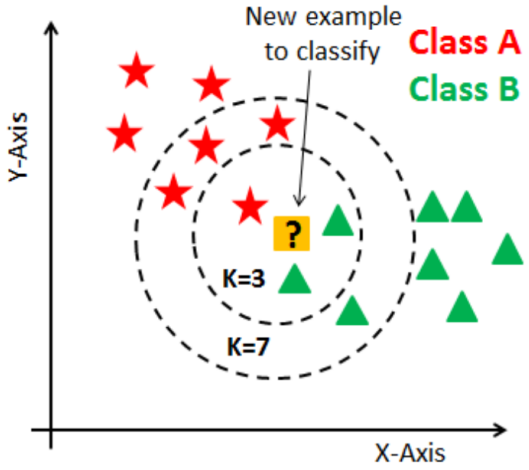

Figura 2.1: El ejemplo a clasificar depende de las clases de sus vecinos más cercanos. Si tenemos en cuenta 3 vecinos, será asignado a la clase B. En cambio, si elegimos 7 vecinos, será asignado a la clase A. Fuente: página web «Introduction to k-Nearest Neighbors (kNN) Algorithm. Medium».

b) Para regresión, mediante la media aritmética:

$$f_K(x) = \frac{1}{K} \sum_{(x_i, y_i) \in N_K(x)} y_i$$

Se puede mejorar *KNN* para que tenga en cuenta la distancia a la que se encuentran los vecinos. Así, si un vecino está muy cerca, tiene más importancia que un vecino lejano. Esto se consigue mediante el uso de pesos w_i que dependen de la distancia. En el caso de la regresión:

$$f_K(x) = \frac{\sum_{(x_i, y_i) \in N_K(x)} w_i y_i}{\sum_{(x_i, y_i) \in N_K(x)} w_i}$$

Por ejemplo, definiendo los pesos

$$w_i = \frac{1}{d(x, x_i)}$$

De esta forma, los pesos de los vecinos que se encuentran más cerca de x son mayores.

En el caso de la clasificación, en lugar de una votación simple, se realiza una votación ponderada donde el peso de cada vecino se suma al total de su respectiva clase:

$$f_K(x) = \arg\max_{y \in \mathcal{Y}} \sum_{i=1}^{k} w_i \cdot 1_{\{y_i = y\}}$$

donde 1_A representa la función indicatriz de un conjunto A.

Este modelo es de los más simples dentro del aprendizaje supervisado. Sin embargo, posee algunas desventajas, como la pérdida de eficacia en espacios de alta dimensión, conocido como la **maldición de la dimensionalidad**, término acuñado por Bellman [3] en 1961. Esto es debido a que al aumentar la dimensión del espacio, la diferencia entre la distancia al punto más cercano y al más lejano tiende a cero para una distribución de datos dada. Una manera de ilustrar este fenómeno es comparar el volumen de una hiperesfera de radio r y un hipercubo de radio $2r$. El volumen de la hiperesfera es $\frac{2r^d \pi^{d/2}}{d\Gamma(d/2)}$, donde Γ es la función gamma y d es la dimensión. Por el otro lado, el volumen del hipercubo es $(2r)^d$. Por tanto,

$$\frac{V_{\text{hiperesfera}}}{V_{\text{hipercubo}}} = \frac{\pi^{d/2}}{d2^{d-1}\Gamma(d/2)} \underset{d\to\infty}{\longrightarrow} 0$$

A medida que aumenta la dimensión d del espacio, el volumen de la hiperesfera se convierte en insignificante en relación con el del hipercubo.

2.3. Clasificación binaria

Empecemos suponiendo que la variable de respuesta es binaria, es decir, $\mathcal{Y}$ está compuesto únicamente por dos valores distintos. Comúnmente se representan con los valores 1 y -1.

2.3.1. Clasificador lineal

El clasificador lineal es un método paramétrico que depende de una función lineal para separar el espacio en dos regiones. Se deben encontrar los parámetros adecuados para minimizar el error de clasificación.

Sea $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^p$. Denotando el elemento i-ésimo de $\boldsymbol{x}$ como x_i, definamos la función lineal f

$$f(x) = b + w_1 x_1 + w_2 x_2 + \ldots + w_p x_p$$

Donde b y w_i son parámetros reales denominados sesgo y pesos, respectivamente.

Denotando de ahora en adelante

$$\boldsymbol{x} = [1, x_1, x_2, \ldots, x_p]$$

y

$$\boldsymbol{w} = [b, w_1, w_2, \ldots, w_p]$$

así podemos escribir de forma más compacta, en notación vectorial,

$$f(x) = \boldsymbol{w}\boldsymbol{x}^T.$$

La ecuación $f(\boldsymbol{x}) = 0$ representa un hiperplano en un espacio de p dimensiones. La función de clasificación h en esta clasificación binaria consiste en asignar una categoría a los puntos que se encuentran a cada lado del hiperplano. Matemáticamente, se representa como

$$h(x) = \text{sign}(f(x)) = \begin{cases} -1 & \text{si } f(x) < 0 \\ \text{indefinido} & \text{si } f(x) = 0 \\ 1 & \text{si } f(x) > 0 \end{cases}$$

A partir de ahora llamaremos a f la**función de decisión**, ya que el clasificador h asigna las clases en función del valor de f. Los puntos que se encuentran sobre el hiperplano no se asignan a ninguna de las dos clases. Estos puntos donde se anula la función de decisión forman la **frontera de decisión**. En la práctica, cuando un punto a clasificar pertenece al hiperplano, se puede asignar arbitrariamente a cualquiera de las dos clases.

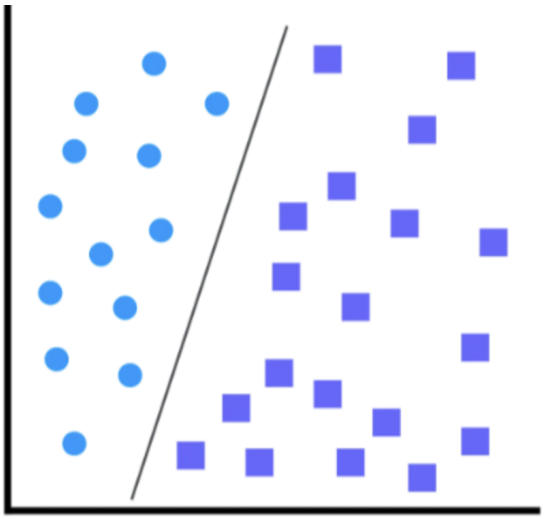

Figura 2.2: Clasificación binaria lineal en $\mathbb{R}^2$. El hiperplano, en este caso una recta, separa el espacio en dos regiones correspondientes a cada etiqueta.

¿Cómo se eligen los parámetros más adecuados para la función de f? Queremos que nuestro clasificador funcione lo mejor posible, es decir, dado un nuevo punto, que sea asignado a la clase que verdaderamente le corresponda. Por tanto, el ajuste de los parámetros depende de los únicos datos que conocemos: el conjunto de entrenamiento $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$. El objetivo es minimizar el error al clasificar los elementos de D. Este error es medido por la **función de pérdida**.

Función de pérdida 0/1

Hemos definido la función de pérdida 0/1 de forma general en la ecuación 2.1. Veámoslo junto a su riesgo empírico en el caso concreto del clasificador lineal.

$$\mathcal{L}_{0/1}(\boldsymbol{w}\boldsymbol{x}^T, y) = \begin{cases} 1 & \text{si } (\boldsymbol{w}\boldsymbol{x}^T) \cdot y_i < 0 \\ 0 & \text{en caso contrario} \end{cases}$$

$$R_n^{0/1}(h) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}_{0/1}(\boldsymbol{w}\boldsymbol{x}_{\boldsymbol{i}}^T, y_i)$$

$R_n^{0/1}(h)$ simplemente devuelve el cardinal del conjunto de puntos mal clasificados según el clasificador lineal h definido por los parámetros $\boldsymbol{w}$. Esta función también se puede expresar como $R_n^{0/1}(\boldsymbol{w})$, ya que el clasificador lineal está definido por el conjunto de parámetros $\boldsymbol{w}$.

Esta función, aunque es intuitiva, en la práctica no sirve, ya que no es diferenciable y esto es necesario para aplicar el método de optimización del descenso del gradiente, empleado para que encuentre el mínimo de la función riesgo. $R_n^{0/1}$ no es continua en $\mathbf{0}$ y además su gradiente es 0 en el resto del dominio.

Función de pérdida cuadrática

La función de pérdida cuadrática, también conocida como error cuadrático medio (*MSE*, *Mean Squared Loss*), es ampliamente utilizada en problemas de regresión y clasificación. Cuantifica la discrepancia entre las predicciones del modelo y los valores reales, al igual que la función de pérdida 0/1, pero con algunas ventajas que veremos.

Formalmente, para un conjunto de n observaciones, la pérdida cuadrática y la función riesgo asociada se definen como:

$$\mathcal{L}_{\text{MSE}} = (y_i - \hat{y}_i)^2$$

$$R_n^{\text{MSE}}(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Como podemos observar, esta función no se puede aplicar directamente a un clasificador cualitativo. Por tanto, necesitaremos interpretar el problema como si fuera una regresión, haciendo uso de las probabilidades, según se ha explicado en 2.1 .

R_n^{MSE} posee varias propiedades matemáticas deseables:

- No negatividad: $\mathcal{L}_{\text{MSE}} \geq 0$.
- Convexidad: Garantiza la existencia de un mínimo global único.
- Diferenciabilidad: Permite el uso de métodos de optimización basados en gradientes.

El interés de esta función radica en la posibilidad de encontrar el mínimo para ajustar los parámetros $\boldsymbol{w}$. Para ello se usa en la práctica el método del descenso del gradiente.

La derivada parcial de R_n^{MSE} se calcula como:

$$\frac{\partial R_n^{\text{MSE}}}{\partial w_j} = \frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)x_i \qquad \forall j = 1, \ldots, p$$

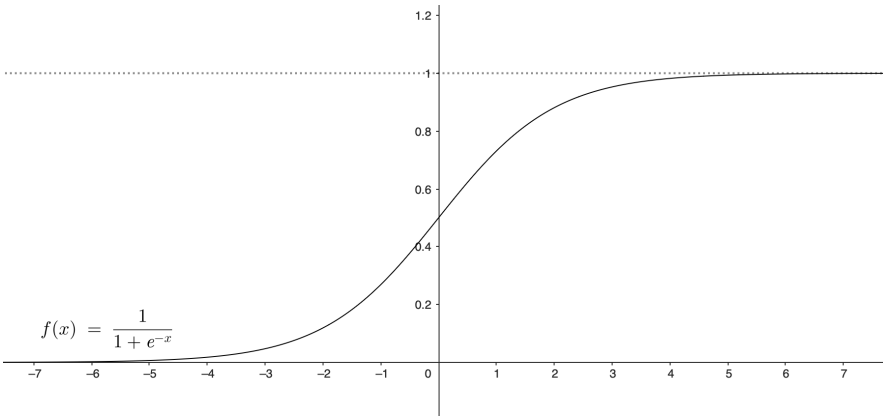

Figura 2.3: Función logística: $\frac{1}{1+e^{-x}}$

$$\frac{\partial R_n^{\text{MSE}}}{\partial b} = \frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)$$

Esta derivada es fundamental para algoritmos de optimización como el descenso de gradiente.

A pesar de sus ventajas, la función de pérdida cuadrática presenta algunas limitaciones:

- Sensibilidad a valores atípicos: El término cuadrático amplifica el efecto de errores grandes.
- Asume implícitamente que los errores siguen una distribución normal.

En el contexto de la clasificación binaria, aunque la pérdida cuadrática puede utilizarse, no es la elección óptima. Para variables de respuesta binarias, funciones como la log-verosimilitud negativa, presentada en la sección siguiente, suelen ser más apropiadas, ya que se alinean mejor con la naturaleza probabilística del problema de clasificación.

La función de pérdida cuadrática sigue siendo, no obstante, una herramienta fundamental en los problemas de regresión, proporcionando una base para el desarrollo de algoritmos más complejos.

2.3.2. Regresión logística

La regresión logística es un método de clasificación binaria que aborda algunas de las limitaciones del clasificador lineal simple. En lugar de clasificar directamente los puntos basándose en el signo de la función lineal $f(x)$, la regresión logística estima la probabilidad de que un punto pertenezca a una clase específica. La idea clave es aplicar una función sigmoide (también conocida como función logística) a la salida del clasificador lineal:

$$p(y = 1|\boldsymbol{x}) = \sigma(f(\boldsymbol{x})) = \frac{1}{1 + e^{-(\boldsymbol{w}\boldsymbol{x}^T)}} \tag{2.3}$$

Donde $\sigma(z) = \frac{1}{1+e^{-z}}$ es la función logística.

El clasificador binario en la regresión logística se define como:

$$h(\boldsymbol{x}) = \begin{cases} 1 & \text{si } p(y=1|\boldsymbol{x}) \geq 0{,}5 \\ -1 & \text{si } p(y=1|\boldsymbol{x}) < 0{,}5 \end{cases}$$

Esta transformación tiene varias ventajas:

- La salida está acotada entre 0 y 1, lo que permite interpretarla como una probabilidad y tratarlo como un problema de regresión.
- La función logística es diferenciable, lo que facilita la optimización de los parámetros.

Para ajustar los parámetros del modelo, se utiliza la función log-verosimilitud negativa. Dado un conjunto de datos $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, el objetivo es maximizar la probabilidad de clasificar correctamente cada elemento. Ya que las muestras del conjunto D son independientes, se aplica la propiedad $p(A, B) = p(A)p(B)$ siendo A, B sucesos independientes.
Se debe maximizar la función de verosimilitud (*likelihood*) asociada al clasificador logístico:

$$l(\boldsymbol{w}) = \prod_{i:y_i=1}^{n} p(y=1|\boldsymbol{x_i}) \prod_{i:y_i=-1}^{n} (1 - p(y=1|\boldsymbol{x_i})).$$

Esta función se acerca a 1 cuando clasifica bien las muestras, mientras que toma el valores cercanos a 0 cuando el conjunto está mal clasificado. Podemos transformar la expresión en la función de pérdida tomando logaritmos para convertir las multiplicaciones en sumas y favorecer el cálculo de sus derivadas. También aplicamos el signo negativo para buscar el mínimo de la función en vez del máximo. Ese punto a buscar será el que minimice el número de muestras mal clasificadas, definiendo el clasificador de parámetros $\boldsymbol{w}$. Por tanto, la función de log-verosimilitud negativa, también denominada entropía cruzada (cross-entropy):

$$E(\boldsymbol{w}) = -\sum_{i=1}^{n} [y_i \log(p(y_i = 1|\boldsymbol{x_i})) + (1 - y_i) \log(1 - p(y_i = 1|\boldsymbol{x_i}))]$$

Esta función de pérdida tiene la ventaja de ser convexa y diferenciable, lo que permite utilizar el descenso de gradiente para encontrar los parámetros óptimos.

La regresión logística es un modelo lineal generalizado con una función de enlace logística. Además de su interpretabilidad y eficiencia computacional, la regresión logística proporciona estimaciones de probabilidad que pueden ser útiles en muchas aplicaciones prácticas donde se requiere una medida de confianza en la clasificación.

2.4. Clasificación multiclase

La clasificación multiclase es una extensión natural de la clasificación binaria, donde el objetivo es asignar a una instancia alguna de las varias clases posibles. Formalmente, en lugar de tener $y \in -1, 1$, ahora tenemos $y \in \{1, 2, ..., K\}$, donde K es el número total de clases.

2.4.1. Combinación de clasificadores binarios

Uno contra el resto

En este enfoque, el problema multiclase se descompone en K problemas de clasificación binaria. Para cada clase k, se entrena un clasificador $h_k(\boldsymbol{x})$ para distinguir entre la clase k y todas las demás clases combinadas. La predicción final $\hat{y}$ se obtiene seleccionando la clase con la puntuación más alta:

$$\hat{y} = \arg \max_{k \in \{1,...,K\}} h_k(\boldsymbol{x})$$

Uno contra uno

Este método implica entrenar $\binom{K}{2} = \frac{K(K-1)}{2}$ clasificadores binarios, uno para cada par de clases. La clase final se determina por votación mayoritaria entre todos estos clasificadores.

2.4.2. Regresión logística multiclase

La regresión logística se puede extender al caso multiclase mediante la generalización conocida como regresión softmax o regresión logística multinomial. Este modelo usa el enfoque *uno contra el resto*.

Construcción de la función *softmax*

Podemos aplicar un logaritmo a ambos lados de la ecuación 2.3 para simplificar la expresión $p(y = 1|\boldsymbol{x})$:

$$\ln p(y = 1|\boldsymbol{x}) = \boldsymbol{w}\boldsymbol{x}^T$$

Sin embargo, como nos encontramos en el contexto de la clasificación multiclase, las probabilidades deben sumar 1, por lo que añadimos un término de normalización Z.

$$\ln p(y = 1|\boldsymbol{x}) = \boldsymbol{w}\boldsymbol{x}^T - \ln(Z)$$

Este es el modelo llamado log-lineal. Con él podemos modelar las probabilidades de cada clases de la siguiente manera. Para cada clase k,

$$\ln p(y = k|\boldsymbol{x}) = \boldsymbol{w}_k\boldsymbol{x}^T - \ln(Z)$$

$$p(y = k|\boldsymbol{x}) = \frac{e^{\boldsymbol{w}_k\boldsymbol{x}^T}}{Z} \qquad \forall k \in \{1, ..., K\}$$

Sabemos que sumando las probabilidades de cada clase debe dar 1.

$$\sum_{k=1}^{K} \frac{e^{\boldsymbol{w}_k\boldsymbol{x}^T}}{Z} = 1$$

$$Z = \sum_{k=1}^{K} e^{\boldsymbol{w}_k\boldsymbol{x}^T}$$

Por tanto, denotando $f_k = \boldsymbol{w}_k \boldsymbol{x}^T$, una vez despejado el factor de normalización, llegamos a la fórmula

$$p(y = k|\boldsymbol{x}) = \frac{e^{f_k(\boldsymbol{x})}}{\sum_{j=1}^{K} e^{f_j(\boldsymbol{x})}}$$

que es siempre positiva y además cumple que

$$\sum_{k=1}^{K} p(y = k|\boldsymbol{x}) = 1,$$

lo que nos permite tratar las salidas de esta función como probabilidades. Esta función es denominada función *softmax*.

Función de pérdida log-verosimilitud negativa multinomial

Para encontrar los mejores parámetros en esta clasificación multiclase, debemos maximizar la probabilidad de que una muestra sea bien clasificada. Dado un conjunto de entrenamiento D, la probabilidad de clasificar correctamente todos los ejemplos para un conjunto de parámetros fijos $\boldsymbol{W} = [\boldsymbol{w}_1, ..., \boldsymbol{w}_K] \in \mathbb{R}^{K \cdot (p+1)}$, se puede escribir como la función de verosimilitud

$$l(\boldsymbol{W}) = \prod_{i=1}^{n} p(y = y_i|\boldsymbol{x}_i)$$

En vez de maximizar esta probabilidad, se puede minimizar la probabilidad logarítmica negativa, tal como se ha explicado en la sección 2.3.2, lo que da lugar a la función de pérdida log-verosimilitud negativa multinomial, aplicando la propiedad de los logaritmos $\log(ab) = \log(a) + \log(b)$ que cambia el producto por un sumatorio, facilitando su computación.

$$E(\boldsymbol{W}) = -\sum_{i=1}^{n} \log p(y = y_i|\boldsymbol{x}_i)$$

A esta función objetivo se la conoce como función de entropía cruzada. Es comúnmente usada en problemas de clasificación.

Capítulo 3

Redes neuronales

Las redes neuronales artificiales (*ANN*, *Artificial Neural Network*) son un paradigma de aprendizaje automático inspirado en la estructura y funcionamiento del cerebro humano. Estos modelos han revolucionado el campo del aprendizaje automático, permitiendo abordar problemas complejos en áreas como la visión por computador, el procesamiento del lenguaje natural y la robótica, entre otras.

Las redes neuronales son modelos computacionales compuestos por unidades básicas llamadas neuronas artificiales o nodos, organizadas en capas y conectadas entre sí. La potencia de las redes neuronales radica en su capacidad para aproximar funciones complejas y no lineales. Esta característica las hace especialmente útiles en tareas donde los métodos tradicionales de aprendizaje automático muestran limitaciones.

Este capítulo ha sido escrito a partir de la información recogida de los libros [13], [8] y [15].

3.1. Concepto de neurona artificial

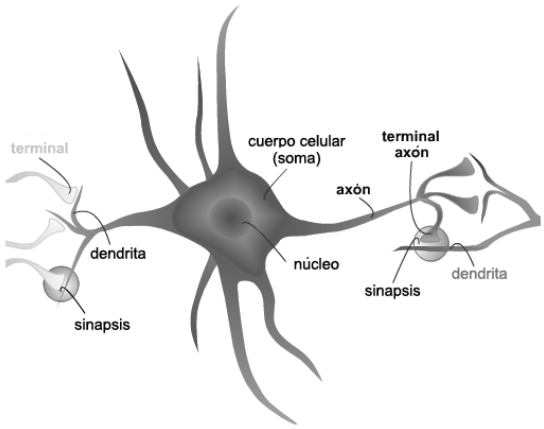

Figura 3.1: Diagrama simplificado de las partes de una neurona biológica.

Las neuronas artificiales, unidades fundamentales de las redes neuronales, están inspiradas en la estructura y funcionamiento de las neuronas biológicas. Aunque simplificadas, las neuronas artificiales capturan aspectos clave de sus contrapartes biológi-

cas. En la Tabla 3.1 se presenta una comparación entre los componentes principales de ambas.

Componente Biológico	**Componente Artificial**	**Función**
Dendritas	Entradas $(x_1, x_2, \ldots, x_n)$	Reciben señales de entrada
Sinapsis	Pesos $(w_1, w_2, \ldots, w_n)$	Modulan la fuerza de las señales de entrada
Soma (cuerpo celular)	Función de suma $(\sum)$	Agrega todas las señales de entrada ponderadas
Umbral de activación	Sesgo (b)	Determina la facilidad de activación de la neurona
Potencial de acción	Función de activación (σ)	Determina la salida de la neurona basada en la entrada agregada.
Axón	Salida (y)	Transmite la señal de salida a otras neuronas

Cuadro 3.1: Comparación entre neuronas biológicas y artificiales

La operación de una neurona artificial con n entradas se puede expresar como:

$$y = \sigma(\sum_{i=1}^{n} w_i x_i + b)$$

Esta operación se reproduce en cada neurona de cada capa de la red, conectando las salidas de la capa anterior en la entrada de la capa siguiente, hasta llegar a la última capa y producir un resultado.

El concepto de neurona artificial se construye en 1943 [1] y más tarde el de red neuronal artificial, gracias al modelo de aprendizaje de Hebb [2]. La primera implementación efectiva de una red neuronal fue el Perceptrón de Frank Rosenblatt en 1961 [4].

3.2. Clasificación en una red neuronal

En el contexto de clasificación, una red neuronal puede verse como una función $f : \mathbb{R}^n \to \mathbb{R}^m$, donde n es la dimensión del vector de entrada y m es el número de clases. Para una red neuronal *feedforward* de L capas, el modelo se puede expresar matemáticamente como:

$$f(\boldsymbol{x}) = f^{(L)}(f^{(L-1)}(\cdots f^{(2)}(f^{(1)}(\boldsymbol{x}))))$$

donde cada $f^{(l)}$ representa la transformación en la capa l. Para una capa densa (también llamada *fully connected*), esta transformación se define como:

$$f^{(l)}(\boldsymbol{h}^{(l-1)}) = \sigma(\boldsymbol{W}^{(l)}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)})$$

$\boldsymbol{h}^{(l-1)}$ es el vector salida de la capa anterior que ahora actúa como entrada, $\boldsymbol{W}^{(l)}$ es la matriz de pesos, $\boldsymbol{b}^{(l)}$ es el vector de sesgos, y σ es una función de activación no lineal.

En la última capa de un problema de clasificación multiclase, típicamente se usa la función *softmax*, definida en la sección 2.4.2. Cada neurona final está asociada a una clase k :

$$f_k^{(L)}(\boldsymbol{h}^{(L-1)}) = p(y = k|\boldsymbol{x}) = \frac{e^{h_k}}{\sum_{j=1}^{K} e^{h_j}}$$

donde h_k es la k-ésima componente del vector de salida de la última capa antes de aplicar *softmax*.

El resultado de las neuronas de esta capa final se traduce como la probabilidad de pertenecer a una determinada clase.

3.3. Teorema universal de aproximación

Teorema 1 (Aproximación Universal para Redes Neuronales)**.** *[8] Sea σ una función no constante, acotada, monótona creciente y continua. Sea I^{m_0} el hipercubo unitario m_0-dimensional $[0,1]^{m_0}$. Denotemos por $C(I^{m_0})$ el espacio de funciones continuas en I^{m_0}. Entonces, dada cualquier función $f \in C(I^{m_0})$ y $\varepsilon > 0$, existen un entero m_1 y conjuntos de constantes reales α_i, b_i, y w_{ij}, donde $i = 1, \ldots, m_1$ y $j = 1, \ldots, m_0$, tales que podemos definir:*

$$F(x_1, \ldots, x_{m_0}) = \sum_{i=1}^{m_1} \alpha_i \sigma \left(\sum_{j=1}^{m_0} w_{ij} x_j + b_i \right)$$

como una aproximación de la función f; es decir,

$$|F(x_1, \ldots, x_{m_0}) - f(x_1, \ldots, x_{m_0})| < \varepsilon$$

para todo $(x_1, \ldots, x_{m_0}) \in I^{m_0}$.

Este teorema fue demostrado en 1989 por Cybenko, se puede consultar la demostración en [6].

Este teorema establece que cualquier función continua en un hipercubo unitario puede ser aproximada con precisión arbitraria por una red neuronal *feedforward* con una sola capa oculta y un número finito de neuronas.

- σ es la función de activación.
- m_0 es la dimensión del espacio de entrada.
- m_1 es el número de neuronas en la capa oculta.
- α_i son los pesos de la salida de la neurona en la posición i.
- w_{ij} son los pesos asociados a la neurona en la posición i de la capa oculta.
- b_i son los sesgos asociados a la neurona en la posición i.
- ε representa la precisión de la aproximación.

Implicaciones importantes:

1. Justifica teóricamente el uso de redes neuronales como aproximadores universales de funciones.

2. Garantiza la existencia de una red que puede aproximar la función, pero no proporciona un método para encontrar los pesos óptimos.

3. No especifica cuántas neuronas son necesarias para lograr una determinada precisión, además de que usualmente es preferible usar varias capas en vez de una sola capa oculta.

3.4. Proceso de aprendizaje: retropropagación y optimización

El aprendizaje en una red neuronal consiste en ajustar los pesos $\mathbf{W}$ (esta notación incluye los sesgos) para minimizar una función objetivo E, de forma que la red produzca predicciones mejores conforme decrece el valor de E. En clasificación, comúnmente se usa la entropía cruzada, definida en la sección 2.4.2

$$E(\boldsymbol{W}) = -\sum_{i=1}^{N} log(p(y = y_i|x_i))$$

donde N es el número de muestras, y $p(y = y_i|x_i)$ es la probabilidad predicha por la red de que la muestra x_i pertenezca a la clase y_i.

3.4.1. Algoritmo de retropropagación

El algoritmo de retropropagación, propuesto por Rumelhart et al. en 1986 [5], es fundamental para el aprendizaje eficiente de los pesos. Se trata de un método estocástico del cálculo del gradiente de la función objetivo. La naturaleza estocástica surge porque calculamos el gradiente para las muestras compuestas por los pares (x_i, y_i), de forma que no conocemos el comportamiento de la red neuronal en otros puntos fuera del conjunto de entrenamiento. Como el par proviene de una variable aleatoria y está sujeto a una variabilidad, el gradiente calculado no es un gradiente determinista, sino estocástico, que tendrá ruido asociado. Por tanto, la convergencia por el descenso del gradiente estocástico será más lenta que con el cálculo determinista del gradiente de una función conocida.

Consiste en dos pasos principales:

1. **Propagación** hacia adelante: Se calcula la salida de la red para una entrada dada.

2. **Retropropagación** del error: Se calculan los gradientes de la función de coste (entropía cruzada en nuestro caso) con respecto a cada peso, propagando el error desde la salida hacia la entrada.

Matemáticamente, para cada peso $w_{ij}^{(l)}$ en la capa l, actualizamos:

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta \frac{\partial E}{\partial w_{ij}^{(l)}}$$

donde $\eta \in \mathbb{R}^+$ es la tasa de aprendizaje. El punto fuerte de la retropropagación es el cálculo eficiente de estas derivadas parciales utilizando la regla de la cadena:

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = \frac{\partial E}{\partial a_i^{(l+1)}} \cdot \frac{\partial a_i^{(l+1)}}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial w_{ij}^{(l)}} \tag{3.1}$$

Donde:

- La salida de la neurona i en la capa $l+1$ es:

$$a_i^{(l+1)} = \sigma(z_i^{(l+1)})$$

- La derivada de la salida con respecto a $z_i^{(l+1)}$ es:

$$\frac{\partial a_i^{(l+1)}}{\partial z_i^{(l+1)}} = \sigma'(z_i^{(l+1)})$$

- La entrada ponderada a la neurona i en la capa $l+1$ es:

$$z_i^{(l+1)} = \sum_j w_{ij}^{(l)} a_j^{(l)}$$

- La derivada de $z_i^{(l+1)}$ con respecto a $w_{ij}^{(l)}$ es:

$$\frac{\partial z_i^{(l+1)}}{\partial w_{ij}^{(l)}} = a_j^{(l)}$$

Entonces, continuando y sustituyendo las expresiones en la ecuación 3.1, el gradiente del error con respecto al peso $w_{ij}^{(l)}$ es:

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = \delta_i^{(l+1)} \cdot a_j^{(l)}$$

donde $\delta_i^{(l+1)}$ es el término de error para la capa $l+1$ y se calcula de forma recursiva.

El primer término de error obtenido es el de la capa final L:

$$\delta_i^{(L)} = \frac{\partial E}{\partial a_i^{(L)}} \cdot \sigma'(z_i^{(L)})$$

Y para las capas ocultas, el término de error $\delta_i^{(l)}$ se calcula propagando el error hacia atrás:

$$\delta_i^{(l)} = \left(\sum_k \delta_k^{(l+1)} w_{ik}^{(l+1)} \right) \cdot \sigma'(z_i^{(l)})$$

Esta fórmula para calcular los gradientes de los pesos, en la práctica se implementa mediante cálculo matricial. De esta forma es mucho más eficiente computacionalmente y permite aumentar la velocidad de entrenamiento de las redes neuronales.

3.4.2. Optimizador SGD

Una vez calculados los gradientes mediante retropropagación, estos se utilizan en un algoritmo de optimización para actualizar los pesos de la red. Los dos más populares son SGD y Adam. El Descenso de Gradiente Estocástico (SGD, *Stochastic Gradient Descent*) actualiza los pesos de la siguiente forma:

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta \frac{\partial E}{\partial w_{ij}^{(l)}}$$

donde η es una constante denominada tasa de aprendizaje. También podemos expresarlo en notación vectorial de forma más compacta, considerando $\boldsymbol{W}^{(l)}$ la matriz de pesos w_{ij} de la capa l y $\mathbf{G}^{(l)} = \frac{\partial E}{\partial \mathbf{W}^{(l)}}$ es el gradiente de la función objetivo respecto a los pesos de la capa l:

$$\boldsymbol{W}^{(l)} \leftarrow \boldsymbol{W}^{(l)} - \eta \mathbf{G}^{(l)}$$

Cuando mayor sea la tasa de aprendizaje, mayores serán los pasos en dirección al mínimo. Sin embargo, puede que nos pasemos de largo si se dan pasos demasiado grandes. Para solventar este problema, una solución es disminur la tasa de aprendizaje progresivamente conforme se avanza en el entrenamiento. De esta forma se comienzan con pasos grandes y se termina acercando al mínimo con mayor precisión, a pasos de menor tamaño.

El descenso por gradiente estocástico (SGD) puede mejorarse utilizando *mini-batches* (pequeños subconjuntos del conjunto de entrenamiento), en lugar de calcular el gradiente y actualizar los pesos en cada muestra del conjunto (método denominado «en línea» u *on line*)).
Esta técnica puede mejorar la convergencia del algoritmo.

En el método por *mini-batches*, el conjunto de datos se divide en subconjuntos de tamaño m. Para cada *mini-batch*, se calcula el gradiente de la función objetivo y se actualizan los pesos de la red. La actualización de pesos se expresa como:

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \frac{\eta}{m} \sum_{k=1}^{m} \mathbf{G}_k^{(l)} \tag{3.2}$$

donde:

- $\mathbf{W}^{(l)}$ es la matriz de pesos de la capa l,
- η es la tasa de aprendizaje,
- m es el tamaño del *mini-batch*,
- $\mathbf{G}_k^{(l)}$ es el gradiente de la función de costo E con respecto a los pesos de la capa l, calculado para la k-ésima muestra del *mini-batch*.

3.4.3. Optimizadores Basados en Momentos

El uso de momentos en SGD ayuda a mejorar la convergencia del modelo al suavizar la trayectoria de actualización de los parámetros. El objetivo es acumular información

sobre las direcciones de los gradientes pasados para avanzar de manera más estable y rápida en la dirección correcta, evitando oscilaciones en direcciones de alta variabilidad. A continuación, se presenta una descripción de dos optimizadores comunes basados en momentos: **SGD con Momentum** y **Adam**.

SGD con Momentum

El optimizador SGD con *momentum* agrega un término de memoria del gradiente anterior al proceso de actualización. La fórmula de actualización de los parámetros $\boldsymbol{W}$ se define como:

$$\boldsymbol{v}^{(l)} \leftarrow \beta \boldsymbol{v}^{(l)} + (1-\beta)\mathbf{G}^{(l)}$$
$$\boldsymbol{W}^{(l)} \leftarrow \boldsymbol{W}^{(l)} - \eta \boldsymbol{v}^{(l)}$$

Donde:

- $\boldsymbol{v}^{(l)}$ la matriz de *momentum* (o velocidad) asociado a los pesos de la capa l. Se inicializa al valor $\mathbf{0}$.
- β es el factor de *momentum*, que controla la contribución del gradiente anterior.
- η es la tasa de aprendizaje.

Este método introduce un promedio ponderado de los gradientes pasados, lo que permite avanzar más rápido en direcciones consistentes y reduce la oscilación en direcciones con alta varianza.

Optimizador Adam

El algoritmo Adam (Adaptive Moment Estimation), propuesto por Kingma y Ba en 2014 [12], es uno de los optimizadores más populares en el aprendizaje profundo. Adam usa la idea del momento, adaptando las tasas de aprendizaje para cada parámetro.

Adam utiliza estimaciones exponencialmente decrecientes de los momentos de primer y segundo orden del gradiente. Las actualizaciones de los parámetros se realizan de la siguiente manera:

Donde ϵ es un pequeño escalar para prevenir la división por cero (típicamente 10^{-8}).

Las características principales de Adam son

- **Tasas de aprendizaje adaptativas**: Adam ajusta las tasas de aprendizaje para cada parámetro basándose en las estimaciones de los momentos.
- **Corrección de sesgo**: Las estimaciones de los momentos se corrigen para contrarrestar su sesgo hacia cero, especialmente durante las primeras iteraciones.
- **Eficiencia computacional**: Requiere poca memoria y es eficiente en términos de cálculo.
- **Convergencia:** rápida en muchos problemas de optimización, comparado a SGD, aunque en algunos problemas puede converger a soluciones subóptimas.

En la práctica, Adam es a menudo la elección predeterminada para muchas aplicaciones de aprendizaje profundo, especialmente en etapas iniciales de desarrollo, debido a su buen rendimiento general y la facilidad de configuración.

Algorithm 1 Optimizador Adam

Requiere α: Tasa de aprendizaje
Requiere $\beta_1, \beta_2 \in [0, 1)$: Tasas de decaimiento exponencial para las estimaciones de los momentos
Requiere $E(\boldsymbol{W})$: Función estocástica objetivo con parámetros $\boldsymbol{W}$
Requiere $\boldsymbol{W}_0$: Parámetros iniciales
$m_0 \leftarrow 0$ (Inicializar vector del primer momento)
$v_0 \leftarrow 0$ (Inicializar vector del segundo momento)
$t \leftarrow 0$ (Inicializar contador de pasos)
mientras $\boldsymbol{W}_t$ no ha convergido **hacer**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_{\boldsymbol{W}} E_t(\boldsymbol{W}_{t-1})$ (Obtener gradientes)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Actualizar primer momento)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Actualizar segundo momento)
    $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Corrección de sesgo primer momento)
    $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Corrección de sesgo segundo momento)
    $\boldsymbol{W}_t \leftarrow \boldsymbol{W}_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Actualizar parámetros)
fin mientras
retornar $\boldsymbol{W}_t$ (Parámetros resultantes)

3.5. Ventajas y desafíos de las redes neuronales

3.5.1. Ventajas de las Redes Neuronales

- **Capacidad de aprendizaje representacional:** Las redes neuronales tienen la habilidad de aprender representaciones complejas y jerárquicas de los datos directamente a partir de grandes volúmenes de datos, extrayendo características y patrones relevantes sin necesidad de ingeniería de características manual.

- **Flexibilidad y adaptabilidad:** Estas arquitecturas son altamente flexibles y pueden ser adaptadas a una amplia gama de problemas y tipos de datos, incluyendo imágenes, texto, y series temporales. La capacidad para ajustar la estructura de la red y los hiperparámetros permite su aplicación en diversos dominios y tareas.

- **Modelado de no linealidades:** Las redes neuronales son capaces de capturar y modelar relaciones complejas y no lineales en los datos, superando las limitaciones de los modelos lineales y permitiendo la representación de estructuras de datos intrínsecamente complejas.

3.5.2. Desafíos y Consideraciones

- **Requerimientos de datos masivos:** El entrenamiento efectivo de redes neuronales profundas suele requerir grandes volúmenes de datos para evitar el sobreajuste y asegurar que el modelo generalice adecuadamente. Esto implica la necesidad de conjuntos de datos extensos y representativos, debido a la usual gran cantidad de parámetros de las redes neuronales más usadas.

- **Costo computacional:** El proceso de entrenamiento de redes neuronales, especialmente aquellas con múltiples capas y parámetros, puede ser altamente intensivo en términos de recursos computacionales y tiempo, comparados a otros métodos de aprendizaje estadístico. Esto a menudo requiere el uso de hardware especializado, como GPUs o TPUs, para realizar cálculos eficientes.
- **Problemas de interpretabilidad:** Las redes neuronales son frecuentemente descritas como «cajas negras» debido a la dificultad inherente en interpretar cómo las decisiones se toman en función de los datos de entrada. Esto representa un desafío significativo en aplicaciones donde la explicación de las decisiones del modelo es crucial.
- **Sobreajuste y regularización:** Las redes neuronales tienen la capacidad de memorizar los datos de entrenamiento, lo que puede llevar al sobreajuste si no se implementan técnicas adecuadas de regularización. Estrategias como la regularización L2, el abandono (*dropout*), y la validación cruzada son esenciales para mitigar este problema.

3.6. Redes neuronales convolucionales (CNN)

Las redes neuronales convolucionales (CNN, *Convolutional Neural Network*) son un tipo especial de redes neuronales artificiales diseñadas principalmente para el procesamiento de datos estructurados en forma de matriz, como las imágenes. A diferencia de las redes neuronales tradicionales, las CNN explotan la estructura espacial de los datos de entrada, lo que les permite capturar características visuales de manera más eficiente Por ello se ha convertido en la herramienta más utilizada del estado del arte del campo del reconocimiento de objetos en imágenes. Este hecho se constató en el año 2012, como se puede observar en el gráfico de la Figura 3.2.

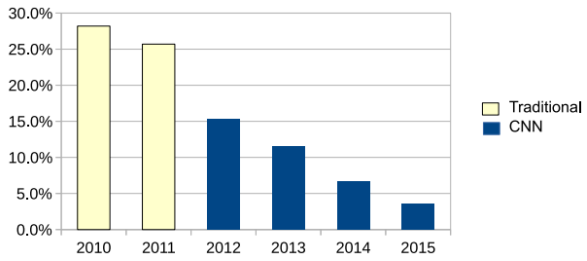

Figura 3.2: Tasa de error de la competición de reconocimiento de imágenes Imagenet. A partir del año 2012, los modelos ganadores fueron redes neuronales convolucionales, que mejoraron en gran medida los resultados de los modelos tradicionales. Fuente: [16].

La arquitectura de una CNN consta típicamente de los siguientes tipos de capas:

- **Capa convolucional**: Aplica un conjunto de filtros (o kernels) aprendidos a la entrada, produciendo mapas de características que capturan patrones locales. Será explicado en profundidad en la siguiente sección.
- **Capa de agrupamiento (*pooling*)**: Reduce la dimensionalidad de los mapas de características, típicamente reducen la información de regiones de la matriz a su valor máximo (*max pooling*). También existen otros tipos de agrupamiento como el *average pooling*, que consiste en calcular la media de la región.

- **Capa densa (*fully connected*)**: Capa típica de una red neuronal convencional donde todas las neuronas de entrada están conectadas con las neuronas de salida, se usa para producir los resultados finales, como las probabilidades de clasificación. Ya ha sido ampliamente explicada a lo largo de este capítulo 3.

3.6.1. Convolución

En su forma más general, la *convolución* es una operación entre dos funciones de una variable real. Para ilustrar su definición, consideremos el siguiente ejemplo: supongamos que estamos rastreando la ubicación de una nave espacial con un sensor láser. Este sensor proporciona una salida $x(t)$, que representa la posición de la nave en el tiempo t, donde tanto x como t son valores reales.

El sensor, sin embargo, está afectado por ruido. Para obtener una estimación más precisa de la posición de la nave, deseamos promediar varias mediciones, dando mayor peso a las más recientes. Este proceso se puede realizar mediante una función de ponderación $w(a)$, donde a es la antigüedad de la medición. Aplicando esta operación de promedio ponderado en cada instante de tiempo, obtenemos una nueva función $s(t)$ que proporciona una estimación suavizada de la posición:

$$s(t) = \int x(a)w(t-a)\,da \tag{3.3}$$

Esta operación se denomina *convolución*, y se representa generalmente con el símbolo asterisco.

$$s(t) = (x * w)(t) \tag{3.4}$$

En este ejemplo, la función w debe ser una función de densidad de probabilidad válida, ya que estamos calculando un promedio ponderado, y debe ser nula para todos los valores negativos de a, pues no tenemos acceso a mediciones futuras. En general, la convolución se define para cualquier par de funciones donde la integral (3.3) esté bien definida, y puede ser utilizada para otros propósitos además de promedios ponderados.

3.6.2. Convolución Discreta

En muchas aplicaciones prácticas, los datos se registran de manera discreta. Supongamos que el sensor láser proporciona una medición por segundo. En este caso, tanto $x(t)$ como $w(t)$ se definen solo en valores enteros de t. La convolución en este contexto se define como:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{3.5}$$

En las redes neuronales convolucionales, el *input* se representa normalmente como un arreglo multidimensional de datos, y el *kernel* es también un arreglo de parámetros que son ajustados mediante el algoritmo de aprendizaje. En la práctica, dado que los valores de x y w solo están definidos en un conjunto finito de puntos, la suma infinita se puede implementar como una suma finita sobre los elementos del arreglo.

Convolución en Dos Dimensiones

Cuando tratamos con imágenes bidimensionales, es común usar convoluciones en dos dimensiones. Si I es una imagen y K es el kernel, la convolución discreta en dos dimensiones se define como

$$S(i,j) = (K * I(i,j)) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{3.6}$$

La convolución es conmutativa, por lo que también podemos escribir

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{3.7}$$

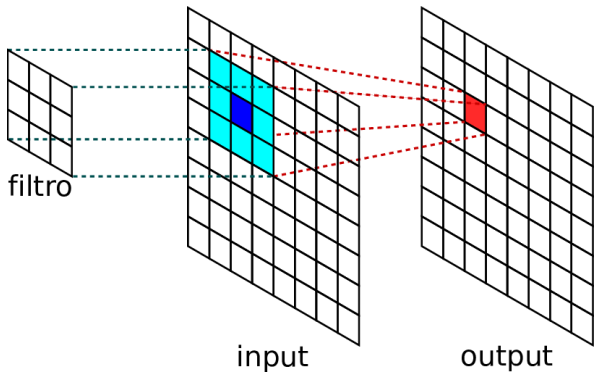

Figura 3.3: Ilustración de la aplicación de un filtro en la operación de convolución.

En las bibliotecas de aprendizaje automático, a menudo se utiliza una operación relacionada llamada *correlación cruzada*, que es similar a la convolución pero sin invertir el kernel:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{3.8}$$

A pesar de que técnicamente la convolución invierte el kernel, en muchas implementaciones de redes neuronales se utiliza la correlación cruzada, aunque se sigue llamando convolución. Esto no afecta el funcionamiento de los algoritmos de aprendizaje, ya que el proceso de entrenamiento ajustará el kernel de manera adecuada, independientemente de si se realiza la inversión o no, debido a la naturaleza conmutativa de la convolución.

En resumen, la convolución discreta puede verse como una multiplicación por una matriz. En el caso unidimensional, esta matriz es una matriz de Toeplitz, donde todas las diagonales son constantes. En el caso bidimensional, se utiliza una matriz circulante por bloques. Además, la convolución suele estar asociada a matrices dispersas, ya que el kernel es generalmente mucho más pequeño que la imagen de entrada.

Esto permite que las CNN aprovechen las capacidades de cómputo paralelo de las unidades de procesamiento gráfico (GPU) para acelerar el entrenamiento y la inferencia de estos modelos.

3.6.3. Entrenamiento de una CNN

Durante el entrenamiento, los parámetros de los filtros convolucionales se actualizan mediante el algoritmo de retropropagación, de manera similar a como se ajustan los pesos en una red neuronal densa, aunque . La combinación de capas convolucionales, de agrupamiento y densas permite a las CNN aprender representaciones jerárquicas de los datos de entrada, especialmente cuando se trata de datos que se organizan naturalmente en forma de matriz, como las imágenes digitales, lo que les confiere una gran capacidad de generalización en tareas de visión por computador.

En comparación a una red neuronal densa, hay muchas menos conexiones entre las capas. En una capa convolucional, una neurona se encuentra conectada a aquellas que afecta a través del filtro. Normalmente se utilizan filtros de tamaño 3×3, lo que limita las conexiones de cada neurona a 9 o 16, en vez de tener conexiones con todas las neuronas de la capa anterior, como ocurriría en una capa densa.

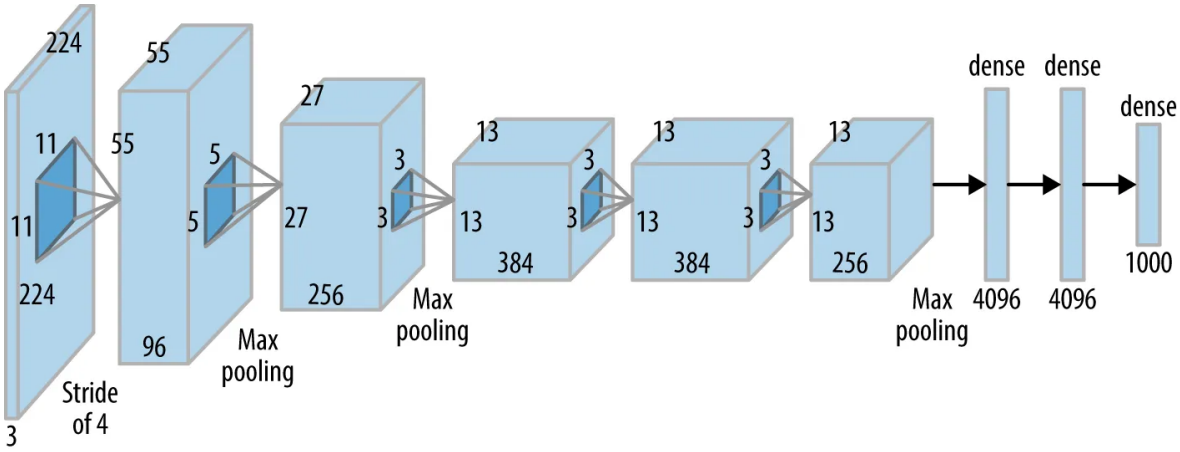

Figura 3.4: Arquitectura de la CNN Alexnet [10], ganadora del concurso ImageNet en 2012.

Capítulo 4

Entrenamiento de una red neuronal convolucional para el reconocimiento de señales de tráfico

En este capítulo se cuenta cómo se ha entrenado la red neuronal elegida y los resultados obtenidos.

4.1. Conjunto de datos

He formado un conjunto de datos único a partir de 3 conjuntos públicos: GTSDB (*German Traffic Sign Detection Benchmark*) [11], TT100K (*Tsinghua-Tencent 100K*) [14], DFG (conjunto de imágenes de señales de tráfico esloveno) [17].

En total, mi conjunto está formado por 764 imágenes para el entrenamiento del modelo y 367 imágenes de validación a partir de las cuales se verificará la calidad del modelo.

El conjunto de clases está balanceado, tal y como se muestra en con la Figura 4.1.

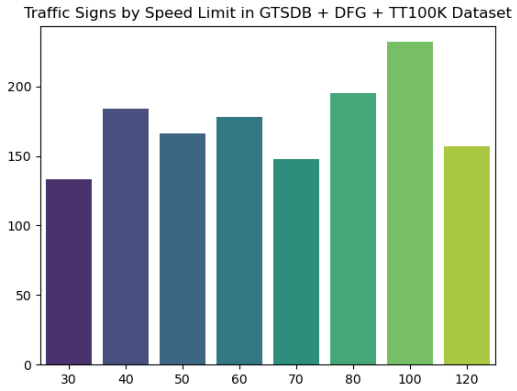

Figura 4.1: Distribución de las clases de señales de limitación de velocidad en el conjunto de datos

4.1.1. Preprocesado

He transformado el formato de las anotaciones de las imágenes para unificarlo, ya que al provenir los datos de diferentes fuentes, estaban expresadas de distinta manera. Cada imagen debe tener asociado un archivo de anotaciones, con la información relativa a las señales de tráfico contenidas y su ubicación.

El formato deseado es el siguiente:

```
clase centroX centroY ancho alto
```

- `clase` es el identificador de la categoría del objeto.
- `centroX` representa la coordenada normalizada x del centro del objeto.
- `centroY` representa la coordenada normalizada y del centro del objeto.
- `ancho` el ancho de la caja delimitadora que envuelve al objeto, normalizado.
- `alto` el alto de la caja delimitadora que envuelve al objeto, normalizado.

Las cajas delimitadoras de los *datasets* de origen están definidos por los extremos en las coordenadas x e y. Esto obliga a realizar una transformación en las coordenadas para unificar el formato.

$$x_{\text{centro}} = \frac{x_{\min} + x_{\max}}{2w}$$
$$y_{\text{centro}} = \frac{y_{\min} + y_{\max}}{2h}$$
$$\text{ancho} = \frac{x_{\max} - x_{\min}}{w}$$
$$\text{alto} = \frac{y_{\max} - y_{\min}}{\text{h}}$$

donde w y h corresponden al ancho y alto de la imagen.

Una vez calculadas estas variables, solamente necesitamos la clase.

$$\begin{gathered}0 = \text{límite 30 km/h},\\ 1 = \text{límite 40 km/h},\\ 2 = \text{límite 50 km/h},\\ 3 = \text{límite 60 km/h},\\ 4 = \text{límite 70 km/h},\\ 5 = \text{límite 80 km/h},\\ 6 = \text{límite 100 km/h},\\ 7 = \text{límite 120 km/h},\end{gathered}$$

Todas estas operaciones han sido llevadas a cabo ejecutando *scripts* escritos por mí en el lenguaje de programación Python, usando las librerías `os`, `json`, `shutil`, `PIL`.

4.2. Métrica mAP

En la evaluación de los modelos de reconocimiento de objetos en imágenes se utiliza comúnmente la métrica mAP o *medium Average Precision*.

Para poder definir esta medida del rendimiento del modelo, debemos realizar un recorrido por los diferentes conceptos en los que se basa.

Definamos los siguientes indicadores básicos::

- **VP** (Verdadero Positivo). El número de instancias que fueron correctamente clasificadas como positivas.
- **FP** (Falso Positivo). El número de instancias que son negativas pero fueron incorrectamente clasificadas como positivas. Por ejemplo, detectar un objeto cuando no hay ninguno o detectar un objeto pero asignarle la clase a la que no pertenece.
- **FN** (Falso Negativo). El número de instancias que son positivas pero fueron incorrectamente clasificadas como negativas. Por ejemplo, la no detección de un objeto.
- **VN** (Verdadero Negativo). El número de instancias que fueron correctamente clasificadas como negativas. Menos relevante en la detección de objetos, ya que toda la imagen se considera negativo excepto el objeto a detectar.

4.2.1. Precisión y Recall

A partir de los conceptos anteriores, podemos construir indicadores que aporten un mayor significado.

La **precisión** es una métrica que mide la proporción de verdaderos positivos entre todos los resultados positivos (tanto verdaderos como falsos):

$$\text{Precisión} = \frac{VP}{VP + FP}$$

La métrica **sensibilidad** o *recall* mide la proporción de verdaderos positivos entre todos los datos positivos (verdaderos positivos más falsos negativos):

$$\text{Sensibilidad} = \frac{VP}{VP + FN}$$

F1 es una medida de evaluación que combina tanto la precisión como la sensibilidad, para que muestre de forma más fiable el desempeño de un modelo. Se calcula como una media armónica:

$$\text{F1} = 2 \cdot \frac{\text{Precisión} \cdot \text{Sensibilidad}}{\text{Precisión} + \text{Sensibilidad}}$$

4.2.2. Intersection over Union

Intersection Over Union **(IoU)** mide la superposición entre el área del objeto predicho y el área real de ese objeto.

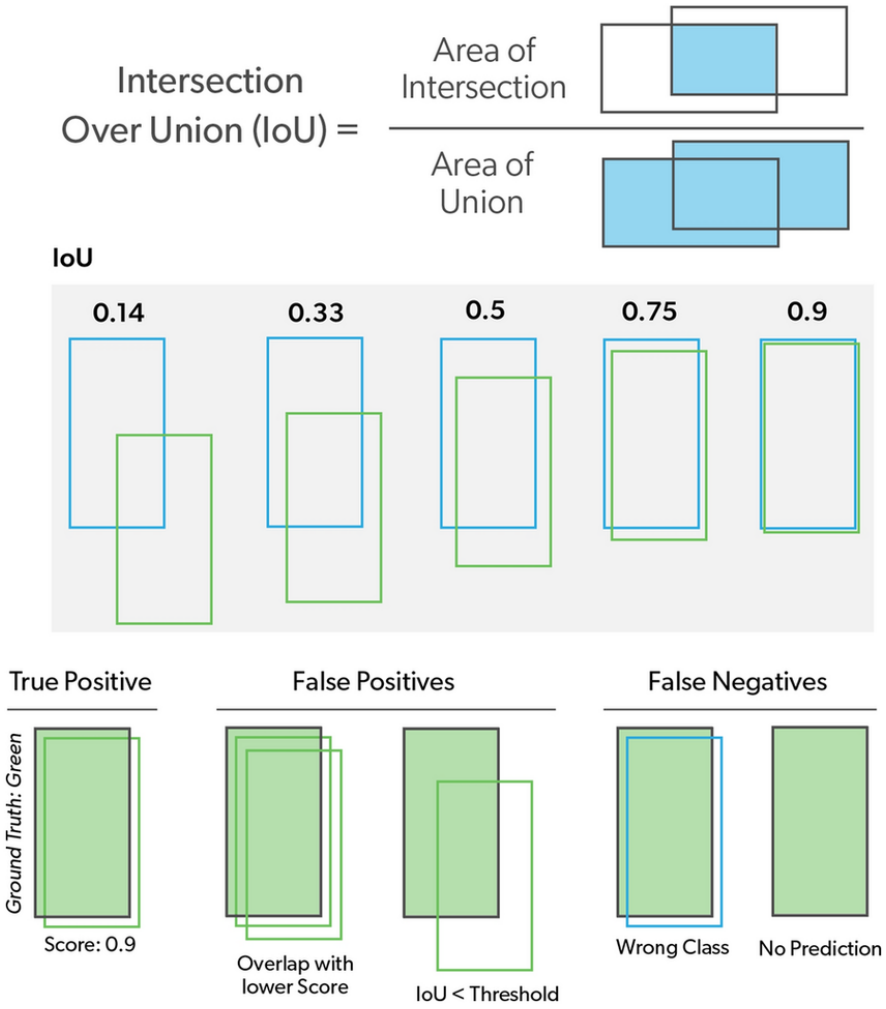

Figura 4.2: Ejemplo de distintas medidas IoU y su efecto en la detección de objetos. El primer caso de falso positivo se debe a un filtrado durante el entrenamiento en el que los modelos se quedan con la predicción de mayor puntuación IoU y desechan el resto como FP. Fuente: [19]

.

$$\text{IoU} = \frac{\text{Área de Intersección}}{\text{Área de Unión}}$$
$$0 \leq \text{IoU} \leq 1$$

Cuanto más se acerque a 1, mejor será la predicción, y si IoU = 1, la caja predicha coincide exactamente con la caja real. Se puede marcar un umbral IoU que separe un VP de un FP.

4.2.3. Confianza

Se denomina **confianza** a la probabilidad producida por el modelo de que un objeto pertenezca a una clase concreta. El **umbral de confianza** limita los posibles resultados a aquellos que sobrepasen un determinado umbral, entre 0 y 1.

Todas los indicadores anteriores son altamente sensibles al valor del umbral de confianza.

Si se selecciona un umbral de confianza bajo, como por ejemplo 0.01, el modelo tiende a clasificar muchos casos como positivos, dado que la mayoría de las predicciones tendrán al menos un valor de confianza de 0.01. Esto resulta en un alto número de positivos, que incluirá tanto falsos positivos como verdaderos positivos. Como consecuencia, se obtiene una alta sensibilidad pero una baja precisión.

Por el contrario, al optar por un umbral de confianza más alto, el número de positivos identificados disminuye. Sin embargo, los positivos identificados en este caso tendrán una mayor probabilidad de ser verdaderos positivos, lo que resulta en una alta sensibilidad y alta precisión.

Esta relación entre el umbral de confianza, la precisión y la sensibilidad se puede visualizar gráficamente mediante la curva Precisión-Recall (PR). Para cada valor del umbral de confianza c, se obtiene un par de valores (p_c, r_c), donde p_c es la precisión y r_c es la sensibilidad, resultado de todo el conjunto de entrenamiento. Al representar en el plano XY estos pares de valores, con la precisión en el eje X y el sensibilidad en el eje Y, y luego interpolar entre los puntos, se obtiene la curva $p(r)$, que es fundamental para calcular la métrica AP (*Average Precision*).

4.2.4. AP y mAP

La métrica **AP** (*Average Precision*) es la integral definida entre 0 y 1 de la curva PR.

$$\text{AP} = \int_0^1 p(r)\, dr$$

AP es independiente de la confianza que se elija para detectar objetos, por eso es una métrica importante para ver el rendimiento del modelo. Cuanto más cerca de 1 esté, mejor funcionará la detección y clasificación del modelo.

Cuando tratamos múltiples clases de objetos, se debe obtener una métrica que refleje el rendimiento global del modelo. La métrica **mAP** (*mean Average Precision*) proporciona esta información al calcular el promedio aritmético de las AP para cada clase.

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}_i \ ,$$

donde N es el número de clases.

Hay diferentes métodos para calcular el mAP según los umbrales IoU. La notación mAP@.5 indica que se utiliza un umbral IoU fijo de 0,5. En contraste, la notación mAP@.5:.95:.5 denota un enfoque distinto, donde el umbral IoU varía de 0,5 a 0,95 en pasos de 0,05. En este caso, se calcula el mAP para cada umbral y luego se promedia el resultado. El usado en mis resultados es el mAP con umbral fijo 0,5.

4.3. YOLOv7

YOLOv7 es una arquitectura de red neuronal convolucional igual que el resto de modelos del estado del arte de la detección y clasificación de objetos en imágenes. Me he decantado por la familia YOLO frente a sus competidores como Faster-RCNN, EfficientDet, SSD o RetinaNet debido a su gran velocidad de inferencia y los buenos resultados obtenidos en artículos de este campo. En los recientes artículos [21], [18] y [20] , se puede observar el uso de YOLO en el reconocimiento de señales de tráfico, además de su superioridad frente al resto de modelos probados por los diferentes autores.

Dentro de la familia de modelos YOLO, gracias al artículo de Terven [22], me he decantado por la versión 7 [24], debido a sus resultados en el reconocido *benchmark* de reconocimiento de objetos COCO2017.

Cuadro 4.1: Comparación de las versiones de YOLO. Fuente: [22]. El mAP calculado para YOLO y YOLOv2 viene del *dataset* VOC2007, mientras que el resto se basaron en COCO2017.

Versión	**Fecha**	**Anclaje**	**Framework**	**Backbone**	**AP (%)**
YOLO	2015	No	Darknet	Darknet24	63.4
YOLOv2	2016	Sí	Darknet	Darknet24	78.6
YOLOv3	2018	Sí	Darknet	Darknet53	33.0
YOLOv4	2020	Sí	Darknet	CSPDarknet53	43.5
YOLOv5	2020	Sí	Pytorch	YOLOv5CSPDarknet	55.8
PP-YOLO	2020	Sí	PaddlePaddle	ResNet50-vd	45.9
Scaled-YOLOv4	2021	Sí	Pytorch	CSPDarknet	56.0
PP-YOLOv2	2021	Sí	PaddlePaddle	ResNet101-vd	50.3
YOLOR	2021	Sí	Pytorch	CSPDarknet	55.4
YOLOX	2021	No	Pytorch	YOLOXCSPDarknet	51.2
PP-YOLOE	2022	No	PaddlePaddle	CSPRepResNet	54.7
YOLOv6	2022	No	Pytorch	EfficientRep	52.5
YOLOv7	2022	No	Pytorch	YOLOv7Backbone	**56.8**
DAMO-YOLO	2022	No	Pytorch	MAE-NAS	50.0
YOLOv8	2023	No	Pytorch	YOLOv8CSPDarknet	53.9
YOLO-NAS	2023	No	Pytorch	NAS	52.2

4.3.1. Arquitectura General

YOLOv7 es una red neuronal convolucional que transforma una imagen de entrada en un conjunto de predicciones.

El proceso que sigue YOLO a la hora de realizar una inferencia es el siguiente:

1. Divide la imagen en $S \times S$ celdas

2. Para cada celda de la cuadrícula, la red predice:

$$\begin{aligned} t_x, t_y &\in \mathbb{R} \quad \text{(offsets del centro, respecto a la celda)} \\ t_w, t_h &\in \mathbb{R} \quad \text{(escala del tamaño, respecto a la celda)} \\ t_o &\in \mathbb{R} \quad \text{(confianza de objeto)} \\ t_{c_1}, \ldots, t_{c_C} &\in \mathbb{R} \quad \text{(scores de clase)} \end{aligned}$$

3. Las predicciones se decodifican de la siguiente manera:

$$\begin{aligned} x &= \sigma(t_x) + i \\ y &= \sigma(t_y) + j \\ w &= p_w \cdot e^{t_w} \\ h &= p_h \cdot e^{t_h} \end{aligned}$$

   Donde:

   - $\sigma(x) = \frac{1}{1+e^{-x}}$ es la función sigmoide
   - i, j son las coordenadas de la celda en la cuadrícula
   - p_w, p_h son parámetros predefinidos

4. Interpretación de las coordenadas:

   - (x, y) representa el centro del objeto relativo a la celda.
   - La función sigmoide asegura que $x, y \in (0, 1)$ relativo a la celda.
   - (w, h) representan el ancho y alto del objeto relativos al tamaño total de la imagen.
   - La función exponencial permite predecir un amplio rango de tamaños.

5. Cálculo de la confianza y clase:

$$\begin{aligned} \text{confianza} &= \sigma(t_o) \\ \text{scores de clase} &= \text{softmax}([t_{c_1}, \ldots, t_{c_C}]) \end{aligned}$$

6. Generación de cajas delimitadoras.
   Para cada celda, se genera una caja delimitadora basada en las predicciones:

$$\begin{aligned} x_{min} &= (x - w/2) \cdot W \\ y_{min} &= (y - h/2) \cdot H \\ x_{max} &= (x + w/2) \cdot W \\ y_{max} &= (y + h/2) \cdot H \end{aligned}$$

   Donde W, H son el ancho y alto de la imagen original.

Matemáticamente, este proceso se puede representar como una función:

$$f : \mathbb{R}^{H \times W \times 3} \rightarrow \mathbb{R}^{S \times S \times (5+C)}$$

Donde:

- $H \times W \times 3$ son las dimensiones de la imagen de entrada (altura, ancho y 3 canales de color)
- $S \times S$ es el número de celdas en la cuadrícula de salida
- 5 representa las 4 coordenadas de la caja (x, y, ancho, alto) más un valor de confianza
- C es el número de clases a predecir

4.3.2. Filtrado y NMS

Se aplica un umbral de confianza para descartar las predicciones con menor confianza.

También se aplica *Non-Maximum Suppression* (NMS), que controla las colisiones de cajas delimitadoras. Si el área de superposición de dos cajas está por encima de un porcentaje establecido, se elimina aquella con menor puntuación de confianza. Es especialmente útil cuando se predice que un objeto puede pertenecer a dos clases distintas, y las cajas delimitadoras de cada clase son diferentes pero se superponen.

Una vez aplicado NMS, se obtienen las detecciones finales.

4.3.3. Función objetivo

La función objetivo de YOLOv7 combina varios componentes:

$$L = \lambda_{\text{coord}} \cdot L_{\text{box}} + \lambda_{\text{obj}} \cdot L_{\text{obj}} + \lambda_{\text{noobj}} \cdot L_{\text{noobj}} + \lambda_{\text{class}} \cdot L_{\text{class}}$$

Donde:

- L_{box} es la pérdida de regresión de cajas (error cuadrático medio)
- L_{obj} y L_{noobj} son pérdidas de confianza para objetos y no objetos
- L_{class} es la pérdida de clasificación (entropía cruzada categórica)
- Los λ son hiperparámetros que ponderan la importancia de cada componente

De esta forma, el modelo ajusta las 3 etapas necesarias en nuestro problema: localización, detección y clasificación (sección 1.1).

4.4. Entrenamientos

Se han realizado 4 entrenamientos distintos con YOLOv7 a partir del conjunto de datos 4.1. Las diferencias entre estos entrenamientos están recogidas en la tabla 4.2. El código del modelo se ha descargado del repositorio de GitHub oficial de YOLOv7 [23].

Se ha tocado la **resolución** para comprobar si el modelo puede reconocer las señales de tráfico a 640 píxeles de ancho o si es necesario aumentar la resolución a 1280 píxeles.

Se ha hecho una prueba en blanco y negro para comprobar si la información de color es realmente necesaria.

Se ha probado con dos tamaños de YOLOv7, el *medium* y el *tiny*. Hay una gran diferencia de parámetros, ya que YOLOv7 *tiny* posee 6.2 millones de parámetros; mientras que YOLOv7 *medium* amplia ese número a 36.9 millones de parámetros, que le permite adaptarse a patrones más complejos, a costa de un aumento en el tiempo de entrenamiento e inferencia.

Todos los entrenamientos de YOLOv7 han puesto en práctica lo siguiente:

- Algoritmo de retropropagación con el optimizador Adam.
- Tamaño de *mini-batch* de 64 o de 8 imágenes, dependiendo de la resolución de imagen fuera 1280 píxeles o 640 píxeles, ya que el tamaño del *mini-batch* está limitado por el espacio que ocupe el lote en la memoria RAM del ordenador.
- Número de épocas: 200. Se llama época a una iteración completa del algoritmo de retropropagación por todo el conjunto de entrenamiento.
- *Data augmentation*: se han realizado transformaciones de escalado, rotación o cambio de color a las imágenes a lo largo del entrenamiento, para evitar el sobreajuste.
- Transferencia de aprendizaje: se han usado como pesos iniciales de la red, a excepción de la capa de salida, aquellos producidos por un entrenamiento de la red en el conjunto COCO2017, que es un *benchmark* muy grande de detección de objetos.

Resolución	A color	Modelo
640	✓	YOLOv7-tiny
1280	✓	YOLOv7-tiny
640	Escala de grises	YOLOv7-tiny
1280	✓	YOLOv7-medium

Cuadro 4.2: Entrenamientos realizados.

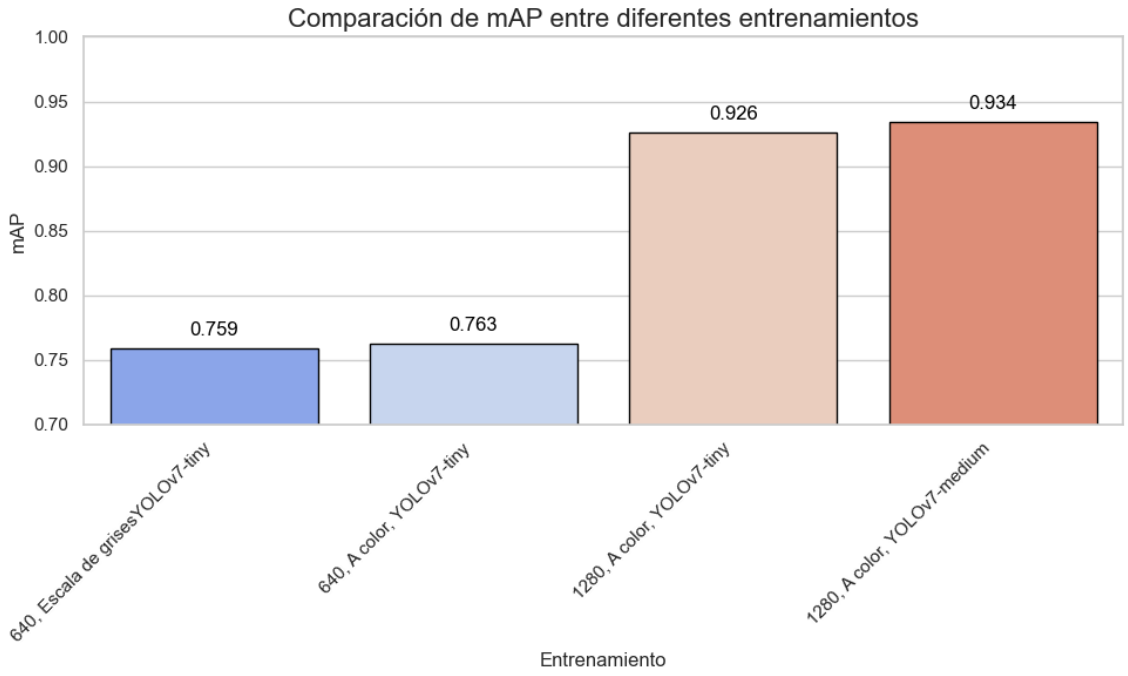

Figura 4.3: Comparación de las diferentes configuraciones del entrenamiento de la red YOLOv7, los resultados de la medida de rendiminento mAP están aplicados al conjunto de validación.

4.5. Resultados

Según vemos en la Figura 4.3, el tamaño de YOLOv7 vencedor ha sido el *medium*, con un resultado de la métrica mAP de 93.4 %. En segundo lugar, la misma configuración (1280 píxeles de ancho, a color) pero con el tamaño *tiny*.

Con una gran diferencia en la métrica, se encuentran los dos entrenamientos realizados a una resolución menor, 640 píxeles de ancho.

Otro aspecto a observar es que el entrenamiento realizado en escala de grises obtiene el último puesto, por debajo de la misma configuración, en la que se usan imágenes a color.

Se muestran a cotinuación algunos ejemplos relevantes del conjunto de validación producidas por el mejor modelo entrenado, YOLOv7 *medium*.

Figura 4.4: Reconocimiento de una señal desde un ángulo lateral.

Figura 4.5: Reconocimiento de una señal parcialmente tapada.

Figura 4.6: Reconocimiento de una señal desgastada. La confianza es baja: 0.05.

Figura 4.7: Reconocimiento de una señal ligeramente tapada por las ramas de un árbol.

Figura 4.8: Reconocimiento de una señal lejana, pequeña respecto al tamaño de la imagen. Podemos además observar un falso positivo con una confianza de 0.01 que en la práctica se desecharía.

4.6. Conclusiones

Gracias a los 4 entrenamientos realizados, se puede concluir que:

- El **color** proporciona información adicional que no se puede obtener en una imagen en escala de grises. La explicación a este hecho se debe a que las señales del conjunto de datos usado están compuestas por un círculo exterior rojo y un interior blanco. El color rojo no es común en un entorno de carretera o urbano, por lo que ayuda al detector a reconocer la ubicación de las señales.
- El **tamaño** de la red neuronal *medium* permite adaptarse mejor a los patrones del conjunto de datos, dotándolo de un mejor resultado respecto al tamaño *tiny*.
- Cuanta mejor sea la **resolución** a la que se produce el entrenamiento, mejores resultados se obtienen. Esto es debido a que en ocasiones las señales tienen un tamaño muy pequeño relativo al tamaño de la imagen total. Por tanto, si se reduce la resolución, se puede llegar a perder la información necesaria para localizar la señal. Sin embargo, al aumentar la resolución también se aumenta la complejidad computacional y los tiempos de procesamiento.

Por otro lado, de forma más general,

- Se ha **generado un modelo** capaz de detectar señales de tráfico de limitación de velocidad con buena precisión.
- Se ha **generado un conjunto de datos único** a partir de conjuntos de datos públicos ya existentes, filtrando las señales de limitación de velocidad y combinando los formatos de las distintas procedencias.

- Se ha **investigado el estado del arte** de la detección y reconocimiento de objetos.

Finalmente, desde un punto de vista matemático,

- Se han comprendido las bases de los **métodos de clasificación** clásicos.
- Se ha comprendido el funcionamiento de las **redes neuronales**, el algoritmo de la retropropagación y los optimizadores; y también se ha ahondado en los principios clave de las redes neuronales convolucionales.

Bibliografía

[1] Warren S. McCulloch y Walter Pitts. «A logical calculus of the ideas immanent in nervous activity». En: *The Bulletin of Mathematical Biophysics* 5 (4 dic. de 1943), págs. 115-133. ISSN: 0007-4985. DOI: 10.1007/BF02478259.

[2] Donald O Hebb. «The first stage of perception: growth of the assembly». En: *The Organization of Behavior* 4.60 (1949), págs. 78-60.

[3] Richard Bellman. «On the approximation of curves by line segments using dynamic programming». En: *Communications of the ACM* 4.6 (1961), pág. 284.

[4] Frank Rosenblatt. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. Inf. téc. Cornell Aeronautical Lab Inc Buffalo NY, 1961.

[5] David E Rumelhart, Geoffrey E Hinton y Ronald J Williams. «Learning representations by back-propagating errors». En: *nature* 323.6088 (1986), págs. 533-536.

[6] George Cybenko. «Approximation by superpositions of a sigmoidal function». En: *Mathematics of control, signals and systems* 2.4 (1989), págs. 303-314.

[7] Sushmito Ghosh y Douglas L Reilly. «Credit card fraud detection with a neural-network». En: *System Sciences, 1994. Proceedings of the Twenty-Seventh Hawaii International Conference on*. Vol. 3. IEEE. 1994, págs. 621-630.

[8] Simon Haykin. *Neural Networks and Learning Machines*. 3.ª ed. Pearson, 2010.

[9] Sergio Escalera et al. *Traffic-Sign Recognition Systems*. Springer London, 2011. ISBN: 978-1-4471-2244-9. DOI: 10.1007/978-1-4471-2245-6.

[10] Alex Krizhevsky, Ilya Sutskever y Geoffrey E Hinton. «Imagenet classification with deep convolutional neural networks». En: *Advances in neural information processing systems* 25 (2012).

[11] Institut für Neuroinformatik. *German Traffic Sign Detection Benchmark*. Accedido: 23-04-2024. 2013. URL: https://benchmark.ini.rub.de/gtsdb_dataset.html.

[12] Diederik P Kingma. «Adam: A method for stochastic optimization». En: *arXiv preprint arXiv:1412.6980* (2014).

[13] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[14] Z. Zhu et al. «Traffic-Sign Detection and Classification in the Wild». En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. https://cg.cs.tsinghua.edu.cn/traffic-sign/. 2016, págs. 2110-2118.

[15] Hamed Habibi Aghdam y Elnaz Jahani Heravi. *Guide to Convolutional Neural Networks: A Practical Application to Traffic-Sign Detection and Classification*. Springer International Publishing, 2017. ISBN: 978-3-319-57549-0. DOI: 10.1007/978-3-319-57550-6.

[16] Léon Bottou, Frank E Curtis y Jorge Nocedal. «Optimization methods for large-scale machine learning». En: *SIAM review* 60.2 (2018), págs. 223-311.

[17] Domen Tabernik y Danijel Skočaj. «Deep learning for large-scale traffic-sign detection and recognition». En: *IEEE transactions on intelligent transportation systems* 21.4 (2019), págs. 1427-1440.

[18] Ahmed Nusayer Ashik et al. «Recognizing Bangladeshi Traffic Signs in the Wild». En: IEEE, dic. de 2022, págs. 1004-1009. ISBN: 979-8-3503-4602-2. DOI: 10.1109/ICCIT57492.2022.10055612.

[19] Emily Long et al. «Automated corrosion detection in Oddy test coupons using convolutional neural networks». En: *Heritage Science* 10 (sep. de 2022). DOI: 10.1186/s40494-022-00778-3.

[20] Yanzhao Zhu y Wei Qi Yan. «Traffic sign recognition based on deep learning». En: *Multimedia Tools and Applications* 81 (13 mayo de 2022), págs. 17779-17791. ISSN: 1380-7501. DOI: 10.1007/s11042-022-12163-0.

[21] Emel Soylu y Tuncay Soylu. «A performance comparison of YOLOv8 models for traffic sign detection in the Robotaxi-full scale autonomous vehicle competition». En: *Multimedia Tools and Applications* (ago. de 2023). ISSN: 1380-7501. DOI: 10.1007/s11042-023-16451-1.

[22] Juan Terven, Diana-Margarita Córdova-Esparza y Julio-Alejandro Romero-González. «A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas». En: *Machine Learning and Knowledge Extraction* 5.4 (2023), págs. 1680-1716.

[23] Chien-Yao Wang, Alexey Bochkovskiy y Hong-Yuan Mark Liao. *YOLOv7 GitHub repository*. Accedido: 07-07-2024. 2023. URL: https://github.com/WongKinYiu/yolov7.

[24] Chien-Yao Wang, Alexey Bochkovskiy y Hong-Yuan Mark Liao. «YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors». En: *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2023, págs. 7464-7475. DOI: 10.1109/CVPR52729.2023.00721.