Trabajo Fin de Grado
en Ingeniería de las Tecnologías de Telecomunicación

Reconocimiento del lenguaje de signos utilizando inteligencia artificial

Autor: Pablo Rasco Miranda

Tutor: Francisco José Simois Tirado

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2025

Trabajo Fin de Grado
en Ingeniería de las Tecnologías de Telecomunicación

Reconocimiento del lenguaje de signos utilizando inteligencia artificial

Autor:
Pablo Rasco Miranda

Tutor:
Francico José Simois Tirado
Profesor Contratado Doctor

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2025

Autor: Pablo Rasco Miranda

Tutor: Francisco José Simois Tirado

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2025

El Secretario del Tribunal

A mi compañera de vida

A mi familia

A mis maestros

Agradecimientos

Gracias a mi compañera de vida, Guadalupe, por acompañarme, animarme y apoyarme en este camino y hacerme ver que hay cosas más importantes que este trabajo y otros, sin tu paciencia y apoyo no hubiera sido posible.

Gracias por haberme permitido sacar todo mi potencial y ayudarme a terminar la carrera que, sin tu apoyo, ánimo y algún toque de atención, más que necesario, no habría sido posible. Has sido, eres y serás mi apoyo, hombro para llorar, de risa o de tristeza y la luz que ilumina mi camino. Simplemente gracias, te quiero muchísimo.

Gracias a mis amigos por apoyarme y hacerme el tiempo libre, y no tan libre, tan maravilloso. Gracias Diego, por nuestro trabajo en equipo, y el que nos queda juntos. Gracias Rafa, por tantas horas de biblioteca y/o cafetería, depende del día. Gracias Pablo, por 4 años de convivencia y estudio y tantos de amistad. Gracias Kiko y Martín, mis BackStreet Boys, por volverme loco a veces, pero siempre estar ahí para todo.

Gracias a mis padres, por darme la oportunidad de estudiar lo que me gusta y animarme siempre. Gracias a mi hermano, por darme la motivación diaria para dar siempre el máximo desde pequeño, eres la persona de la que más he aprendido en esta vida. Gracias a mis abuelos por, desde la distancia, estar siempre pendientes de mí y darme apoyo en cada paso que doy.

Gracias a mis profesores por tener tanta paciencia y transmitirme vuestra sabiduría. Gracias Juanjo, por confiar en mí para colaborar contigo, ayudarme siempre y enseñarme tanto. Gracias Juan Antonio, por apoyarme en mis proyectos y hacerme ver mi propio valor, que a veces ni yo mismo veo. Gracias María José por ser tan paciente conmigo y mis fallos tontos, por siempre estar dispuesta a ayudarme y por animarme siempre a dar el máximo. Gracias Irene, por ayudarme siempre que lo he necesitado y animarme a desarrollar mis ideas. Gracias Begoña, por propocionarme siempre la ayuda que he necesitado. Gracias Francisco, por darme la oportunidad de realizar este trabajo y ayudarme en todo lo que he necesitado, ha sido un placer tenerte de tutor.

Pablo Rasco Miranda

Sevilla, 2025

Resumen

La comunicación es un pilar fundamental en la interacción humana, pero para las personas con discapacidad auditiva, el desconocimiento generalizado del lenguaje de signos representa una barrera significativa. Esta limitación afecta su inclusión social, laboral y educativa, dificultando el acceso a oportunidades y servicios esenciales. La necesidad de soluciones accesibles y efectivas para superar estas barreras es evidente y urgente.

La inteligencia artificial surge como una herramienta transformadora para abordar esta problemática, ofreciendo sistemas automatizados de reconocimiento del lenguaje de signos. Estos sistemas, basados en redes neuronales convolucionales, permiten interpretar gestos en tiempo real a partir de secuencias de imágenes, garantizando una comunicación inmediata y fluida entre personas sordas y oyentes. Este avance no solo facilita el acceso a la interacción diaria, sino que también elimina barreras en entornos clave como la educación, el trabajo y los servicios médicos.

Más allá de la tecnología, este enfoque responde a una necesidad social crítica: construir una sociedad más inclusiva y equitativa. Al permitir una comunicación accesible, no solo se beneficia a las personas con discapacidad auditiva, sino que también se fomenta la empatía y el respeto en la sociedad, promoviendo una cultura de accesibilidad para todos.

Abstract

Communication is a fundamental pillar in human interaction, but for people with hearing disabilities, the widespread lack of knowledge of sign language represents a significant barrier. This limitation affects their social, work and educational inclusion, making it difficult to access essential opportunities and services. The need for accessible and effective solutions to overcome these barriers is evident and urgent.

Artificial intelligence emerges as a transformative tool to address this problem, offering automated sign language recognition systems. These systems, based on convolutional neural networks, allow gestures to be interpreted in real time from image sequences, guaranteeing immediate and fluid communication between deaf and hearing people. This advance not only facilitates access to daily interaction but also eliminates barriers in key environments such as education, work and medical services.

Beyond technology, this approach responds to a critical social need: to build a more inclusive and equitable society. By enabling accessible communication, not only are people with hearing impairments benefited, but empathy and respect are fostered in society, promoting a culture of accessibility for all.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Glosario

IA	Inteligencia Artificial (Artificial Intelligence)
YOLO	You Only Look Once
CNN	Red Neuronal Convolucional (Convolutional Neuronal Network)
ML	Aprendizaje Máquina (Machine Learning)
DL	Aprendizaje Profundo (Deep Learning)
NN	Red Neuronal (Neuronal Network)
SNN	Red Neuronal Estándar (Standard Neuronal Network)
RNN	Red Neuronal Recurrente (Recurrent Neuronal Network)
GAN	Red generativa antagónica
VAE	Codificador automático variable
LSTM	Memoria a Largo o Corto Plazo (Long Short-Term Memory)
GRU	(Gated Recurrent Unit)
NLP	Procesamiento del Lenguaje Natural (Natural Language Processing)
Batch	Lote
FPS	Fotogramas por segundo

1 PREFACIO

Espero inspirar a la gente que oye. Las personas oyentes tienen la capacidad de eliminar las barreras que impiden a las personas sordas alcanzar sus sueños.

- Marlee Matlin-

La IA (Inteligencia Artificial) está revolucionando la manera en que abordamos los desafíos de la vida diaria, ofreciendo soluciones innovadoras que antes eran inimaginables. Sin embargo, su adopción sigue enfrentando ciertas reticencias debido a preocupaciones ampliamente compartidas. Estas inquietudes, aunque comprensibles, no deben nublar la importancia de su implementación en áreas donde la necesidad de innovación es crítica, como la accesibilidad y la inclusión.

Entre los principales temores asociados a la IA destacan su impacto laboral, pérdida de privacidad y en el caso más extremo la rebelación de las máquinas frente al ser humano.

A pesar de estas preocupaciones, no podemos ignorar el potencial transformador de la IA en la resolución de problemas complejos y en la mejora de la calidad de vida de las personas. En el caso del lenguaje de signos, las barreras comunicativas que enfrentan las personas sordas siguen siendo un desafío urgente que afecta su inclusión social, educativa y laboral.

Un sistema de detección y traducción de lenguaje de signos basado en IA, como el que se puede desarrollar con modelos avanzados como YOLOv8, aborda directamente esta necesidad. Estas tecnologías tienen la capacidad de interpretar gestos en tiempo real, permitiendo una comunicación más fluida entre personas sordas y oyentes. En entornos educativos, pueden facilitar el aprendizaje al traducir contenido en tiempo real; en el ámbito laboral, permiten una mayor integración en procesos clave; y en la vida cotidiana, eliminan barreras que limitan la participación plena en actividades esenciales.

La implementación de un sistema de este tipo no solo aporta un beneficio práctico inmediato, sino que también representa un compromiso con una sociedad más inclusiva y accesible. La IA, en este caso, deja de ser solo una innovación tecnológica para convertirse en una herramienta fundamental que responde a una necesidad social urgente y promueve una interacción más equitativa para todos.

1.1 Motivación

La creciente necesidad de mejorar la comunicación entre personas sordas y oyentes ha sido un factor clave en la realización de este trabajo. A pesar de los avances en tecnologías de accesibilidad, la barrera comunicativa sigue siendo un obstáculo importante en contextos educativos, laborales y sociales. Esto genera exclusión y limita las oportunidades de participación plena para las personas con discapacidad auditiva. Por ello, es esencial desarrollar soluciones innovadoras que promuevan la inclusión y la equidad en la comunicación.

1.2 Propósito y objetivos

El propósito principal de este trabajo es desarrollar un sistema de detección de lenguaje de signos mediante IA, específicamente utilizando CNNs (Redes Convolucionales) como YOLOv8. Los objetivos son:

- Facilitar la comunicación en tiempo real entre personas sordas y oyentes.
- Mejorar la accesibilidad y promover la inclusión social en contextos diversos como la educación y el ámbito laboral.
- Evaluar la precisión y el tiempo de respuesta del sistema en situaciones reales.

1.3 Ámbito, alcance y limitaciones

El ámbito de este trabajo se centra en la creación y evaluación de un sistema basado en inteligencia artificial para la interpretación de gestos del lenguaje de signos en situaciones cotidianas.

El alcance de la investigación incluye:

- El diseño, implementación y evaluación de un modelo de detección de lenguaje de signos.
- La recolección de datos relevantes, su procesamiento y el entrenamiento del modelo utilizando técnicas de aprendizaje profundo.
- Desarrollo de una aplicación en tiempo real.

Los límites de la investigación se enfocan en:

- La interpretación de una selección específica de gestos del lenguaje de signos.
- La integración del sistema con plataformas tecnológicas existentes, sin abordar la creación de nuevas interfaces o aplicaciones externas.
- El coste computacional limitado del que disponemos.

1.4 Metodología

La metodología utilizada para este trabajo incluye los siguientes pasos:

- Búsqueda de datos: Se ha buscado un dataset ya etiquetado que se comentará más adelante para utilizarlo como base de datos de nuestro modelo.
- Entrenamiento del modelo: Se emplearon técnicas de aprendizaje profundo utilizando YOLOv8 para entrenar el modelo en la identificación de gestos.

- Evaluación: El desempeño del sistema se evaluó mediante pruebas de precisión y tiempo de respuesta, simulando situaciones reales de interacción.
- Optimización: Se utilizó un enfoque iterativo para mejorar la precisión y eficiencia del modelo, ajustando parámetros y evaluando los resultados en cada etapa.

2 ESTADO DEL ARTE

La inteligencia artificial es la nueva electricidad.

- Andrew Ng-

El estado del arte en el ámbito de la detección de lenguaje de signos mediante IA ha avanzado significativamente en los últimos años, impulsado por los progresos en el aprendizaje automático y las redes neuronales. La integración de tecnologías como el reconocimiento de imágenes y el procesamiento en tiempo real ha permitido el desarrollo de sistemas más precisos y eficientes para la interpretación de gestos. Diversos estudios han explorado el uso de modelos de visión por computadora y aprendizaje profundo, destacando la efectividad de CNN y arquitecturas como YOLO en tareas de detección y clasificación de gestos. Sin embargo, aún persisten retos relacionados con la precisión, la capacidad de adaptación a diferentes contextos y la interpretación en condiciones no controladas, lo que subraya la necesidad de seguir investigando y mejorando estos sistemas para lograr una comunicación inclusiva y fluida entre personas sordas y oyentes.

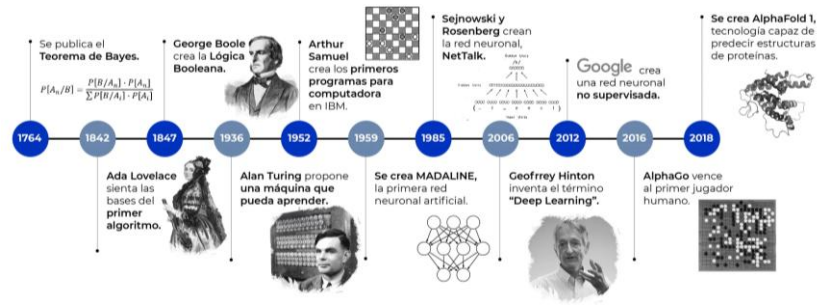

Figura 1 Historia de la IA [1]

2.1 IA,Machine Learning y Deep Learning

La IA [2] es el campo general de estudio que busca crear sistemas capaces de realizar tareas que normalmente requieren inteligencia humana. Esto incluye razonamiento, resolución de problemas, aprendizaje, percepción, comprensión del lenguaje natural, y toma de decisiones. Es un área amplia que abarca una variedad de enfoques y tecnologías, desde algoritmos simples hasta redes neuronales complejas.

Como se puede apreciar en la *Figura 2,* el ML (Machine Learning) es una subárea de la IA que se centra en el desarrollo de algoritmos que permiten a las máquinas aprender de los datos. En lugar de programar explícitamente cada paso, el modelo "aprende" patrones y toma decisiones a partir de ejemplos previos. En ML, el sistema mejora su rendimiento a medida que recibe más datos sin intervención humana directa.

El DL (Deep Learning) [3] es una subárea dentro del ML que utiliza redes neuronales profundas para modelar

problemas complejos. Las redes neuronales profundas [4] están formadas por múltiples capas de neuronas artificiales, lo que les permite aprender representaciones jerárquicas de los datos. Es particularmente eficaz en tareas que involucran grandes volúmenes de datos y patrones complejos, como el reconocimiento de imágenes, el procesamiento del lenguaje natural y la traducción automática.

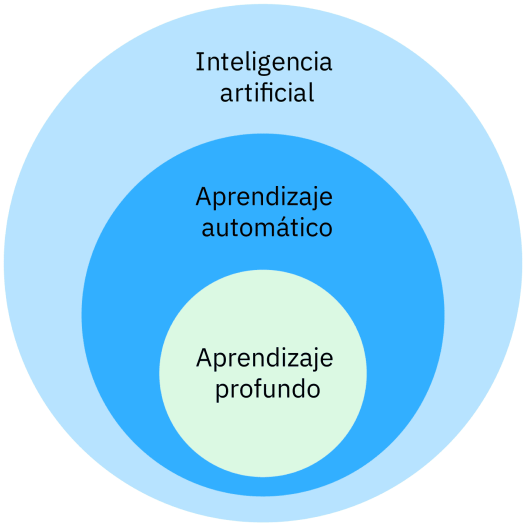

Figura 2 Inteligencia Artificial, Aprendizaje Automático, Aprendizaje profundo

[5]

2.2. Tipos de Aprendizaje

2.2.1. Aprendizaje Supervisado

En el aprendizaje supervisado, los humanos dan a un sistema de IA lo que se llama datos estructurados. Se trata de un conjunto de datos y cifras organizados en categorías ordenadas y etiquetadas.

Usando la forma en que se han estructurado los datos, el sistema de IA puede detectar patrones y usar esos patrones para predecir datos futuros.

Los datos estructurados también pueden tomar la forma de imágenes. Si, por ejemplo, los datos incluyen fotos de flores, algunas de las cuales son rosas y están etiquetadas como rosas, la máquina aprenderá cosas sobre la disposición de los píxeles en esas fotos. Más tarde, cuando vea una nueva foto de rosa y se le pregunte "¿Qué es esto?", responderá "rosa".

El sistema dará un valor de confianza para indicar, por ejemplo, que podría predecir con un 85 % de certeza que su respuesta es correcta. Cuantos más datos ingiera el sistema, mayor será su precisión.

2.2.2. Aprendizaje No Supervisado

A diferencia del aprendizaje supervisado, el aprendizaje no supervisado entrena una máquina con datos no etiquetados, como el texto de un libro. Entrenar un sistema de IA con datos no etiquetados es más difícil porque el sistema no puede hacer predicciones hasta que haya estructurado los propios datos. En el ejemplo del libro, estructurar los datos implicaría dividir el texto y encontrar relaciones entre palabras y frases. Así pues, los

algoritmos exploran los datos y tratan de encontrar una estructura.

2.2.3. Aprendizaje de Refuerzo

Con el aprendizaje de refuerzo, una máquina no recibe información específica que ingerir. Aprende mediante el sistema de ensayo y error. Los algoritmos de la máquina son recompensados cuando realiza una acción correcta y penalizados cuando no es así.

La máquina responde Sí o NO, y se asigna a sí misma un valor de confianza que está entre 100 % correcto y 100 % incorrecto.

Luego se le da nueva información que indica si esa respuesta es correcta o incorrecta. Aquí es donde tiene lugar el refuerzo. Por cada respuesta que es en gran medida incorrecta, la máquina es penalizada: sus algoritmos se ajustan y la imagen se vuelve a enviar para otro intento. Pero por cada respuesta que es en gran medida correcta, los algoritmos son recompensados.

Después de varias penalizaciones y recompensas, las respuestas de la máquina se vuelven más precisas y su valor de confianza aumenta.

Esto resulta particularmente útil cuando una máquina tiene que trabajar sobre un tipo específico de problema. La máquina toma decisiones continuamente hasta que alcanza un objetivo a largo plazo. Cada decisión que toma la máquina se basa en la decisión anterior.

2.3. Red Neuronal

Una red neuronal [3] es un modelo computacional inspirado en la estructura y el funcionamiento del cerebro humano. En una red neuronal artificial, las neuronas artificiales están organizadas en capas, y estas capas trabajan juntas para realizar tareas como clasificación, predicción, reconocimiento de patrones, etc.

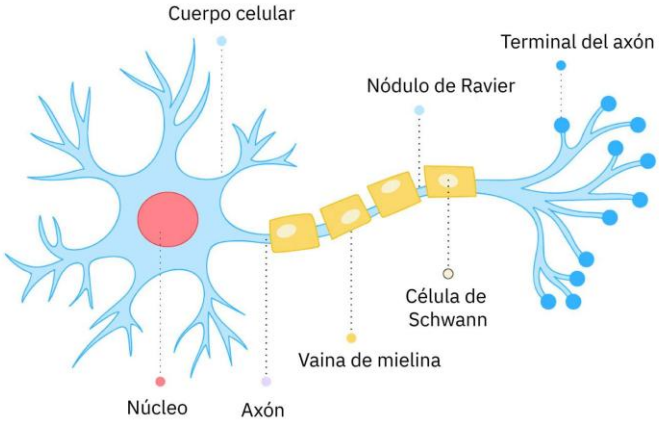

Figura 3 Neurona Humana

[5]

Las redes neuronales artificiales están inspiradas en el funcionamiento de las neuronas biológicas en el cerebro humano. A continuación, explicaremos este funcionamiento.

1. Neurona Biológica (*Figura 3*):
   - Las neuronas en el cerebro humano reciben señales de otras neuronas a través de conexiones llamadas sinapsis.
   - Cada señal recibida tiene una potencial de membrana que varía en función de la intensidad de la

señal.

- Cuando la señal alcanza un umbral específico, la neurona se activa y transmite una señal a otras neuronas, generando un impulso eléctrico que viaja por el axón.

2. Neurona Artificial (*Figura 4*):
   - En una red neuronal artificial, cada neurona artificial recibe entradas (características o valores) de otras neuronas o de los datos de entrada. Estas entradas se multiplican por pesos que determinan la importancia de cada entrada.
   - Las entradas ponderadas se suman y se pasan a través de una función de activación, que determina si la neurona se activará o no, similar al proceso de activación de las neuronas biológicas.
   - El resultado de esta activación se transmite a las neuronas de la siguiente capa.
3. Funcionamiento de una neurona artificial:
   - Entradas: Cada neurona recibe una serie de entradas, *x1, x2, ..., xn,* que representan características de los datos de entrada o de la capa anterior.
   - Pesos: Cada entrada tiene un peso asociado, *w1, w2, ..., wn,* que indica la importancia de esa entrada.
   - Suma ponderada: La neurona calcula una suma ponderada de todas las entradas:

$$SUMA = x1 \cdot w1 + x2 \cdot w2 + \cdots + xn \cdot wn + b$$

Donde b es un sesgo o bias que ayuda a ajustar el resultado.

- Función de activación: La suma ponderada pasa a través de una función de activación, como sigmoide, ReLU o tangente hiperbólica, que decide si la neurona debe activarse (es decir, pasar la señal a la siguiente capa) o no.
- Salida: El resultado de la activación es transmitido a la siguiente capa o como salida final de la red neuronal.

$$SALIDA = f(SUMA)$$

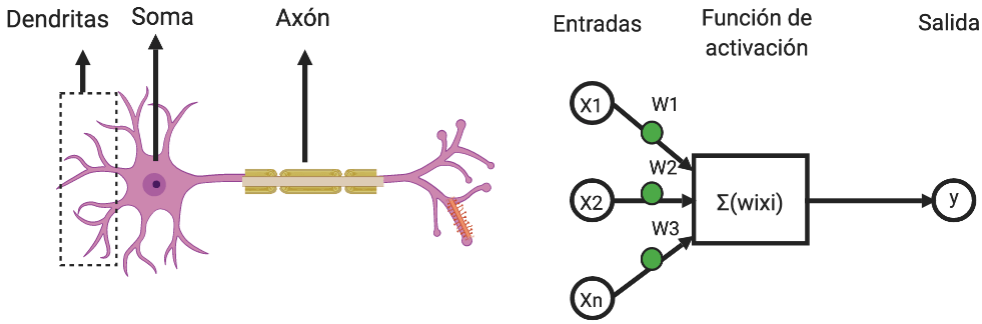

Figura 4 Neurona Humana vs Neurona Artificial

[5]

2.3.1. Estructura de una Red Neuronal

1. Capa de Entrada (Input Layer):
   - Esta es la primera capa de la red neuronal y se encarga de recibir los datos de entrada. Cada nodo

en esta capa representa una característica o atributo de los datos de entrada.

2. Capas Ocultas (Hidden Layers):
   - Las redes neuronales pueden tener una o más capas ocultas entre la capa de entrada y la capa de salida. Las neuronas en estas capas realizan transformaciones y combinaciones de las entradas para aprender representaciones complejas de los datos.
   - Cada capa oculta puede tener múltiples neuronas y el número de neuronas suele depender de la complejidad del problema.
   - Las neuronas de las capas ocultas realizan el procesamiento computacional. Estas capas permiten que la red aprenda patrones complejos.
3. Capa de Salida (Output Layer):
   - La capa de salida es la última capa de la red y produce el resultado final de la red neuronal. Dependiendo del tipo de problema, esta capa puede tener una o más neuronas.

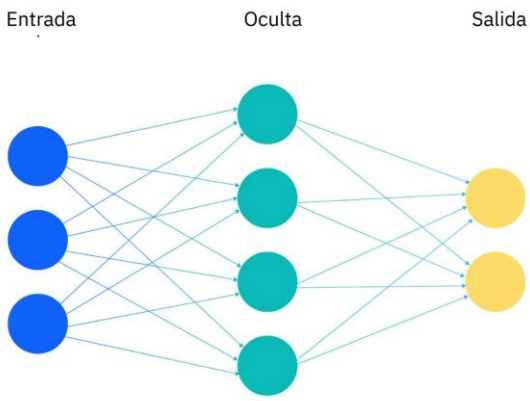

Figura 5 Estructura Red Neuronal

[5]

Cuando se apilan varias neuronas en una fila, constituyen una capa, y varias capas apiladas una al lado de la otra se denomina Red Neuronal Multicapa. En la *Figura 5* se puede observar la arquitectura típica de una red neuronal básica y en la *Figura 6* una red neuronal profunda.

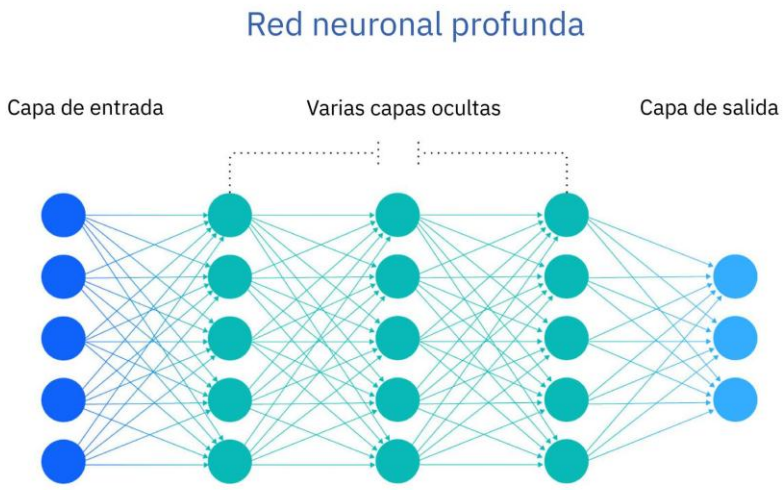

Figura 6 Estructura Red Neuronal Profunda Multicapa

[5]

2.4. Tipos de redes neuronales

2.4.1. SNN

Son las comentadas en el Apartado 2.3.1.

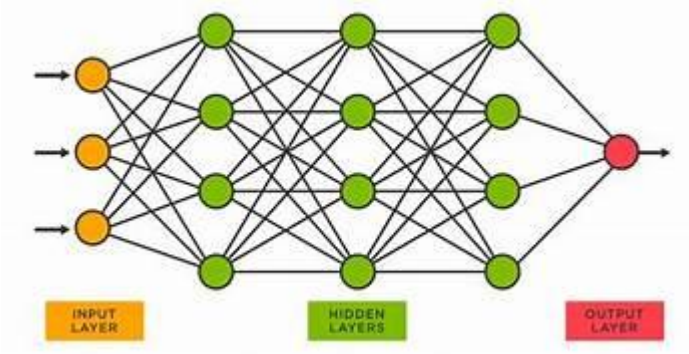

Figura 7 Red Neuronal Estándar

[6]

2.4.2. CNN

Se explicarán en detalle en el Capítulo 3 de este documento.

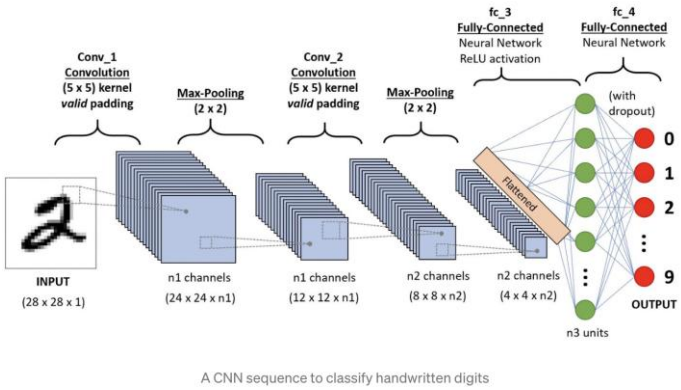

Figura 8 Red Neuronal Convolucional

[6]

2.4.3. Redes Neuronales Recurrentes

Las redes neuronales recurrentes (RNN) son un tipo de red neuronal profunda diseñada para procesar datos secuenciales o de series temporales, permitiendo la creación de modelos de machine learning capaces de realizar predicciones basadas en secuencias previas. A diferencia de las redes neuronales tradicionales, que consideran las entradas y salidas como independientes, las RNN incorporan una "memoria" que les permite utilizar la información de estados anteriores para influir en las predicciones actuales.

Al igual que las redes convolucionales (CNN), las RNN aprenden a partir de datos de entrenamiento. Sin embargo, su característica distintiva es la capacidad de mantener y utilizar información del pasado dentro de una secuencia. Esto es especialmente útil en tareas como el procesamiento de lenguaje natural, donde la posición y contexto de cada palabra afectan la predicción de la siguiente.

Otra particularidad de las RNN es que comparten parámetros a lo largo de cada capa de la red, a diferencia de las redes feedforward, en las que cada nodo tiene pesos independientes. Aun así, estos parámetros se ajustan mediante técnicas de retropropagación y descenso de gradiente para optimizar el aprendizaje.

El entrenamiento de las redes neuronales recurrentes se basa en el algoritmo de retropropagación a través del tiempo (BPTT), una extensión de la retropropagación tradicional adaptada a datos secuenciales. Este método permite calcular los gradientes considerando la influencia de estados anteriores, facilitando así el aprendizaje de patrones en secuencias de datos. En la *Figura 10* se pueden observar las diferencias entre una RNN y una SNN.

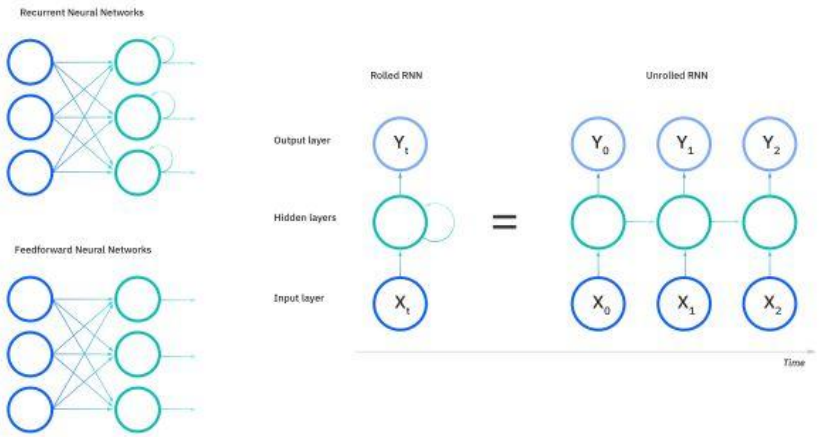

Figura 9 Diferencias RNN vs SNN

[5]

2.4.3.1. Tipos de RNN

2.4.3.1.1. Redes neuronales LSTM

Las redes neuronales LSTM son una variante de las redes neuronales recurrentes diseñada para abordar el problema del desvanecimiento y la explosión del gradiente. Estas redes utilizan unidades de memoria especializadas llamadas celdas LSTM que permiten capturar y mantener información a largo plazo.

En una red LSTM, cada unidad de memoria tiene una celda de memoria, tres compuertas y operaciones específicas para controlar el flujo de información. Estas compuertas permiten que la unidad de memoria decida qué información debe ingresarse, olvidarse y emitirse en cada paso de tiempo, lo que le da a la red la capacidad de mantener y actualizar estados a lo largo de la secuencia.

Las redes neuronales LSTM han demostrado ser especialmente efectivas en problemas que involucran dependencias a largo plazo, como el procesamiento del lenguaje natural y la generación de texto. Estas redes han logrado resultados sobresalientes y han superado las limitaciones de las RNN básicas en términos de aprendizaje de dependencias a largo plazo.

2.4.3.1.2. Redes neuronales GRU

Las redes neuronales GRU son otra variante de las redes neuronales recurrentes que también abordan el problema del desvanecimiento y la explosión del gradiente. Al igual que las LSTM, las GRU utilizan unidades de memoria especializadas llamadas unidades GRU que tienen compuertas para controlar el flujo de información.

A diferencia de las LSTM, las GRU tienen una estructura más simple con menos componentes. Cada unidad GRU tiene una compuerta de actualización y una compuerta de reinicio que permiten controlar la actualización de estados y el flujo de información.

Las redes neuronales GRU han demostrado ser efectivas en tareas de procesamiento del lenguaje natural, traducción automática y otras aplicaciones que involucran datos secuenciales. Aunque son más simples que las LSTM, las GRU han logrado resultados comparables y se han convertido en una alternativa popular en el campo del aprendizaje automático. En la *Figura 11* se puede observar la estructura de una red recurrente.

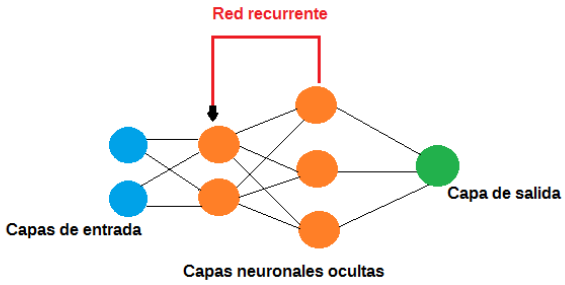

Figura 10 Estructura Red Recurrente

Imagen generada con IA

2.4.3.2. Aplicaciones de las redes neuronales recurrentes

2.4.3.2.1. NLP (Procesamiento del lenguaje natural)

El NLP es una de las áreas donde las redes neuronales recurrentes han tenido un gran impacto. Las RNN son especialmente efectivas en tareas como el análisis de sentimientos, la clasificación de texto, la generación de texto y la traducción automática.

En tareas de análisis de sentimientos, las RNN pueden capturar las dependencias contextuales y modelar la relación entre las palabras en un texto, lo que les permite clasificar el sentimiento asociado con una determinada

opinión o texto.

2.4.3.2.2. Traducción automática

La traducción automática es una aplicación clave de las redes neuronales recurrentes. Las RNN han demostrado ser muy eficaces para la traducción de texto entre diferentes idiomas.

En los sistemas de traducción automática basados en redes neuronales recurrentes, la entrada y la salida se representan como secuencias de palabras. La red neuronal recurrente se entrena para aprender las correspondencias entre las palabras en el idioma de origen y las palabras en el idioma de destino.

Las RNN capturan las dependencias y las estructuras sintácticas en las secuencias de palabras, lo que les permite generar traducciones precisas y coherentes.

2.4.3.2.3. Generación de texto

Las RNN pueden utilizarse para generar texto coherente y convincente en una variedad de dominios, como la escritura creativa, la generación de diálogos y la composición musical.

En la generación de texto, las RNN aprenden a modelar la distribución de probabilidad de las palabras en un corpus de entrenamiento y pueden generar nuevas secuencias de texto que se asemejan al estilo y contenido del corpus de origen. Estos modelos han sido utilizados para tareas como la generación automática de noticias, la escritura de poemas y la creación de diálogos de personajes en aplicaciones de inteligencia artificial.

Las redes neuronales recurrentes han demostrado ser herramientas poderosas para la generación de texto, y su capacidad para capturar estructuras y dependencias en los datos secuenciales ha llevado a avances significativos en esta área.

2.4.4. GAN

Imaginemos una competición entre un experto falsificador y un gran crítico de arte. El falsificador intenta crear réplicas de pinturas originales, mientras que el crítico se esfuerza por detectar qué obras son auténticas y cuáles son falsificaciones. Con el tiempo, el falsificador perfecciona su técnica, obligando al crítico a volverse más astuto en su análisis. Este ciclo de mejora continua lleva al falsificador a producir imitaciones casi indistinguibles de las originales.

Este es el principio detrás de las redes generativas antagónicas (GAN). En este modelo, el generador se encarga de crear nuevos datos, mientras que el discriminador evalúa su autenticidad. El generador busca producir datos lo suficientemente convincentes como para engañar al discriminador, mientras que este último aprende a diferenciar con mayor precisión entre datos reales y generados. Esta dinámica competitiva impulsa al generador a generar contenido cada vez más realista.

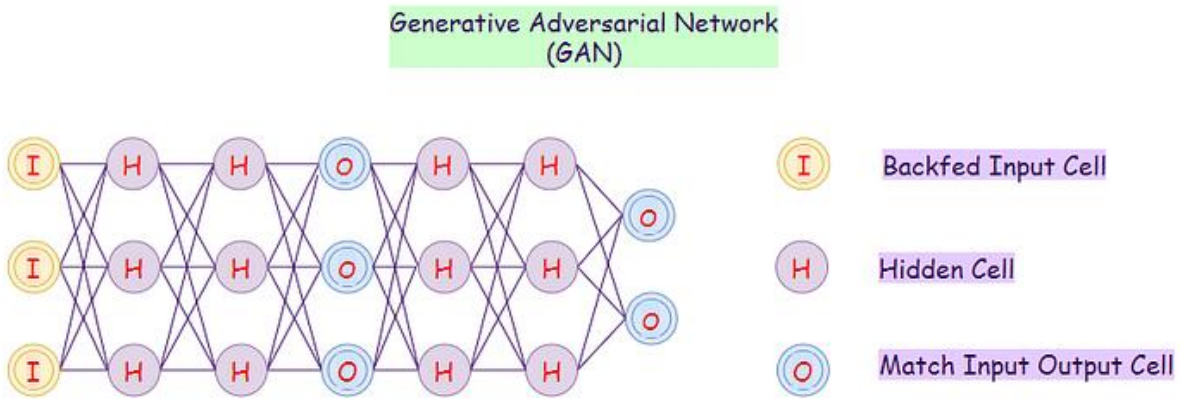

Figura 11 Red GAN

[5]

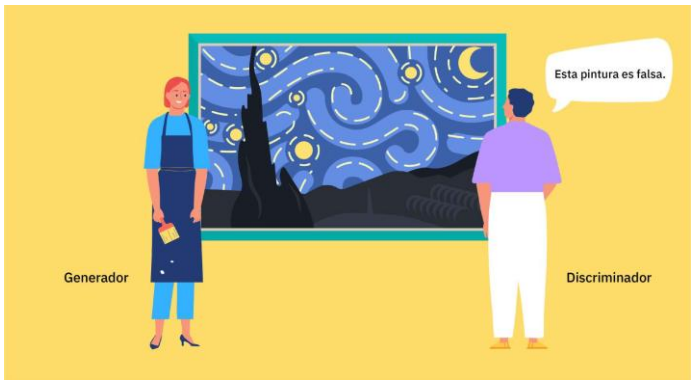

Figura 12 Ejemplo GAN

[5]

2.4.5. VAE

Pensemos en un artista que observa una pintura y crea un boceto simplificado que conserva sus elementos esenciales. Luego, basándose únicamente en ese boceto, reconstruye la pintura original con gran fidelidad. Este proceso le permite captar la esencia de la obra y replicarla con precisión.

Los modelos de Autoencoders Variacionales (VAE) funcionan de manera similar. Una red "codificadora" reduce los datos de entrada a una representación de menor dimensión, capturando su estructura fundamental. Luego, una red "decodificadora" utiliza esta representación comprimida para reconstruir los datos originales. Gracias a este mecanismo, los VAE pueden aprender patrones subyacentes en los datos y generar nuevas muestras con características similares.

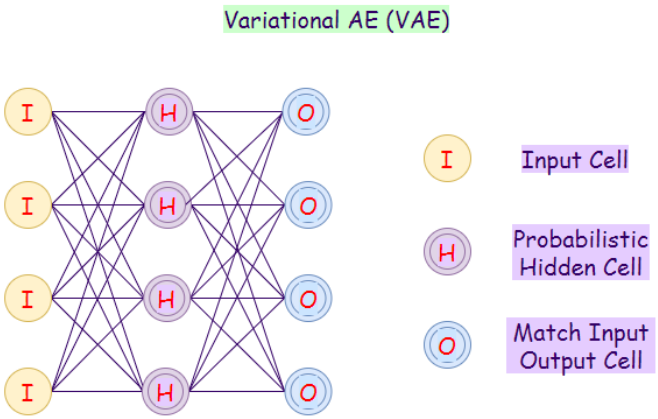

Figura 13 Red VAE

[5]

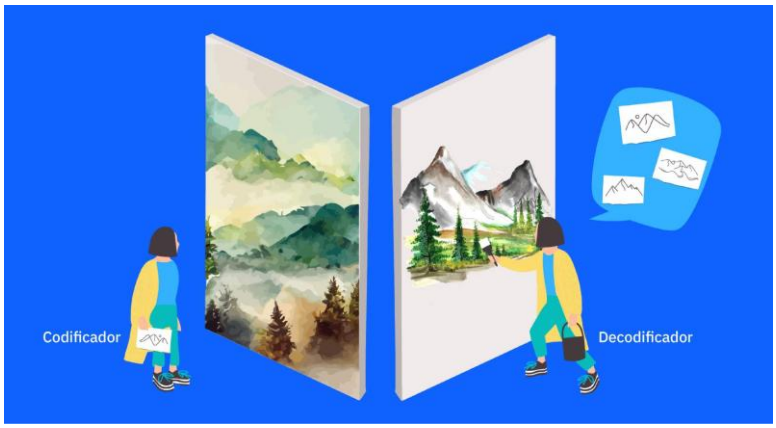

Figura 14 Ejemplo Red VAE

[5]

2.4.6. Autoregresión

Imaginemos un narrador experto que escucha el inicio de una historia y, basándose en los eventos y palabras previas, la continúa de manera fluida y envolvente. Para hacerlo, utiliza su conocimiento del lenguaje, la gramática y las estructuras narrativas, asegurándose de que la historia mantenga coherencia y atractivo a medida que avanza.

Los modelos autorregresivos funcionan de manera similar: generan contenido nuevo prediciendo cada elemento de una secuencia a partir de los anteriores. Son especialmente eficaces en la generación de texto, ya que modelan las probabilidades condicionales de palabras y caracteres dentro de una oración, permitiendo producir secuencias coherentes y naturales.

Figura 15 Ejemplo Autoregresión

[5]

2.4.7. Transformers

Los Transformers aparecieron como una novedosa arquitectura de DL para NLP en [7] que presentaba unos ingeniosos métodos para poder realizar traducción de un idioma a otro superando a las redes seq-2-seq LSTM de aquel entonces.

Esta arquitectura surge como una solución a problemas de aprendizaje supervisado en NLP, obteniendo grandes ventajas frente a los modelos utilizados en ese entonces. El transformer permitía realizar la traducción de un

idioma a otro con la gran ventaja de poder entrenar al modelo en paralelo; lo que aumentaba drásticamente la velocidad y reducción del coste; y utilizando como potenciador el mecanismo de atención, que hasta ese momento no había sido explotado del todo.

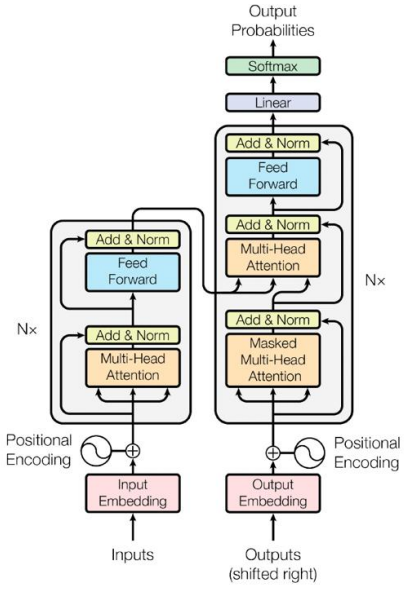

Figura 16 Transformers

[7]

La entrada pasará por una serie de Encoders que se encadenan uno tras otro y luego envían su salida a otra serie de Decoders hasta emitir la salida final.

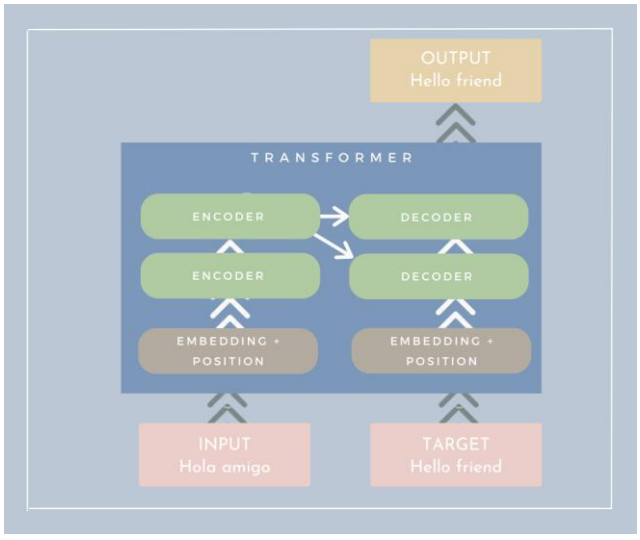

Figura 17 Estructura Transformer

Imagen generada con IA

2.5. Algoritmos de ajustes de datos utilizados en IA

2.5.1. Regresión Lineal

La regresión lineal se refiere a datos que pueden representarse gráficamente como una línea recta. Por ejemplo, una empresa podría pensar que un mayor gasto en publicidad conduce a un aumento de las ventas. Esto podría representarse gráficamente como una serie de puntos que forman una línea recta ascendente, como se muestra en la *Figura 19.*

Como sugiere el gráfico, a medida que aumenta la publicidad, también lo hacen las ventas. Hay muchos resultados posibles (diferentes cantidades de publicidad conducen a diferentes cantidades de ventas), pero el cambio sube en el gráfico en línea recta.

La situación se complica si las ventas reales de una empresa muestran datos diferentes para productos diferentes, en lugares diferentes, en fechas diferentes, etc. Con un gran número de variables e instancias, el gráfico se convierte en una masa de puntos que no forman una línea recta. Si no se ajusta, el gráfico resultante es demasiado general para ayudar a una empresa a tomar una buena decisión. Ahí es donde la regresión lineal puede ayudar. La regresión lineal puede aprender todas las variables y luego calcular una predicción razonablemente precisa de cómo influirá la publicidad en las ventas en un momento y lugar determinados del futuro. En efecto, la regresión lineal resuelve la masa de puntos en una línea "más probable" que puede utilizarse para una predicción simple.

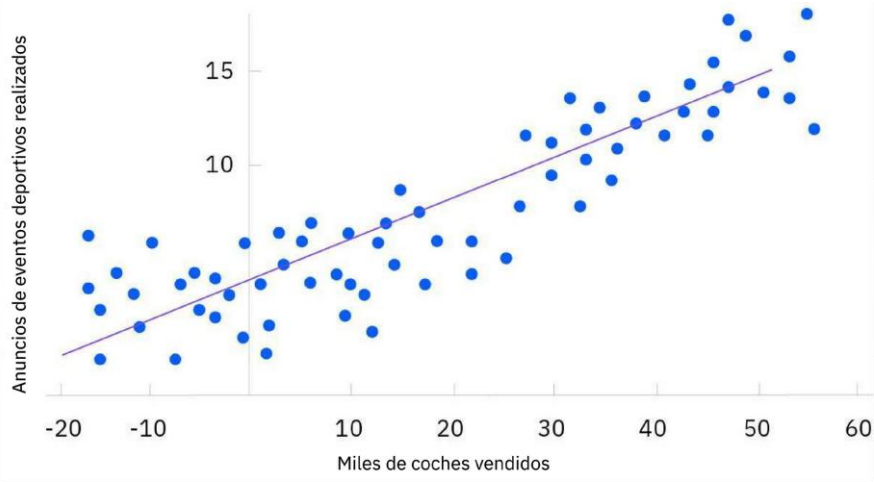

Figura 18 Regresión Lineal

[5]

2.5.2. Regresión logística o Sigmoide

En algunas situaciones, una relación no se desarrolla en línea recta. A veces, un sistema utiliza valores que requieren un tipo de resultado específico y limitado, como algo entre 0 y 1 (o NO y SÍ). En esta situación, un gráfico puede formar lo que se denomina una función sigmoidea, o una curva en forma de S. Para cualquier conjunto de variables, el resultado (que es un punto de la curva S) está comprendido entre 0 y 1. En la *Figura 20* se puede observar el funcionamiento de esta función.

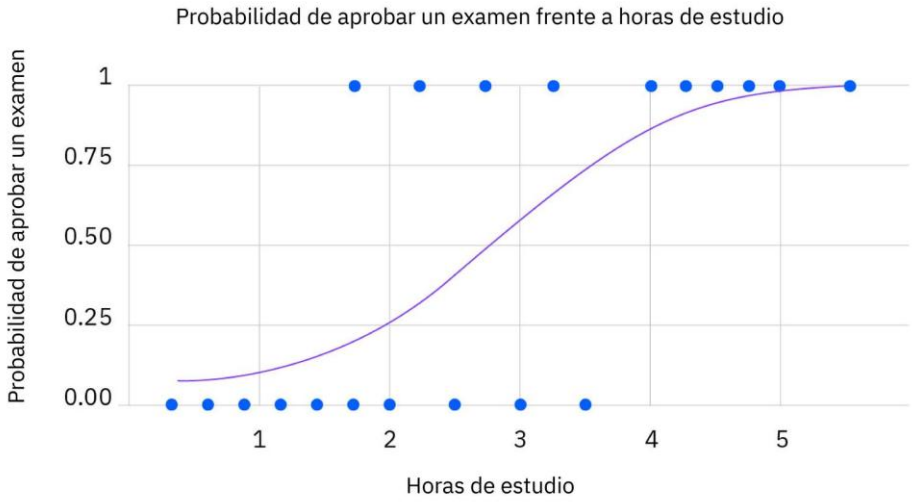

Figura 19 Sigmoide

[5]

2.5.3. Árbol de decisión

Un árbol de decisión es un algoritmo de aprendizaje supervisado. Funciona como un diagrama de flujo. Un diagrama de flujo es como un árbol de decisión invertido. El diagrama de flujo tiene un nodo raíz (donde empieza el diagrama), ramas que conectan con nodos internos y más ramas que conectan con nodos hoja.

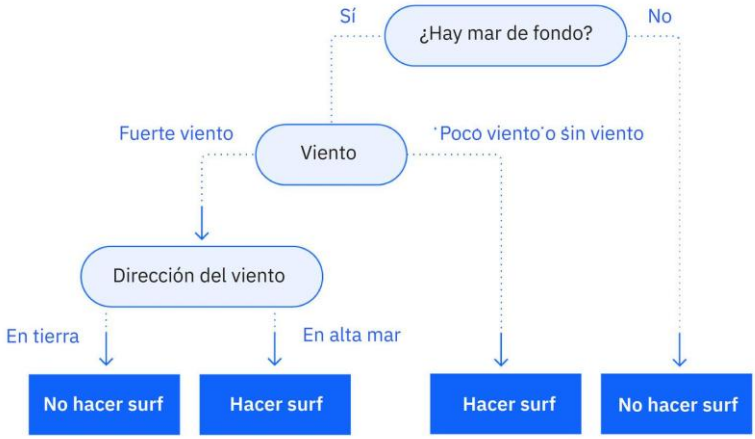

Figura 20 Árbol de Decisión

[5]

2.6. Entrenamiento de Redes Neuronales

2.6.1. Forward Propagation

La propagación hacia adelante es el proceso en una red neuronal en el que los datos de entrada pasan a través de las capas de la red para generar una salida. Implica los siguientes pasos:

1. Capa de entrada: los datos de entrada se introducen en la capa de entrada de la red neuronal.
2. Capas ocultas: los datos de entrada se procesan a través de una o más capas ocultas. Cada neurona de una capa oculta recibe entradas de la capa anterior, aplica una función de activación a la suma ponderada de estas entradas y pasa el resultado a la siguiente capa.
3. Capa de salida: los datos procesados pasan por la capa de salida, donde se genera el resultado final de la red. La capa de salida normalmente aplica una función de activación adecuada para la tarea, como softmax para la clasificación o activación lineal para la regresión.
4. Predicción: La salida final de la red es el resultado de la predicción o clasificación de los datos de entrada.

La propagación hacia adelante es esencial para hacer predicciones en redes neuronales. Calcula la salida de la red para una entrada determinada en función de los valores actuales de los pesos y sesgos. Luego, la salida se compara con el valor objetivo real para calcular la pérdida, que se utiliza para actualizar los pesos y sesgos durante el proceso de entrenamiento.

Para calcular las pérdidas, se utiliza la función de pérdidas L, cuya fórmula matemática es la siguiente:

$$L(\hat{y}, y) = -(y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y}))$$

y: resultado esperado

$\hat{y}$: resultado real

Esta fórmula es la más utilizada en clasificación, en problemas de regresión no se utiliza.

Esto mide las pérdidas para un solo ejemplo, para medir el rendimiento real del conjunto completo del entrenamiento se utiliza la función de costo J, cuya formula matemática es la que sigue:

$$J(w, b) = \frac{1}{m} \cdot \sum_{i=1}^{m} L(\hat{y}(i), y(i))$$

m: número de ejemplos en el conjunto de datos

w: pesos del modelo

b: sesgo del modelo

$L(\hat{y},y)$: función de pérdidas

El objetivo final es encontrar los valores de w y b que minimicen la función de costo. De esto se encarga la fase de propagación hacia atrás.

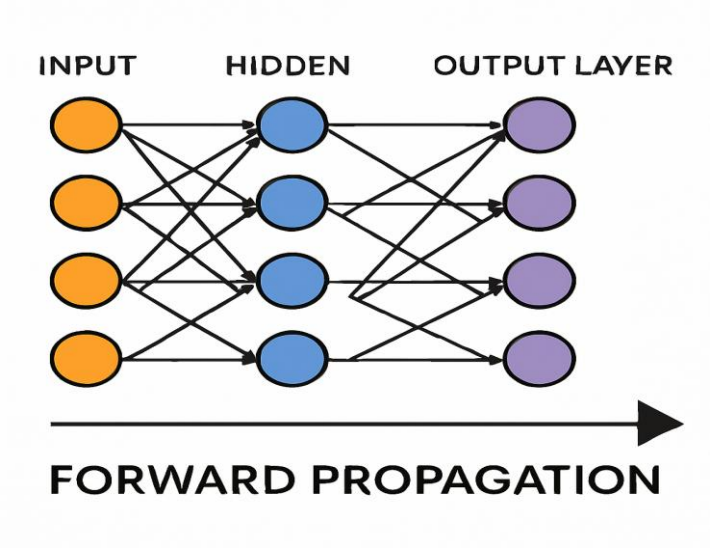

Figura 21 Forward Propagation

Imagen generada con IA

2.6.2. Descenso de Gradiente

Se trata de un algoritmo de optimización y el método con el que las redes neuronales consiguen aprender, basado en alcanzar el punto mínimo de un error. Además de utilizarse para redes neuronales, también es compatible con cálculos de regresión lineal, regresión logística o máquinas de vectores de soporte, haciendo que sean más complejos.

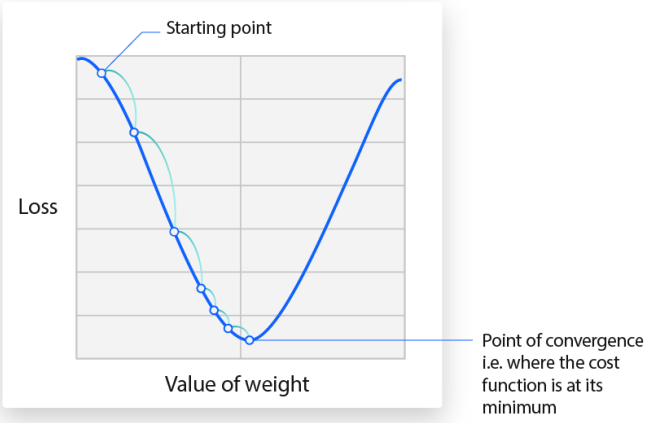

Figura 22 Descenso de Gradiente

[5]

La tasa de aprendizaje determina el tamaño de los pasos que toma el algoritmo de descenso de gradiente en dirección al mínimo local, afectando directamente la velocidad con la que se aproxima a los valores óptimos de la función de coste.

Por esta razón, la elección de la tasa de aprendizaje, γ, es crucial y tiene un impacto significativo en la efectividad del algoritmo. Si la tasa es demasiado alta, el algoritmo dará saltos grandes en su intento de alcanzar el punto óptimo, oscilando de un lado a otro sin converger adecuadamente, lo que puede incluso empeorar la función de coste.

En contraste, si la tasa de aprendizaje es demasiado baja, el algoritmo avanzará de manera muy lenta, con pasos diminutos que garantizan precisión, pero a costa de una optimización extremadamente lenta. Encontrar un equilibrio adecuado es clave para garantizar una convergencia eficiente y efectiva como indica la *Figura 23*.

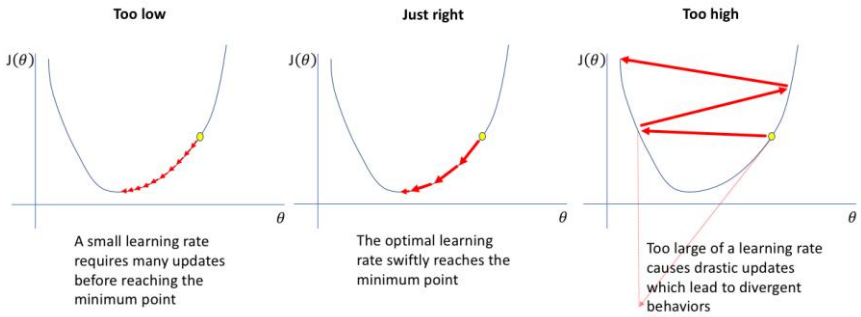

Figura 23 Tasa de aprendizaje

[8]

2.6.3. Back Propagation

Este proceso se aplica en múltiples capas de una red neuronal. El algoritmo avanza desde la capa de entrada hasta la capa de salida, donde se calcula la diferencia entre el resultado obtenido y el resultado esperado. Luego, la señal de error se propaga en sentido inverso, desde la capa de salida hacia las capas ocultas, ajustando los pesos de las neuronas artificiales en cada paso. Esto se puede apreciar en la *Figura 24*.

A medida que el margen de error disminuye, la red mejora su capacidad de predicción. La función sigmoide permite visualizar esta evolución, mostrando cómo el error se reduce progresivamente durante el entrenamiento.

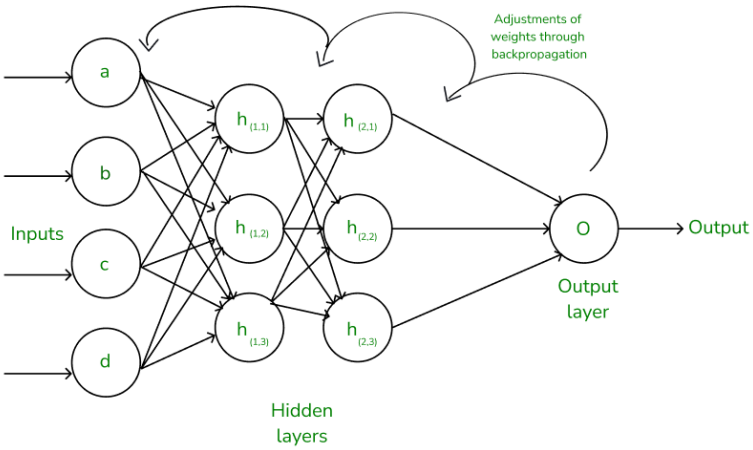

Figura 24 Back Propagation

[9]

2.7. Problemas comunes de entrenamiento y soluciones

2.7.1. Sobreajuste (Overfitting)

El sobreajuste [10] es un fenómeno clave en el aprendizaje automático que ocurre cuando un modelo se ajusta excesivamente a los datos de entrenamiento, afectando su capacidad de generalización a nuevos datos.

El propósito del aprendizaje automático es identificar patrones en los datos de entrenamiento para realizar predicciones precisas sobre información no vista. Para lograrlo, es fundamental extraer patrones generales en lugar de memorizar detalles específicos. En este sentido, el sobreajuste se asemeja a cuando una persona memoriza información sin comprender realmente el concepto subyacente.

Este problema suele aparecer cuando un modelo se entrena demasiado o con datos atípicos, lo que lleva al algoritmo a aprender patrones irrelevantes en lugar de captar tendencias generales. Los modelos más complejos son más propensos a sobreajustarse en comparación con modelos más simples. Además, si la cantidad de datos disponibles es limitada, aumenta la probabilidad de que el modelo se ajuste demasiado a ellos, reduciendo su capacidad de generalización.

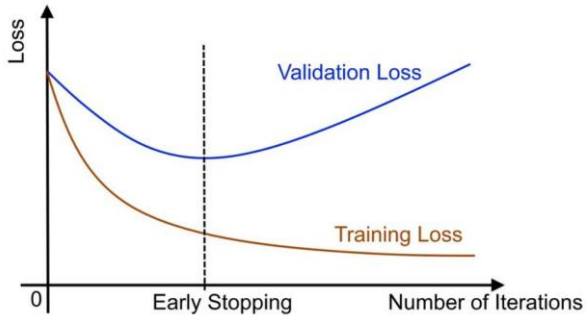

Figura 25 Sobreajuste

Imagen generada con IA

2.7.1.1. Soluciones

2.7.1.1.1. Regularización

Las técnicas de regularización, como L1 y L2, ayudan a prevenir el sobreajuste en modelos de aprendizaje automático al penalizar coeficientes grandes y controlar la complejidad del modelo.

La regularización L1 añade una penalización basada en el valor absoluto de los coeficientes de las características. Esto fomenta la escasez al forzar algunos coeficientes a cero, lo que, en la práctica, equivale a realizar una selección automática de características, eliminando aquellas menos relevantes. Esta propiedad hace que L1 sea especialmente útil en conjuntos de datos con muchas características, ya que simplifica el modelo al enfocarse solo en las más importantes, reduciendo así el sobreajuste.

Por su parte, la regularización L2 introduce una penalización proporcional al cuadrado de los coeficientes. En este caso, los coeficientes no se anulan completamente, sino que se reducen de manera uniforme, lo que permite que todas las características contribuyan al modelo en cierta medida. L2 es especialmente eficaz para manejar la multicolinealidad y suele producir modelos más estables y menos dispersos en comparación con L1.

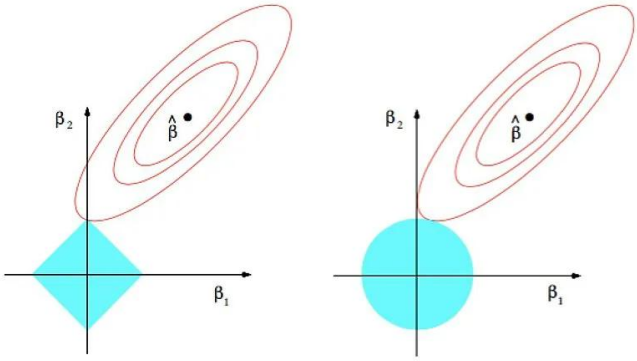

Figura 26 Regularización L1 y L2

Imagen generada por IA

2.7.1.1.2. Data Augmentation

El aumento de datos es el proceso de generar artificialmente nuevos datos a partir de datos existentes, principalmente para entrenar nuevos modelos de ML. Los modelos de ML requieren conjuntos de datos grandes y variados para el entrenamiento inicial, pero obtener conjuntos de datos del mundo real suficientemente diversos puede ser un desafío debido a la limitación de datos, las regulaciones y otras limitaciones. El aumento de datos aumenta artificialmente el conjunto de datos al realizar pequeños cambios en los datos originales.

Los modelos de DL se basan en grandes volúmenes de datos diversos para desarrollar predicciones precisas en diversos contextos. El aumento de datos complementa la creación de variaciones de datos que pueden ayudar a un modelo a mejorar la precisión de sus predicciones. Los datos aumentados son fundamentales en el entrenamiento.

La primera etapa del aumento de datos consiste en analizar un conjunto de datos existente y comprender sus características. Características como el tamaño de las imágenes de entrada, la distribución de los datos o la estructura del texto proporcionan un contexto adicional para el aumento.

Podemos seleccionar diferentes técnicas de aumento de datos en función del tipo de datos y de los resultados deseados. Por ejemplo, aumentar un conjunto de datos con muchas imágenes incluye agregarles ruido, escalarlas o recortarlas. Alternativamente, aumentar un conjunto de datos de texto para el NLP reemplaza los sinónimos o parafrasear fragmentos.

Una vez que hayamos seleccionado la técnica de aumento de datos que funcione mejor para el objetivo deseado, se podrán aplicar diferentes transformaciones. Las muestras de imágenes del dataset se transforman mediante el método de aumento seleccionado, lo que proporciona una gama de nuevas muestras aumentadas. Se pueden aplicar todas las descritas en la *Figura 27*.

Durante el proceso de aumento, se mantienen las mismas reglas de etiquetado para garantizar la coherencia de datos, lo que garantiza que los datos sintéticos incluyan las mismas etiquetas correspondientes a los datos de origen.

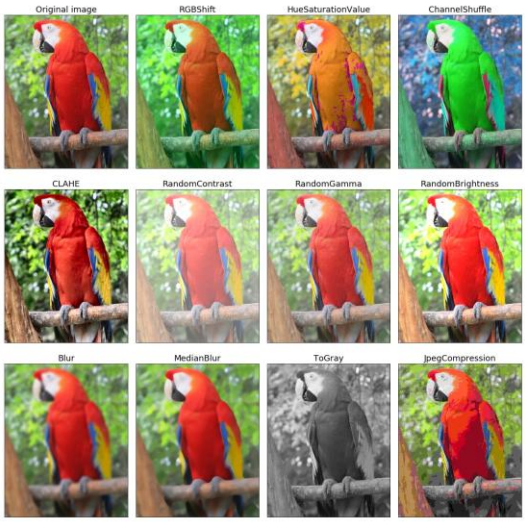

Figura 27 Tipos de aumento de imágenes

[11]

La IA generativa es esencial en el aumento de datos porque facilita la producción de datos sintéticos. Ayuda a aumentar la diversidad de datos, optimizar la creación de datos realistas y preservar la privacidad de los datos.

Para ello se utilizan redes GAN y VAE ya comentadas en este documento con anterioridad.

2.7.1.1.3. Detención temprana

La detención temprana es una técnica fundamental en el entrenamiento de modelos de aprendizaje profundo, diseñada para prevenir el sobreajuste y optimizar el rendimiento en conjuntos de datos no vistos. Esta estrategia permite observar el rendimiento del modelo en un conjunto de validación durante el proceso de entrenamiento, permitiendo que el proceso se interrumpa una vez que se detecta que la precisión del modelo deja de mejorar. Esto no solo ahorra recursos computacionales, sino que también ayuda a obtener modelos más generalizables.

2.7.2. Desvanecimiento de Gradiente

Un gradiente de error es la dirección y magnitud calculada durante el entrenamiento de una red neuronal que se utiliza para actualizar los pesos de la red en la dirección correcta y en la cantidad correcta.

Si la función de activación elegida es la función sigmoide, una de las más comunes, debido a que su derivada siempre está acotada entre 0 y 0.25, cuando tenemos una red con muchas capas el valor de gradiente cada vez es más cercano a 0 ya que estamos multiplicando muchas veces un valor pequeño. Debido a este problema, las primeras capas de una red neuronal son las más lentas y difíciles de entrenar ya que el valor del gradiente que se usa para actualizarlas en cada iteración del entrenamiento es muy pequeño. Y esto causa otro problema, que, si las primeras capas no están bien entrenadas, el problema se arrastra a las capas posteriores.

2.7.2.1. Soluciones

2.7.2.1.1. Función ReLu

Como hemos comentado, la derivada de la función sigmoide es menor o igual que 0.25, pero si cogemos como función de activación la función ReLU, cuya derivada es 1 por encima de 0, podríamos obtener mejores soluciones.

2.7.2.1.2. Uso de LSTMs

Para el caso de las RNN, la solución pasa por usar LSTMs, donde cada nodo o neurona es una célula de memoria. De esta forma la red es capaz de retener información de entradas anteriores en el tiempo, “almacenándolas en memoria” y así poder tener en cuenta dependencias temporales largas.

2.7.3. Explosión de Gradiente

En redes profundas o RNN, los errores de gradiente pueden acumularse durante una actualización y dar como resultado gradientes muy grandes. Estos, a su vez, dan como resultado grandes actualizaciones de los pesos de la red y, a su vez, una red inestable. En caso extremo, los valores de los pesos pueden volverse tan grandes que se desbordan y dan como resultado valores NaN.

La explosión ocurre a través del crecimiento exponencial al multiplicar repetidamente los gradientes a través de las capas de la red que tienen valores mayores que 1.

2.7.3.1. Soluciones

2.7.3.1.1. Uso de LSTMs

En las redes neuronales recurrentes, puede producirse una explosión de gradiente debido a la inestabilidad inherente al entrenamiento de este tipo de red, por ejemplo, mediante la retropropagación a través del tiempo, que esencialmente transforma la red recurrente en una red neuronal de perceptrón multicapa profunda.

La explosión de gradientes se puede reducir mediante el uso de LSTM y, tal vez, estructuras neuronales de tipo cerrado relacionadas.

La adopción de unidades de memoria LSTM es una nueva práctica recomendada para las redes neuronales recurrentes para la predicción de secuencias.

2.7.3.1.2. Gradient Clipping

Las explosions de gradiente aún pueden ocurrir en redes de perceptrones multicapa muy profundas con un gran tamaño de batch y LSTM con longitudes de secuencia de entrada muy largas.

Si aún se producen explosiones de gradientes, se puede verificar y limitar el tamaño de los gradientes durante el entrenamiento de su red.

Esto se denomina recorte de gradiente.

Específicamente, los valores del gradiente de error se comparan con un valor umbral y se recortan o se establecen en ese valor umbral si el gradiente de error excede el umbral.

2.7.3.1.3. Regularización de pesos

Utilizar las tecnicas de regularización L1 y L2 ya comentadas con anterioridad.

2.8. Transfer Learning

El transfer learning [12] se basa en el concepto de reutilización de elementos de los modelos de ML preentrenados en nuevos modelos que se utilizarán para fines similares. Esto permite optimizar los recursos y los datos etiquetados necesarios para el entrenamiento.

El transfer learning consiste en reutilizar los elementos de un modelo de ML previamente entrenado en un nuevo modelo. Si ambos se desarrollan para realizar tareas similares, entonces el conocimiento se puede compartir entre ellos.

Este enfoque reduce los recursos utilizados y la cantidad de datos etiquetados necesarios para entrenar a los nuevos modelos, por lo que se está convirtiendo en una parte importante de la evolución del machine learning, utilizándose cada vez más como técnica en el proceso de desarrollo.

El transfer learning se usa generalmente cuando el entrenamiento de un modelo de aprendizaje automático para resolver una nueva tarea requiere muchos recursos. El proceso reutiliza partes relevantes de un modelo de aprendizaje automático existente para solventar un problema nuevo similar.

El concepto clave del transfer learning es la generalización, es decir, solo se transfiere el conocimiento que puede ser utilizado por otro modelo en diferentes escenarios o condiciones. En lugar de que los modelos estén ligados a un conjunto de datos de entrenamiento, los modelos utilizados en el aprendizaje por transferencia son más

generalizados. Los modelos desarrollados de esta manera se pueden utilizar en condiciones cambiantes y con diferentes conjuntos de datos.

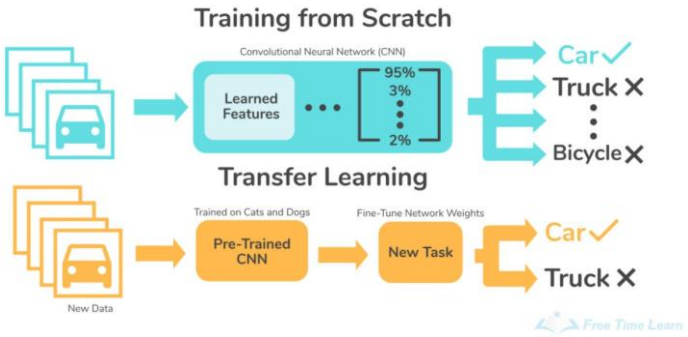

Figura 28 Transfer Learning

[13]

3 CNNs

En este capítulo se explicarán en profundidad las redes neuronales convolucionales. Se detallará su arquitectura y se explicarán sus funciones de activación. Este apartado es importante detallarlo aparte de las redes neuronales, ya que es la que hemos utilizado para nuestro proyecto.

3.1. Historia

Las CNNs son NN ampliamente utilizadas que se inspiran en el mecanismo de percepción visual de los cuerpos vivos. Su historia comienza a mediados de los noventa y se desarrolló rápidamente a finales del siglo XX.

En 1990, algunos autores publicaron un artículo en el que desarrollaron una red neuronal artificial llamada LENET-5, que era multicapa. Fue construida para clasificar dígitos escritos a mano. Era capaz de reconocer patrones visuales directamente a partir de píxeles con poco o ningún preprocesamiento.

Alrededor de 2010, los investigadores propusieron una arquitectura CNN conocida como Alex Net, que era similar a LeNet-5 pero con una estructura más profunda. Después del éxito de Alex Net, se propusieron muchas otras arquitecturas como ZF Net, VGG Net, Google Net y Res Net.

Las tendencias en la evolución de las arquitecturas CNN muestran que las redes son cada vez más profundas. La red puede aprender la variable objetivo con mayor precisión y obtener mejores representaciones de características debido a una arquitectura más profunda.

3.2. Aplicaciones y usos de las CNN

Reconocimiento de imágenes	Las CNN permiten clasificar imágenes en categorías, de esta forma se pueden etiquetar fotos o en aplicaciones de búsqueda visual de una manera más sencilla.
Detección de objetos	Otra de las aplicaciones de estas redes es la de identificar y localizar objetos específicos dentro de una imagen. Esto se utiliza en tecnologías de vehículos autónomos, vigilancia y sistemas de seguridad.
Reconocimiento facial	Son muy usadas en aplicaciones de autenticación biométrica, seguridad y redes sociales con el objetivo de identificar y verificar personas a través de sus rostros.
Diagnóstico médico	Gracias a las CNN, es posible la interpretación de imágenes médicas, como radiografías, resonancias magnéticas y tomografías, detectando antes enfermedades y anomalías

Tabla 1 Usos de las CNNs

3.3. Arquitectura

Las redes neuronales convolucionales (CNN) [14] son modelos de aprendizaje automático diseñados para procesar datos con estructura espacial, como imágenes y señales de audio, inspirándose en el sistema visual humano.

Una CNN es un tipo de red neuronal artificial compuesta por múltiples capas organizadas jerárquicamente. Cada capa extrae características progresivamente más abstractas de los datos de entrada, permitiendo un análisis profundo de los patrones presentes en la información.

El principio fundamental de las CNN es la convolución, una operación matemática que combina los datos de entrada con filtros o kernels para detectar características relevantes. Estos filtros recorren la imagen o señal, realizando operaciones de multiplicación y suma en cada posición para identificar patrones y estructuras clave.

Comprender la arquitectura de una CNN es esencial para entender su capacidad en el procesamiento de datos. A continuación, se explorarán las diferentes capas que componen una CNN y su función dentro del proceso de aprendizaje automático. La arquitectura típica es la comentada en la *Figura 29*.

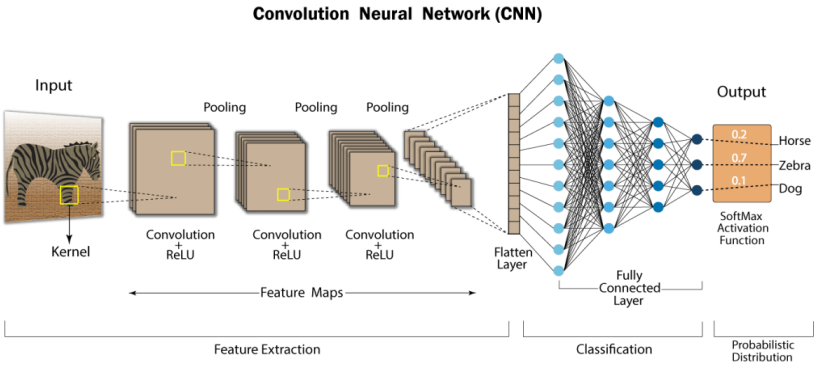

Figura 29 CNN

[15]

3.3.1. Tipos de capas

3.3.1.1. Capas de entrada

La capa de entrada es la primera capa de una red neuronal convolucional. En esta capa, se introduce la información de entrada, que suele ser una imagen en el caso de aplicaciones de visión por computadora. La imagen se representa mediante una matriz de píxeles, donde cada píxel contiene información sobre el color y la intensidad de la imagen

3.3.1.2. Capas convolucionales

La capa convolucional [15] es el bloque de creación principal de una CNN, y es donde se realizan la mayoría de los cálculos. Requiere algunos componentes, como datos de entrada, un filtro y un mapa de características.

Supongamos que la entrada es una imagen en color compuesta por una matriz de píxeles en 3D. Esto significa que la entrada tendrá tres dimensiones: altura, anchura y profundidad, que corresponden a la composición RGB en una imagen. También hay un detector de características, conocido como kernel o filtro, que se mueve por los campos receptivos de la imagen para comprobar si la característica está presente. Este proceso se denomina convolución.

El detector de características es una matriz bidimensional (2D) de pesos que representa una parte de la imagen. Aunque su tamaño puede variar, el tamaño del filtro suele ser una matriz de 3x3; esto también determina el tamaño del campo receptivo.

A continuación, el filtro se aplica a un área de la imagen y se calcula un producto escalar entre los píxeles de entrada y el filtro. Este producto escalar se introduce en una matriz de salida. Después, el filtro se desplaza un poco y repite el proceso hasta que el kernel haya recorrido toda la imagen. El resultado final de la serie de productos escalares de la entrada y el filtro se conoce como mapa de características, mapa de activación o característica convolucionada.

Los pesos del detector de características permanecen fijos a medida que se desplaza por la imagen, lo que también se conoce como compartición de parámetros. Algunos parámetros, como los valores de peso, se ajustan durante el entrenamiento a través del proceso de retropropagación y descenso del gradiente. Sin embargo, hay tres hiperparámetros que afectan al tamaño del volumen de la salida y que deben configurarse antes de que comience el entrenamiento de la red neuronal. Entre ellos se incluyen:

1. El número de filtros afecta a la profundidad de la salida. Por ejemplo, tres filtros distintos producirían tres mapas de entidades diferentes, lo que crearía una profundidad de tres.
2. El stride es la distancia, o el número de píxeles, que el kernel mueve sobre la matriz de entrada.

Aunque los valores de stride iguales o superiores a dos son poco frecuentes, un stride mayor produce una salida menor.

3. El zero-padding suele utilizarse cuando los filtros no se ajustan a la imagen de entrada. Esto pone a cero todos los elementos que quedan fuera de la matriz de entrada, produciendo una salida de mayor o igual tamaño. Existen tres tipos de padding:
   - Valid padding: también conocido como no padding. En este caso, la última convolución se descarta si las dimensiones no se alinean.
   - Same padding: este garantiza que la capa de salida tenga el mismo tamaño que la capa de entrada.
   - Full padding: este tipo de padding aumenta el tamaño de la salida añadiendo ceros al borde de la entrada.

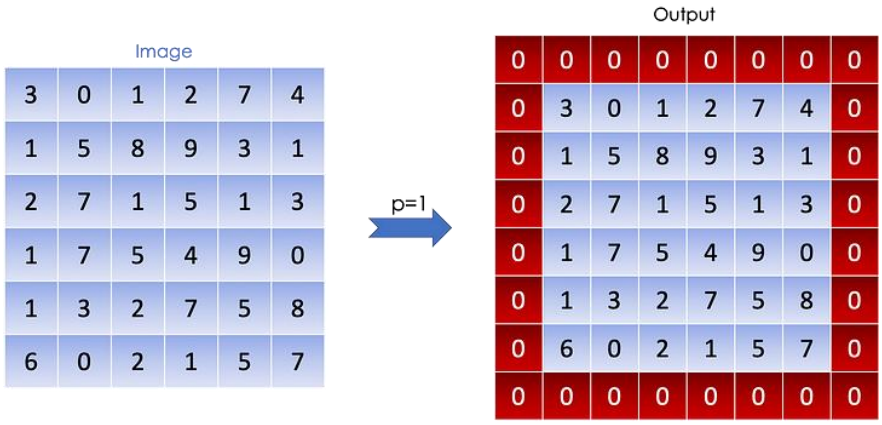

Figura 30 Padding

[5]

Después de cada operación de convolución, la CNN aplica una transformación de unidad lineal rectificada (ReLU) al mapa de características, introduciendo la no linealidad en el modelo.

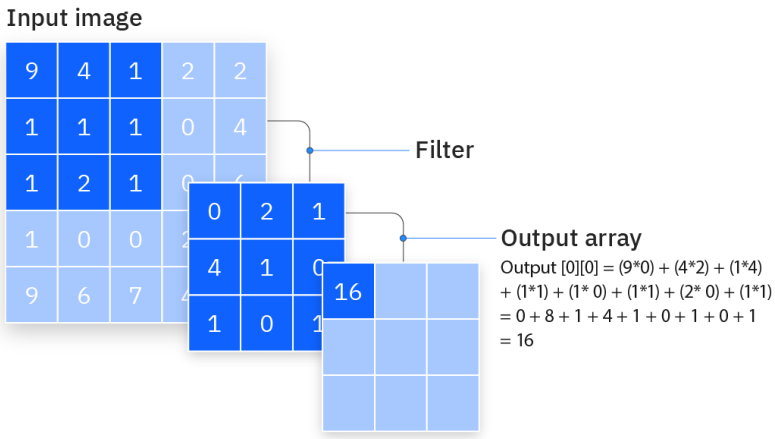

Figura 31 Capas Convolucionales

[5]

3.3.1.3. Capas de pooling

La agrupación de capas, también conocida como submuestreo o Pooling en inglés, permite reducir la dimensión mediante la reducción del número de parámetros de la entrada. De manera similar a la capa convolucional, la operación de agrupación barre toda la entrada con un filtro, pero la diferencia es que este filtro no tiene ningún peso. En su lugar, el kernel aplica una función de agregación a los valores dentro del campo receptivo y llena así la matriz de salida. Hay dos tipos principales de agrupación:

- Agrupación máxima: a medida que el filtro recorre la entrada, selecciona el píxel con el valor más alto para enviarlo a la matriz de salida. Este enfoque suele utilizarse más que la agrupación media.
- Agrupación media: conforme el filtro avanza por la entrada, calcula el valor promedio dentro del campo receptivo para enviarlo a la matriz de salida.

Aunque se pierde mucha información en la capa de agrupación, esta tiene una serie de beneficios para la CNN. Ayuda a reducir la complejidad, mejora la eficiencia y limita el riesgo de sobreajuste.

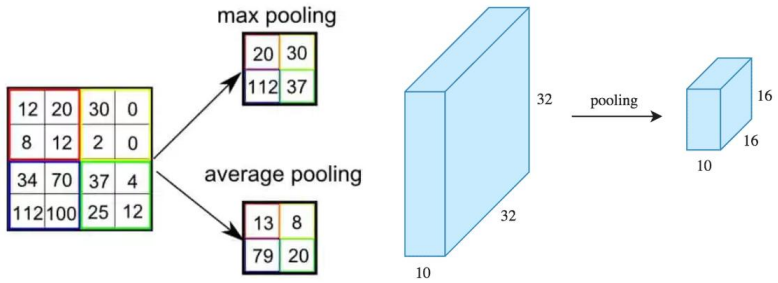

Figura 32 Pooling

[16]

3.3.1.4. Capas aplanadoras (flattening)

La salida final y la clasificación de los datos se realizan mediante capas completamente conectadas. Una vez que se genera la salida después de las operaciones de convolución y agrupación, el siguiente paso es aplanar la salida. Esto se hace para alimentar los datos agrupados a una red neuronal artificial. En realidad, la salida está en forma de volúmenes 3D, que deben convertirse en una forma aceptable de números mediante el aplanamiento. Eso se convierte en la entrada de la capa completamente conectada. El aplanamiento organiza una imagen 3D en un vector unidimensional o de columna.

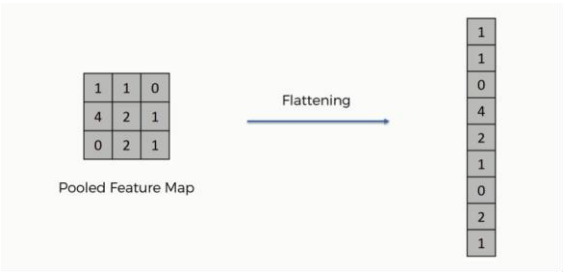

Figura 33 Flattening

[5]

3.3.1.5. Capas totalmente conectadas

El nombre de la capa totalmente conectada describe con precisión la capa en sí. Como se ha mencionado anteriormente, los valores de píxel de la imagen de entrada no están conectados directamente con la capa de salida en las capas parcialmente conectadas. Sin embargo, en la capa totalmente conectada, cada nodo de la capa de salida sí está conectado directamente a un nodo de la capa anterior.

Esta capa realiza la tarea de clasificación basándose en las características extraídas de las capas anteriores y sus diferentes filtros. Las capas convolucionales y de agrupación suelen utilizar funciones ReLU, mientras que las capas totalmente conectadas generalmente usan una función de activación softmax para clasificar adecuadamente las entradas y generar una probabilidad entre 0 y 1.

3.3.1.6. Capas de salida

Proporcionan la clasificación final de la imagen.

3.4. Funciones de activación

Las funciones de activación son fundamentales en las CNN porque introducen la no linealidad necesaria para que el modelo pueda aprender y representar relaciones complejas en los datos. Sin ellas, estaríamos limitados a un modelo lineal, incapaz de capturar patrones sofisticados.

En las CNN, las funciones de activación desempeñan un papel crucial en las capas de convolución, las capas totalmente conectadas y las capas de salida. Tras cada operación de convolución, el resultado se pasa a través de una función de activación, lo que permite al modelo extraer características más complejas y significativas.

Asimismo, en las capas totalmente conectadas y de salida, estas funciones son esenciales, ya que las características extraídas se utilizan para realizar clasificaciones o predicciones. Aquí, las funciones de activación permiten al algoritmo aprender interacciones complejas entre las características, asegurando un desempeño robusto en tareas como el reconocimiento de imágenes o la detección de objetos.

3.4.1. Capas convolucionales

3.4.1.1. ReLu

ReLU tiene una salida 0 si la entrada es menor que 0 y una salida bruta en caso contrario. Es decir, si la entrada es mayor que 0, la salida es igual a la entrada. El funcionamiento de ReLU está más cerca de la forma en que funcionan nuestras neuronas biológicas.

ReLU no es lineal y tiene la ventaja de no tener ningún error de retropropagación a diferencia de la función sigmoidea, también para NNs más grandes, la velocidad de construcción de modelos basados en ReLU es muy rápida en comparación con el uso de Sigmoides.

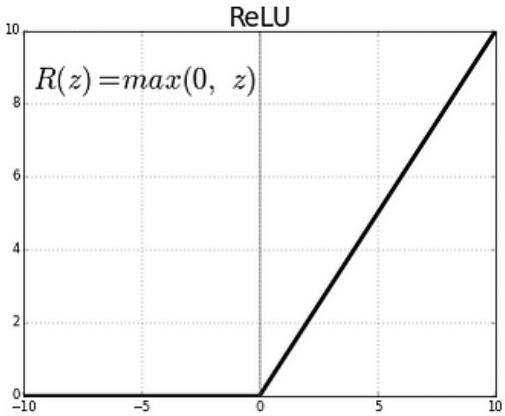

Figura 34 ReLu

Imagen generada con IA

3.4.1.2. Leaky ReLu

Un problema que se puede observar en ReLU es el problema de *Dying ReLU*, donde algunas neuronas de ReLU esencialmente mueren por todas las entradas y permanecen inactivas sin importar la entrada que se suministre, por lo que no fluye gradiente y si hay una gran cantidad de neuronas muertas en una red neuronal, su rendimiento es afectado.

Esto puede corregirse haciendo uso de lo que se llama Leaky ReLU donde la pendiente se cambia a la izquierda de x = 0 en la gráfica y, por lo tanto, provoca una fuga y extiende el rango de ReLU.

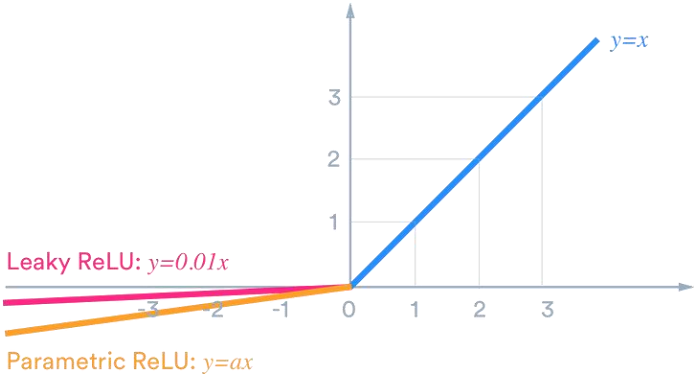

Figura 35 Leaky ReLu

Imagen generada con IA

3.4.2. Capas de salida

3.4.2.1. Sigmoide

La función sigmoidea realiza la siguiente transformación en la entrada x, lo que genera un valor de salida entre 0 y 1.

Aunque la función sigmoidea y su derivada es simple, lo que ayuda a reducir el tiempo requerido para hacer modelos, existe un gran inconveniente de pérdida de información debido a que la derivada tiene un rango corto.

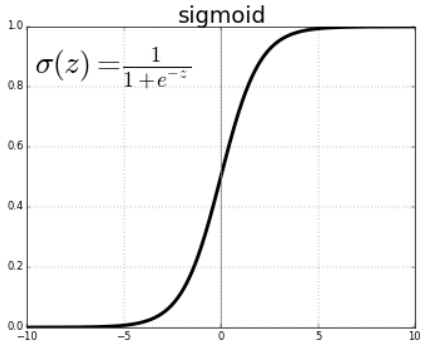

Figura 36 Sigmoide

Imagen generada por IA

3.4.2.2. SoftMax

La función SoftMax es una función de activación que se utiliza en la capa de salida de una CNN para realizar la clasificación multiclase. Esta función se utiliza para convertir las salidas de la capa anterior en probabilidades que suman uno. Las probabilidades se utilizan para medir la confianza del modelo en la pertenencia a cada clase.

La función SoftMax toma como entrada un vector de valores y produce una salida en forma de vector de la misma dimensión. La función SoftMax realiza dos operaciones: primero, exponencia los elementos del vector de entrada, y segundo, divide cada uno de los elementos exponenciados por la suma de los elementos exponenciados.

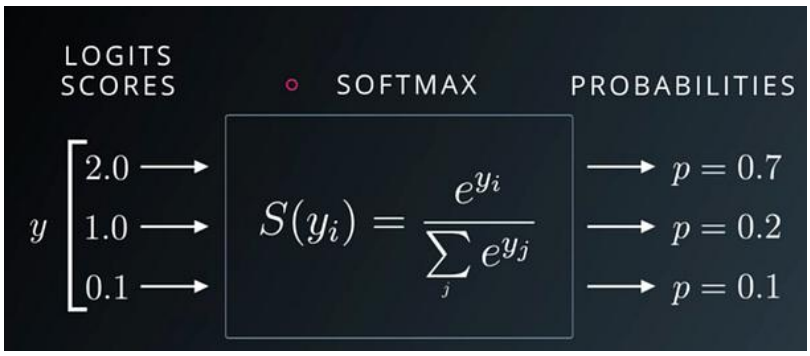

Figura 37 Funcionamiento Softmax

Imagen generada por IA

La función SoftMax se utiliza en la clasificación multiclase porque asigna una probabilidad a cada clase. Esto significa que la función SoftMax es útil cuando se necesita saber la probabilidad de que una instancia pertenezca a cada clase.

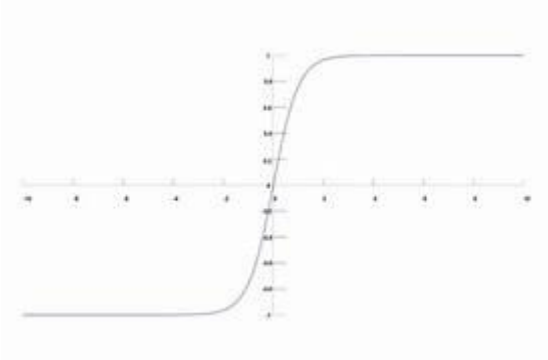

Figura 38 Softmax

Imagen generada por IA

3.5. Tipos de Pérdidas

3.5.1. Error cuadrático

Es la métrica comúnmente utilizada para problemas de regresión, se caracteriza por concentrarse en las distancias al cuadrado entre dos valores numéricos, la primera razón por la cual se consideran valores al cuadrado es para eliminar la posibilidad de que errores grandes negativos se anulen con errores grandes positivos.

$$MSE\ (y, \hat{y}) = (y - \hat{y})^2$$

3.5.2. Error del coseno

En algunos casos no será suficiente con calcular el error entre dos números, pensemos por ejemplo cuando la variable objetivo es más rica en estructura como las respuestas de un LLM a un prompt, en estos casos nos gustaría poder comparar dos vectores en lugar de comparar únicamente dos números.

El error del coseno compara la relación lineal que existe coordenada a coordenada la cual coincide con el ángulo entre los dos vectores.

$$Error\ coseno\ (y, \hat{y}) = \frac{\langle y, \hat{y} \rangle}{\|y\| \cdot \|\hat{y}\|}$$

3.5.3. Error de entropía cruzada

Cuando el tipo de variable que deseamos predecir es parecida a una categórica podría ser necesario utilizar métricas que se generalicen mejor para distintas clases como lo es la entropía cruzada.

La motivación de esta función de pérdida proviene del estudio de la información desde un punto de vista matemático el cual fue iniciado por Shannon.

$$Error\ entropía\ (y, \hat{y}) = -y \cdot \log(\hat{y}) - \hat{y} \cdot \log(y)$$

3.5.4. Error disperso

En las redes neuronales profundas la sobreparametrización de un modelo puede tener graves consecuencias tanto para el proceso computacional como para la aproximación estadística de un buen desempeño del modelo. En algunos casos es posible incluir ciertas hipótesis sobre la naturaleza de nuestros datos, por ejemplo, si son dispersos.

$$Error\ sparse\ (y, \hat{y}) = Error(y, \hat{y}) - \|M\|_1$$

3.5 Métricas

3.5.1. Pérdidas de entrenamiento

3.5.1.1. Pérdidas de Clasificación

Esta métrica cuantifica cuánto se desvían las predicciones del modelo de las etiquetas reales durante el entrenamiento. Por ejemplo, la pérdida de entropía cruzada para clasificación.

3.5.2 Métricas de validación y test

3.5.2.1. Precisión

Mide la exactitud de las predicciones positivas. Se calcula como la proporción de verdaderos positivos sobre la suma de los verdaderos positivos y los falsos positivos.

$$precision = \frac{verdaderos\ positivos}{verdaderos\ positivos + falsos\ positivos}$$

3.5.2.2. Recall

Mide la capacidad del modelo para identificar correctamente todas las instancias positivas. Se calcula como la proporción de verdaderos positivos sobre la suma de los verdaderos positivos y los falsos negativos.

$$recall = \frac{verdaderos\ positivos}{verdaderos\ positivos + falsos\ negativos}$$

3.5.2.3. F1 Score

Es la media armónica de la precisión y el recall, balanceando ambas métricas. Es útil cuando hay una distribución desigual entre clases (por ejemplo, cuando las clases positivas son mucho menos frecuentes).

$$F1\ Score = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

3.5.2.4. mAP50

Mide la precisión media en detección de objetos, evaluando la precisión a través de varias puntuaciones de intersección sobre la unión (IoU). La loU es una métrica de detección de objetos que mide el grado de solapamiento entre el cuadro delimitador previsto y el cuadro delimitador real. Los valores de loU oscilan entre 0 y 1, donde uno significa una coincidencia perfecta. La loU es esencial porque mide hasta que punto los límites predichos coinciden con los límites reales del objeto Es un valor comúnmente usado para evaluar modelos de detección de objetos, donde se calcula a una intersección mínima de 50%.

3.5.2.5. mAP50-95

Similar a mAP50, pero calcula la media de precisión a diferentes umbrales de IoU entre 50% y 95%, proporcionando una evaluación más exhaustiva del rendimiento en la detección de objetos a distintos niveles de superposición.

3.5.2.6. Matriz de Confusión

Aunque no es realmente una métrica como tal, sino una tabla de contingencia utilizada para evaluar el rendimiento de un modelo de clasificación, la utilizaremos para medir métricas posteriormente. Muestra los verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos para cada clase en el conjunto de entrenamiento. Es útil para entender los tipos de errores cometidos por el modelo.

Figura 39 Matriz de confusión

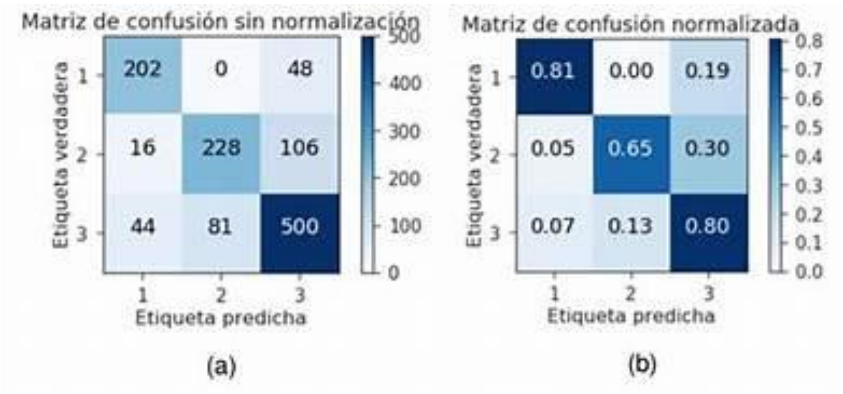

Figura 40 Ejemplos de Matriz de Confusión

[5]

3.6. Hiperparámetros ajustables

3.5.5. Tasa de aprendizaje

La tasa de aprendizaje (*Learning rate*) tiene como objetivo evitar que los incrementos que se aplican a los parámetros internos del modelo, los pesos de la neurona sean demasiado grandes.

Una tasa de aprendizaje menor supone incrementos menores, por lo que resultará más fácil encontrar los valores óptimos para los pesos, pero también supone que serán necesarios más cambios en los pesos para dar con ellos.

Habitualmente suele ser un valor pequeño como 0.001, aunque se han ido desarrollando estrategias cada vez más eficientes para gestionar este parámetro. Así, la tasa de aprendizaje puede ser un valor constante como ocurría con el Perceptrón o puede variar durante el proceso de entrenamiento del algoritmo.

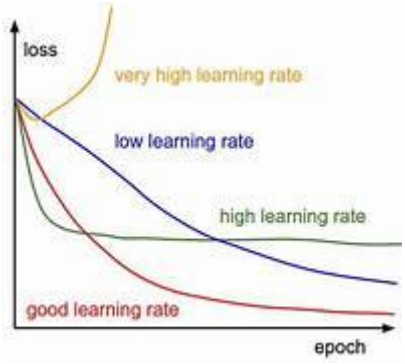

Figura 41 Learning Rate

[8]

3.5.6. Tamaño del batch

El tamaño del batch, la cantidad de ejemplos de entrenamiento en una iteración, adquiere mayor importancia en dimensiones superiores. En este caso, cuando los datos de entrada y la arquitectura del modelo involucran múltiples características o capas, la elección del tamaño del batch se convierte en una tarea delicada.

Se deben tener en cuenta las siguentes consideraciones:

- Sobreajuste y generalización: en dimensiones más altas, el riesgo de sobreajuste se vuelve más elevado. Los tamaños de batch más pequeños mejoran la generalización del modelo al actualizar los pesos con frecuencia, pero introducen ruido. Los tamaños de lote más grandes ofrecen una major convergencia, pero pueden provocar sobreajuste.
- Eficiencia computacional: las dimensiones más altas exigen mayores recursos computacionales. Los tamaños de lote más pequeños explotan el paralelismo, aprovechando las capacidades de GPU o CPU. Sin embargo, sobrecargar las actualizaciones frecuentes de peso puede contrarrestar los efectos positivos que tienen.
- Restricciones de memoria: los tamaños de lote más grandes consumen más memoria, lo que limita la escalabilidad del modelo.

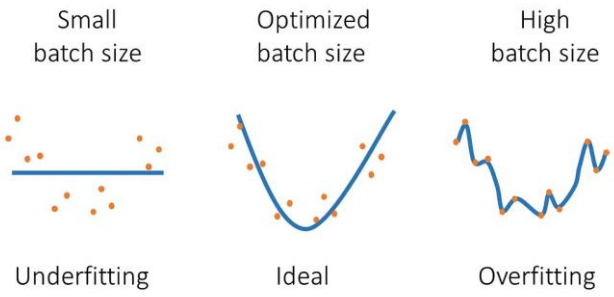

Figura 42 Batch Size

[5]

3.5.7. Optimizadores

3.5.7.1. AdaGrad

AdaGrad (Adaptive Gradient Algorithm) es un optimizador basado en gradiente que ajusta automáticamente la tasa de aprendizaje (learning rate) para cada parámetro del modelo de acuerdo con su historial de actualizaciones.

Esto permite que los parámetros con gradientes frecuentes se actualicen con un paso más pequeño, mientras que los parámetros con gradientes raros reciben pasos más grandes. AdaGrad ajusta la tasa de aprendizaje usando la suma acumulativa de los cuadrados de los gradientes.

Presenta un problema de amortiguación: La acumulación de gradientes puede hacer que la tasa de aprendizaje se vuelva muy pequeña con el tiempo, deteniendo el aprendizaje. Se suele usar en NLP debido a la naturaleza dispersa de los datos.

3.5.7.2. RMSProp

RMSprop es una técnica de optimización basada en gradientes que se utiliza para entrenar redes neuronales. Los gradientes de funciones muy complejas, como las redes neuronales, tienden a desaparecer o explotar a medida que los datos se propagan a través de la función. RMSProp se desarrolló como una técnica estocástica para el aprendizaje por mini lotes.

RMSprop utiliza una tasa de aprendizaje adaptativa en lugar de tratar la tasa de aprendizaje como un hiperparámetro. Esto significa que la tasa de aprendizaje cambia con el tiempo.

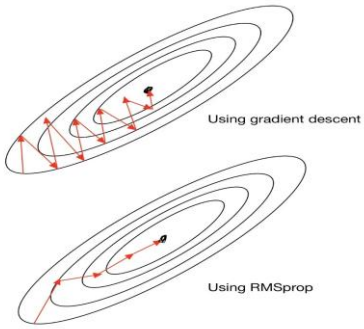

Figura 43 RMSProp

Imagen generada por IA

3.5.7.3. Adam

El optimizador Adam (Adaptive Moment Estimation) es uno de los algoritmos más populares en el entrenamiento de modelos de aprendizaje profundo. Su eficacia se debe a su capacidad para combinar las ventajas de dos optimizadores: AdaGrad y RMSProp. Este método utiliza estimaciones de momentos de primer y segundo orden para ajustar de manera adaptativa las tasas de aprendizaje de cada parámetro.

Adam se ha convertido en uno de los algoritmos más populares para el entrenamiento de redes neuronales debido a sus múltiples ventajas. Una de las principales características que lo distingue es su capacidad para ajustar automáticamente la tasa de aprendizaje a lo largo del proceso de entrenamiento. Esto permite una convergencia más rápida y eficiente, ya que el optimizador puede adaptarse a las características específicas de cada problema y a las diferentes dinámicas del gradiente.

Otra ventaja de Adam es su capacidad para manejar grandes volúmenes de datos y parámetros. Esto se traduce en una mayor estabilidad durante el entrenamiento, especialmente en situaciones donde los datos son ruidosos o escasos.

El uso de Adam también contribuye a la reducción del sobreajuste, ya que su mecanismo incorporado de regularización ayuda a mantener el modelo generalizable.

3.5.7.4. AdamW

En 2017, Ilya Loshchilov y Frank Hutter introdujeron una versión más avanzada del popular algoritmo Adam en su artículo "Regularización por Decaimiento de Peso Desacoplado." AdamW, que destaca por desacoplar el decaimiento del peso del proceso de actualización del gradiente.

AdamW separa el decaimiento del peso del paso del gradiente, garantizando que la regularización afecte directamente a los parámetros sin alterar el mecanismo de aprendizaje adaptativo.

Esto conduce a una regularización más precisa, ayudando a que los modelos generalicen mejor, sobre todo en tareas que implican conjuntos de datos grandes y complejos.

3.6. CNNs más famosas

3.6.1. LeNet

LeNet-5, una red convolucional de 7 niveles que clasifica dígitos, fue aplicada por varios bancos para reconocer números escritos a mano en cheques (cheques) digitalizados en imágenes de entrada en escala de grises de 32x32 píxeles. La capacidad de procesar imágenes de mayor resolución requiere capas más grandes y convolucionales, por lo que esta técnica está limitada por la disponibilidad de recursos informáticos.

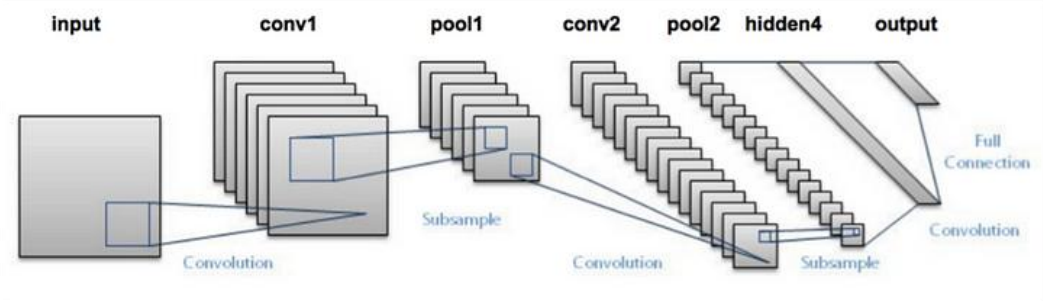

Figura 44 LeNet

[5]

3.6.2. AlexNet

AlexNet tiene una arquitectura muy similar a la de LeNet, pero más profunda, con más filtros por capa y capas convolucionales apiladas.

Su arquitectura se compone de cinco capas convolucionales cuyas salidas son normalizadas por batch. La primera, cuarta y quinta capas convolucionales vienen seguidas cada una de ellas por una capa de agrupación (max-pooling) de 3×3, en la que existe un desplazamiento de dos, lo que provoca que haya un solapamiento de las celdas de esta capa. Las capas convolucionales y de pooling vienen seguidas de dos capas densas, de 4.096 neuronas cada una, y de una última capa densa de 1.000 neuronas de salida, dotada de la función de activación softmax. Son estas últimas las encargadas de realizar la clasificación de la imagen.

Tiene activaciones de ReLU después de cada capa convolucional y completamente conectada.

Durante cada batch de entrenamiento, se procede al apagado aleatorio de la mitad de las neuronas, con el objetivo de prevenir el overfitting de la red, y favorecer las capacidades de generalización de esta última.

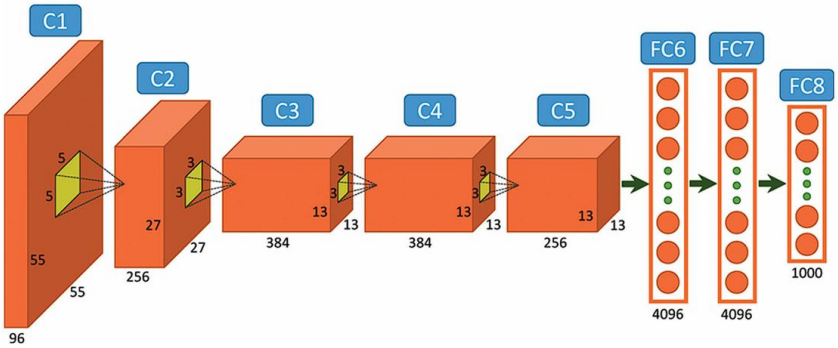

Figura 45 AlexNet

[3]

3.6.3. VGGNet

VGGNet consta de 16 capas convolucionales y tiene una arquitectura muy uniforme. Similar a AlexNet, solo convoluciones 3x3, pero muchos filtros.

Es una de las opciones más preferidas para extraer características de imágenes. La configuración de pesos de VGGNet está disponible públicamente, sin embargo, VGGNet consta de 138 millones de parámetros, que pueden ser un poco difíciles de manejar.

Los componentes clave de la arquitectura VGGNet son:

1. Capas convolucionales: filtros 3x3 con un paso de 1 y un relleno de 1 para preservar la resolución espacial.
2. Función de activación: ReLU (Unidad lineal rectificada) se aplica después de cada capa convolucional para introducir no linealidad.
3. Capas de agrupación: agrupación máxima con un filtro 2x2 y un paso de 2 para reducir las dimensiones espaciales.
4. Capas completamente conectadas: tres capas completamente conectadas al final de la red para la clasificación.
5. Capa Softmax: capa final para generar probabilidades de clase.

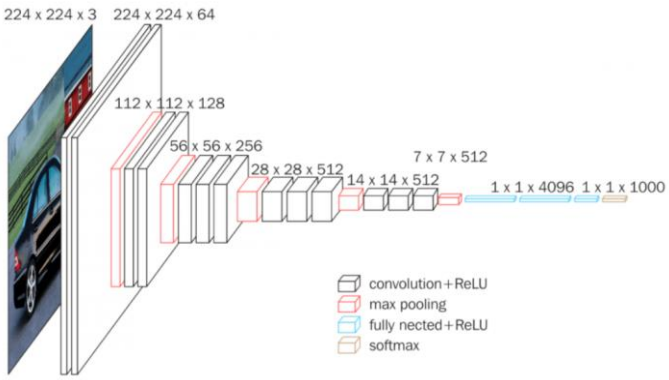

Figura 46 VGGNet

[3]

VGGNet se ha utilizado ampliamente en transfer learning debido a sus sólidas capacidades de extracción de características. Los modelos VGGNet entrenados previamente en grandes conjuntos de datos como ImageNet suelen ajustarse para diversas tareas de visión artificial, incluidas la detección de objetos, la segmentación de imágenes y la transferencia de estilos.

3.6.4. GoogleLeNet

GoogleLeNet utiliza una CNN inspirada en LeNet, pero implementa un elemento novedoso que se denomina módulo de inicio. Este módulo se basa en varias convoluciones muy pequeñas para reducir drásticamente el número de parámetros. Su arquitectura consiste en una CNN de 22 capas de profundidad, pero reduce el número de parámetros de 60 millones (AlexNet) a 4 millones.

La arquitectura de GoogLeNet es muy diferente de las arquitecturas de vanguardia anteriores, como AlexNet y ZF-Net. Utiliza muchos tipos diferentes de métodos, como la convolución *1×1* y la agrupación de promedios globales, que le permiten crear una arquitectura más profunda. En la arquitectura, analizaremos algunos de estos métodos:

- Convolución *1×1*: La arquitectura Inception utiliza *una convolución 1×1* en su arquitectura. Estas convoluciones solían reducir la cantidad de parámetros (pesos y sesgos) de la arquitectura. Al reducir los parámetros, también aumentamos la profundidad de la arquitectura.
- Agrupamiento de promedio global: en la arquitectura anterior, como AlexNet, las capas completamente conectadas se utilizan al final de la red. Estas capas completamente conectadas contienen la mayoría de los parámetros de muchas arquitecturas, lo que provoca un aumento en el costo computacional. En la arquitectura GoogLeNet, hay un método llamado agrupamiento de promedio global que se utiliza al final de la red. Esta capa toma un mapa de características de 7x7 y lo promedia a 1x1.
- Módulo Inception: el módulo Inception es diferente de las arquitecturas anteriores, como AlexNet y ZF-Net. En esta arquitectura, hay un tamaño de convolución fijo para cada capa. En el módulo Inception, la convolución 1×1, 3×3, 5×5 y la agrupación máxima 3×3 se realizan de forma paralela en la entrada y la salida de estas y se apilan juntas para generar la salida final. La idea detrás de eso es que los filtros de convolución de diferentes tamaños manejarán mejor los objetos en múltiples escalas.

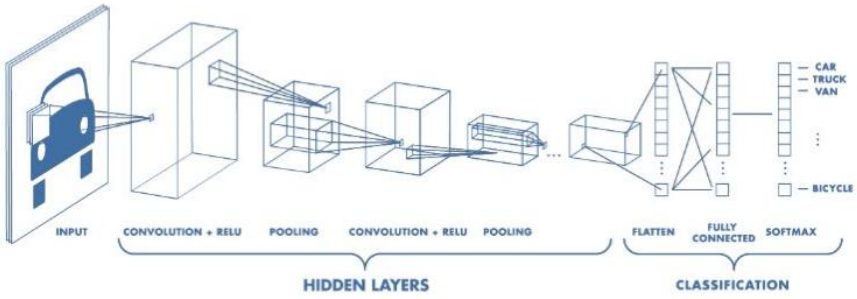

Figura 47 Google LeNet

[3]

3.6.5. DenseNet

DenseNet [17] propone un nuevo patrón de conectividad dentro de las CNN, abordando desafíos como la reutilización de características, los gradientes de desaparición y la eficiencia de los parámetros. A diferencia de las arquitecturas CNN tradicionales donde cada capa está conectada solo a las capas posteriores, DenseNet establece conexiones directas entre todas las capas dentro de un bloque. Esta conectividad densa

permite que cada capa reciba mapas de características de todas las capas anteriores como entradas, lo que fomenta un amplio flujo de información en toda la red.

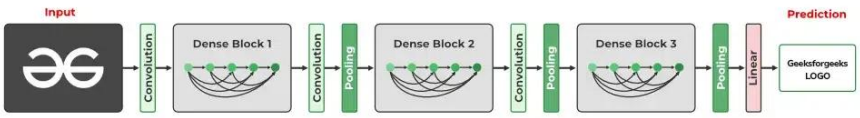

Figura 48 DenseNet

[17]

Los bloques densos son los componentes básicos de las arquitecturas DenseNet. Cada bloque denso consta de múltiples capas convolucionales, generalmente seguidas de una normalización por lotes y una función de activación no lineal (por ejemplo, ReLU). Es importante destacar que cada capa dentro de un bloque denso recibe mapas de características de todas las capas anteriores como entradas, lo que facilita la reutilización y propagación de características.

Ejemplo de un bloque denso:

- Capa 1: Recibe mapas de características de entrada.
- Capa 2: Recibe mapas de características de entrada + salida de la Capa 1.
- Capa 3: Recibe mapas de características de entrada + salida de la Capa 1 + salida de la Capa 2.

Este patrón continúa para todas las capas dentro del bloque, lo que garantiza una arquitectura altamente interconectada.

Las capas de transición se utilizan para conectar bloques densos. Cumplen dos propósitos principales: reducir la cantidad de mapas de características y reducir el tamaño de las dimensiones espaciales de los mapas de características. Esto ayuda a mantener la eficiencia computacional y la compacidad de la red. Una capa de transición típica consta de:

- Normalización por lotes: normaliza los mapas de características.
- Convolución 1x1: reduce la cantidad de mapas de características.
- Agrupamiento promedio: reduce la resolución de las dimensiones espaciales.

La tasa de crecimiento (k) es un hiperparámetro fundamental en DenseNet. Define la cantidad de mapas de características que produce cada capa de un bloque denso. Una tasa de crecimiento mayor significa que se agrega más información en cada capa, pero también aumenta el costo computacional. La elección de k afecta la capacidad y el rendimiento de la red.

4 YOLO

The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point.

Claude Shannon, 1948

En esta sección se presenta la red utilizada en nuestra aplicación, que se centra en la detección y localización de objetos dentro de imágenes. A diferencia de las técnicas de clasificación de imágenes, que asumen un único objeto principal y se limitan a determinar su categoría, nuestro enfoque busca identificar todos los objetos presentes, asignarles una clase y señalar su ubicación exacta mediante cuadros delimitadores. Para esta tarea, YOLO (*You Only Look Once*) [18],[19] se destaca como una solución ideal, ya que permite realizar detecciones rápidas y precisas en una sola pasada por la imagen. Esta capacidad para combinar la identificación de categorías con la localización de cada objeto hace que YOLO sea fundamental para cumplir con los requisitos de nuestra aplicación, que van más allá de la simple clasificación.

4.1. Historia de YOLO

La familia YOLO ha transformado la visión artificial desde su debut en 2016. Joseph Redmon y su equipo introdujeron este innovador marco de trabajo, alterando el panorama del procesamiento en tiempo real en inteligencia artificial.

El origen de YOLO comenzó con YOLOv1, que empleaba redes neuronales convolucionales para la detección rápida de objetos en imágenes. El procesamiento en tiempo real y la alta precisión de este modelo lo distinguieron de los detectores de dos etapas anteriores.

A medida que YOLO avanzaba, cada iteración introducía mejoras notables. YOLOv2 y YOLOv3 mejoraron la arquitectura, y las versiones posteriores, como YOLOv4 y YOLOv5, elevaron aún más el rendimiento. Las últimas versiones, incluida YOLOv11, siguen mejorando la eficiencia y la precisión en la detección de objetos.

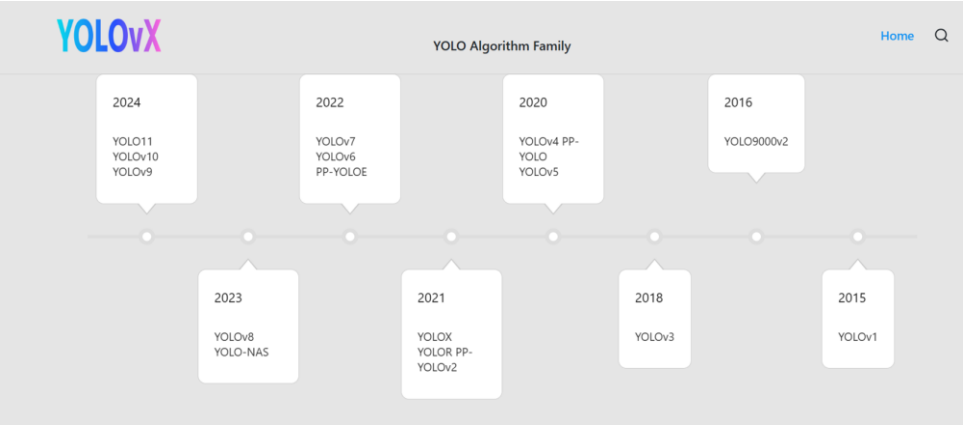

Figura 49 Historia de YOLO

[18]

4.2. Comparativa de versiones de YOLO

A continuación, realizaremos una comparativa de rendimiento de las versiones de YOLO, aunque somos conscientes de que existen versiones más recientes y con mejor rendimiento, son menos estables y tienen peor soporte y una comunidad más pequeña para búsqueda de información.

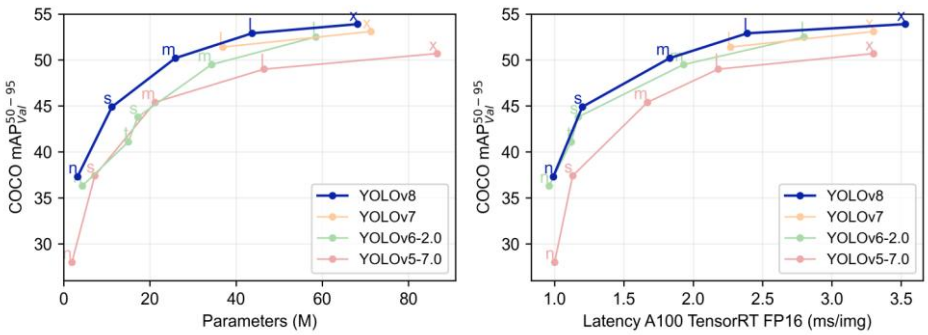

Figura 50 Comparativa de versiones de YOLO

[18]

La *Figura 51* presenta una comparación del rendimiento de las versiones YOLOv5, YOLOv6, YOLOv7 y YOLOv8 en dos aspectos clave: la precisión (COCO mAP50−95) frente al número de parámetros y frente a la latencia.

El COCO mAP50-95, también conocido como mAP@[0.5:0.95], es una métrica de evaluación usada en la detección de objetos, especialmente en el dataset COCO. Se refiere al "Mean Average Precision" (mAP) calculado en un rango de umbrales de IoU (Intersección sobre Unión) desde 0.5 hasta 0.95, con incrementos de 0.05.

1. Gráfico de precisión vs. parámetros (izquierda):
    - Este gráfico evalúa cómo varía la precisión a medida que aumenta la complejidad del modelo, representada por el número de parámetros (M).
    - YOLOv8 supera consistentemente a las versiones anteriores (YOLOv5, YOLOv6 y YOLOv7) para todas las configuraciones del modelo (nano, small, medium, large, extra-large).
    - Esto demuestra que YOLOv8 logra un mejor equilibrio entre precisión y complejidad del modelo, maximizando el rendimiento incluso con modelos más pequeños.
2. Gráfico de precisión vs. latencia (derecha):
    - Este gráfico analiza la relación entre la precisión y la latencia, medida como el tiempo de inferencia por imagen en milisegundos (ms/img) utilizando una GPU A100 con TensorRT FP16.
    - YOLOv8 muestra una clara ventaja sobre las versiones anteriores, ofreciendo una mayor precisión con tiempos de respuesta más bajos. Esto es especialmente importante para aplicaciones en tiempo real, donde la rapidez es crucial.
    - La mejora es evidente en todas las escalas de los modelos, lo que resalta la eficiencia de YOLOv8.

YOLOv8 representa un avance significativo en comparación con YOLOv5, YOLOv6 y YOLOv7, al proporcionar mejores resultados en términos de precisión y eficiencia. Esto lo convierte en una opción ideal para aplicaciones de detección de objetos, especialmente aquellas que requieren alta precisión y baja latencia.

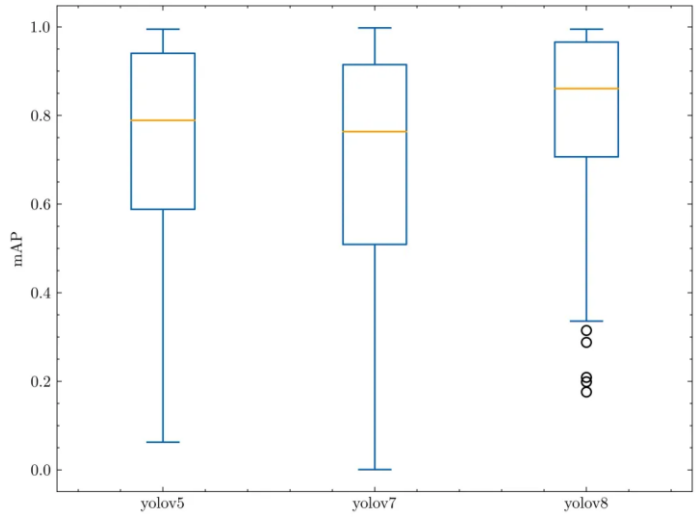

Figura 51 Comparativa mAP YOLO

[18]

4.3. YOLOv8

4.3.1. Modelos de YOLOv8

En este apartado realizaremos una comparación de las diferentes variantes de YOLOv8 [18], [19], [20] (Nano, Small, Medium, Large y XLarge) mediante tres parámetros clave: tamaño del modelo en MB, latencia en GPU (A100, FP16), y el desempeño medido en términos de mAP (Mean Average Precision) sobre el conjunto de datos COCO.

- Tamaño del modelo (MB):
    - YOLOv8n (Nano): Es la variante más ligera, con solo 6.5 MB.
    - YOLOv8x (XLarge): Es la más grande, alcanzando 136.9 MB.

  A medida que el modelo aumenta en tamaño (de Nano a XLarge), también lo hace la complejidad de su arquitectura.
- Latencia (ms en A100):
    - YOLOv8n (Nano): Tiene la menor latencia, procesando imágenes en 0.99 ms.
    - YOLOv8x (XLarge): La latencia aumenta hasta 3.53 ms, reflejando la mayor demanda computacional.

  Existe un compromiso entre la velocidad de inferencia y el tamaño del modelo.
- mAP en COCO (Mean Average Precision):
    - YOLOv8n (Nano): Obtiene un mAP de 37.3%, el más bajo.
    - YOLOv8x (XLarge): Alcanza un mAP de 53.9%, el más alto.

  El rendimiento mejora significativamente con el tamaño del modelo.
- Descripción de YOLOv8s (Small):
    - Tamaño: 22.6 MB, lo que lo posiciona como una solución intermedia, más ligera que Medium, Large y XLarge.
    - Latencia: 1.2 ms por imagen, siendo rápido en sistemas modernos con GPU.
    - Precisión (mAP COCO): 44.9%, mostrando un balance aceptable entre rendimiento y

eficiencia computacional.

YOLOv8s [21] logra una mAP del 44.9%, lo cual es suficiente para muchas aplicaciones prácticas, como detección en tiempo real, que es nuestro caso o en sistemas embebidos.

Su latencia baja (1.2 ms) permite procesamiento eficiente en escenarios de tiempo real, ideal para dispositivos que priorizan la rapidez.

Con un tamaño de solo 22.6 MB, YOLOv8s es lo suficientemente compacto para implementarse en dispositivos con capacidad de memoria limitada, sin sacrificar significativamente la precisión.

Aunque no es tan ligero como Nano, Small sigue siendo más económico en coste computacional que Medium, Large o XLarge, lo que reduce los requerimientos de hardware y energía.

Hemos elegido YOLOv8s para priorizar un equilibrio entre velocidad, precisión y tamaño del modelo.

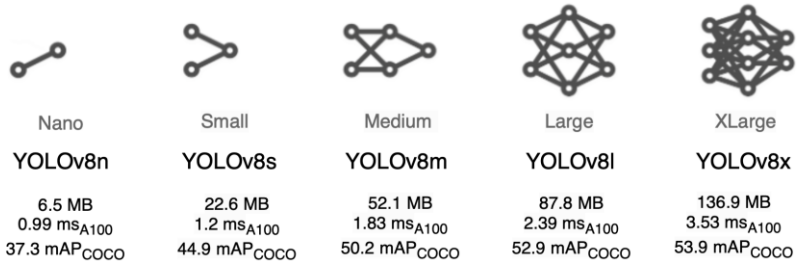

Figura 52 Modelos de YOLOv8

[18]

4.3.2. Cuadros delimitadores

Un cuadro delimitador [21] es un rectángulo que rodea un objeto detectado en una imagen. En YOLOv8, cada cuadro se representa mediante las siguientes coordenadas y atributos:

- x, y: Coordenadas del centro del cuadro en términos relativos al tamaño de la imagen.
- Width, Height: Ancho y alto del cuadro, también en términos relativos al tamaño de la imagen.
- Clase: Etiqueta asignada al objeto detectado (por ejemplo, "persona", "automóvil", etc.).
- Confianza: Probabilidad de que el cuadro contiene el objeto de la clase asignada. Es un valor entre 0 y 1.

Figura 53 Cuadros delimitadores

[18]

YOLOv8 divide la imagen en una cuadrícula y predice múltiples cuadros delimitadores para cada celda. Cada predicción incluye las coordenadas del cuadro, la clase y la confianza asociada.

En lugar de generar múltiples propuestas y refinarlas, YOLOv8 predice directamente las coordenadas y las características de los cuadros desde la red neuronal, optimizando la velocidad de detección.

YOLOv8 detecta objetos a diferentes niveles de la imagen, permitiendo identificar tanto objetos pequeños como grandes de manera precisa.

4.4. Arquitectura de YOLOv8

YOLOv8 es una evolución de la familia YOLO, diseñada para mejorar la detección de objetos, segmentación y clasificación. Su arquitectura está basada en principios de diseño eficientes, combinando innovaciones de las versiones anteriores con mejoras propias que optimizan tanto la precisión como la velocidad. YOLOv8 está estructurado en tres bloques principales: Backbone, Neck y Head, cada uno con un propósito específico en la extracción, procesamiento y predicción de características a partir de las imágenes.

A diferencia de sus predecesores, YOLOv8 [21] incorpora un enfoque más flexible en la configuración del modelo, técnicas modernas como predicción libre de anclajes (anchor-free) y una integración de módulos ligeros que permiten implementaciones en dispositivos con recursos limitados, además de grandes sistemas de inferencia en la nube.

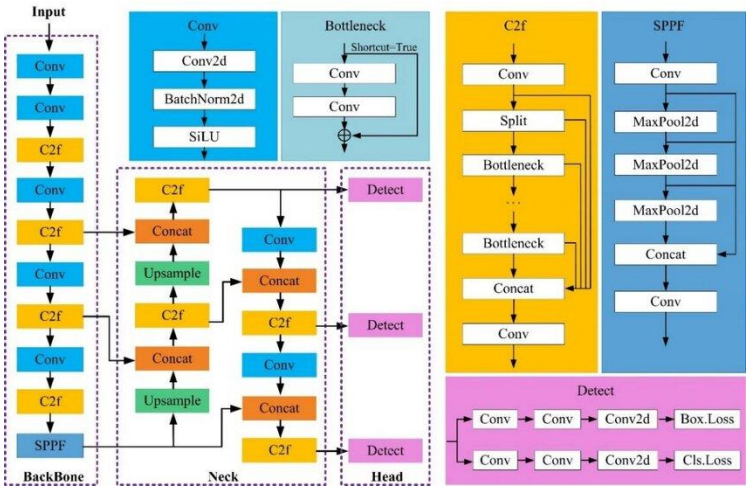

Figura 54 Arquitectura de YOLOv8

[18]

4.4.1. Bloques principales

4.4.1.1. Backbone

El Backbone es el encargado de extraer características jerárquicas de la imagen de entrada. Se basa en redes convolucionales profundas que detectan patrones básicos como bordes, texturas y colores, que luego se combinan para identificar características más complejas.

Capas principales del Backbone:

- Focus Layer: Reduce el tamaño espacial de la imagen inicial y aumenta su profundidad, capturando

características importantes desde el inicio.

- Convoluciones estándar y profundas: Extraen características básicas y avanzadas mediante filtros convolucionales, BatchNormalization y función de activación Swish.
- C2f: Una variante mejorada de C3 usada en YOLOv8, que emplea convoluciones con concatenaciones y atajos para mejorar la propagación de información.
- SPPF (Spatial Pyramid Pooling Fast):
    - Fusiona características de múltiples escalas espaciales, permitiendo que el modelo capture información de objetos en distintas dimensiones (grandes, medianos y pequeños).

4.4.1.2. Neck

El Neck actúa como un puente entre el Backbone y el Head. Su objetivo principal es procesar y fusionar las características extraídas del Backbone para que sean útiles en la tarea de detección. Se centra en mejorar la resolución de las características y manejar información de múltiples escalas.

Capas principales del Neck:

- Upsample: Se usa para aumentar la resolución de la imagen en ciertas capas y mejorar la detección de objetos pequeños.
- Concat: Concatenación de características de distintas escalas para mejorar la información de los objetos en la imagen.
- C2f: Similar a la capa C2f del Backbone, ayuda a refinar las características fusionadas.

4.4.1.3. Head

El Head es el componente final encargado de generar las predicciones. Este bloque toma las características procesadas por el Neck y las transforma en salidas que incluyen las coordenadas de los cuadros delimitadores, las clases de los objetos detectados y la confianza asociada a cada predicción.

Capas principales del Head:

- Detect: Se aplican capas de convolución para predecir cajas delimitadoras (Box Loss) y clases (Cls Loss) y se generan múltiples predicciones a diferentes escalas.
- Bottleneck: Contiene atajos (Shortcut=True) para mejorar la eficiencia del entrenamiento y la propagación del gradiente.
- Output Layers: Generan las salidas finales en el formato requerido (coordenadas *x, y, w, h* y confianza).

5 DESARROLLO DEL MODELO

En este capítulo vamos a describir el desarrollo, paso a paso del modelo. Empezando por la elección del dataset, analizando el rendimiento de un entrenamiento inicial, realizando aumento de datos y ajuste de hiperparámetros y terminando con una comparativa del rendimiento del modelo una vez mejorado y optimizado.

5.1. Tecnologías utilizadas

5.1.1 Lenguaje de programación

El desarrollo de nuestra aplicación se ha llevado a cabo utilizando Python, un lenguaje ampliamente adoptado en el ámbito de la inteligencia artificial, el aprendizaje automático y el procesamiento de datos, gracias a su flexibilidad, gran ecosistema de bibliotecas y facilidad de uso.

5.1.2 Entorno de desarrollo

Debido a las altas exigencias computacionales asociadas con el entrenamiento y la ejecución de modelos de aprendizaje profundo, hemos combinado varios entornos y plataformas:

- Visual Studio Code: empleado para el desarrollo y gestión del código fuente.
- Google Colab: utilizado para realizar pruebas y entrenar los modelos en entornos de alta capacidad de cómputo, aprovechando el acceso gratuito a GPUs y TPUs.

5.1.3 Frameworks y bibliotecas utilizadas

Para implementar los modelos y optimizar su rendimiento, hemos utilizado los siguientes frameworks y bibliotecas:

Frameworks principales:

- PyTorch y TensorFlow, ambos esenciales para el diseño, entrenamiento y evaluación de redes neuronales avanzadas.
- Roboflow, una herramienta clave para gestionar datasets y optimizar procesos de anotación y preparación de datos en proyectos de visión por computadora.

Librerías de soporte:

- Numpy y Pandas, fundamentales para la manipulación y análisis de datos.
- Matplotlib y Seaborn, para la visualización de datos y resultados de entrenamiento.
- Albumentations, utilizada para realizar aumentación de datos, mejorando la capacidad de generalización de los modelos.
- Ultralytics, para facilitar la implementación de modelos avanzados de detección de objetos.
- Shutil y os, para la manipulación de archivos y directorios.
- cv2 (OpenCV), imprescindible para el procesamiento de imágenes y videos.

Estas herramientas y bibliotecas han sido seleccionadas por su rendimiento, facilidad de integración y amplia documentación, asegurando una base tecnológica robusta para nuestro proyecto.

5.2. Elección del dataset

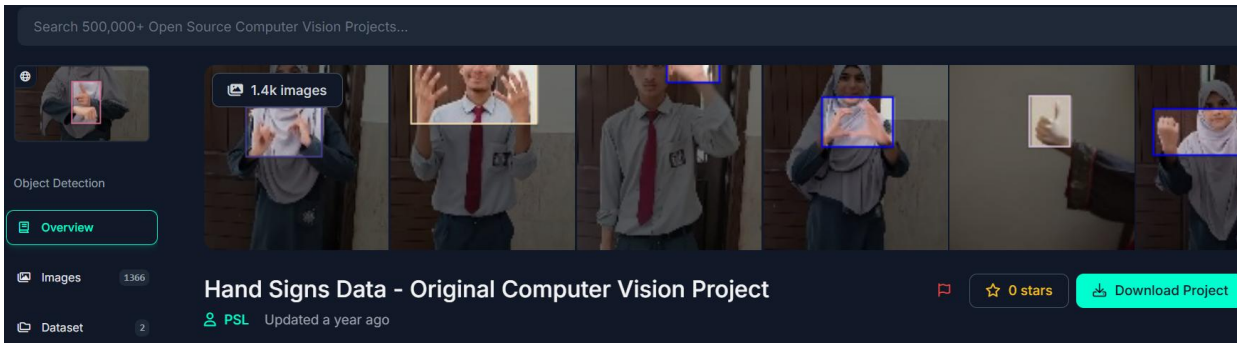

Figura 55 Dataset elegido

5.3. Estudio del dataset

5.3.1. Distribución de directorios

En la siguiente imagen se puede apreciar un diagrama de la distribución de directorios de nuestro dataset. La distribución es la siguiente:

- Directorio principal: *Hands Signs Data*

Este es el directorio raíz que contiene todos los elementos del proyecto. En él se encuentra el archivo *data.yaml* y tres carpetas principales para el conjunto de datos: train, valid y test.

- Archivo: *data.yaml*

Este archivo, ubicado directamente en el directorio principal, contiene la configuración del conjunto de datos. Normalmente incluye información como:

  - Clases del conjunto de datos.
  - Rutas a las imágenes y etiquetas para los conjuntos de entrenamiento, validación y prueba.
  - Formato de las anotaciones. Este archivo es esencial para que los modelos de aprendizaje automático puedan interpretar y cargar el conjunto de datos correctamente.

- Carpetas: train, valid, test

Estas carpetas organizan los datos en tres subconjuntos:

  - train: Contiene los datos utilizados para entrenar el modelo. En nuestro dataset lo componen 1092 imágenes, que suponen el 80% del total.
  - valid: Contiene los datos utilizados para validar el rendimiento del modelo durante el entrenamiento. En nuestro dataset lo componen 160 imágenes, que suponen el 12% del total.
  - test: Contiene los datos reservados para evaluar el rendimiento final del modelo después del entrenamiento En nuestro dataset lo componen 114 imágenes, que suponen el 8% del total.

- Subcarpetas: labels y images

Dentro de cada carpeta principal (train, valid y test), hay dos subcarpetas:

- labels: Almacena los archivos de anotación en formato .txt, que definen las clases y posiciones de los objetos en las imágenes.
- images: Contiene las imágenes reales asociadas con las etiquetas. Estas imágenes son el insumo principal para el modelo.

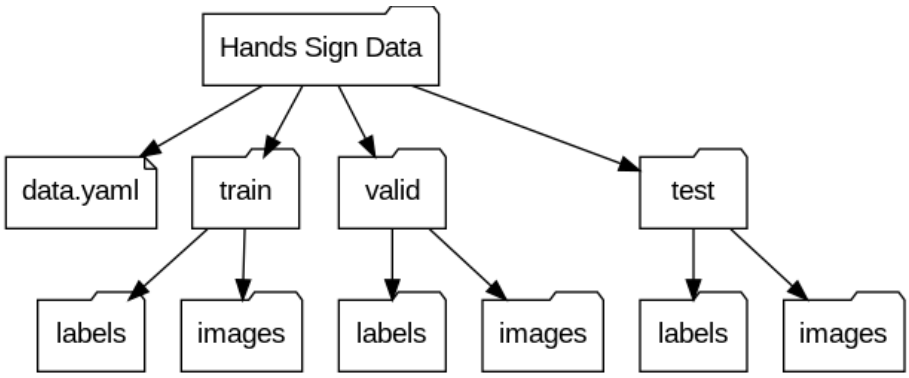

Figura 56 Diagrama de directorios

Imagen generada con IA

5.3.2. Clases del dataset

En la siguiente tabla se pueden apreciar las diferentes clases del dataset con ejemplos de imágenes de la carpeta train. Aunque en estos ejemplos no aparezca el bounding box, las imágenes están etiquetadas mediante archivos txt en la carpeta labels como hemos comentado en anterioridad.

Clase	Gesto
Communication	

Hardworking	
Interview	
Learned	

Meeting	
Opportunity	
Skill	

Sorry	
Success	
Time	

Tabla 6. 1 Clases del dataset

Imágenes extraídas de mi dataset

5.3.3. Carpeta train

Clase	Imágenes en train	Imágenes en valid	Imágenes en test
Communication	130	16	14
Hardworking	85	13	6
Interview	108	32	15
Learned	92	12	7
Meeting	131	16	16
Opportunity	83	10	2
Skill	148	19	18
Sorry	2	0	1
Success	144	18	18
Time	116	14	10

Tabla 6. 2 Distribución de clases

La *Tabla 6.2* presenta la distribución de imágenes por clase en tres subconjuntos: entrenamiento, validación y prueba. En general, se observa que la mayoría de las clases cuentan con un número relativamente equilibrado de imágenes, especialmente categorías como Skill, Success, Communication, Meeting e Interview, que tienen una representación adecuada en los tres conjuntos. Estas clases presentan cifras consistentes y suficientes como para permitir un entrenamiento robusto y una validación confiable del modelo.

Sin embargo, hay otras clases con una cantidad más reducida de imágenes, como Hardworking, Learned, Opportunity y Time. Aunque estas categorías sí están presentes en los tres conjuntos, su menor volumen podría afectar la capacidad del modelo para generalizar correctamente sobre ellas. A pesar de ello, su presencia sigue siendo suficiente para que el modelo pueda recibir al menos algo de información sobre estas clases durante el entrenamiento.

El caso más preocupante es el de la clase Sorry. Esta categoría está fuertemente desbalanceada: solo cuenta con dos imágenes en el conjunto de entrenamiento, ninguna en el de validación y una sola imagen en el conjunto de prueba. Esta situación implica que el modelo prácticamente no tendrá información sobre esta clase durante el aprendizaje, y tampoco podrá evaluar su rendimiento ni ajustar sus parámetros con respecto a ella. En la práctica, esto puede llevar a que el modelo simplemente ignore la clase Sorry o sea incapaz de reconocerla correctamente, generando sesgos y disminuyendo la precisión en tareas multiclase.

5.4. Resultados de entrenamiento del modelo inicial

5.4.1. Resultados validación

5.4.1.1. F1 Score

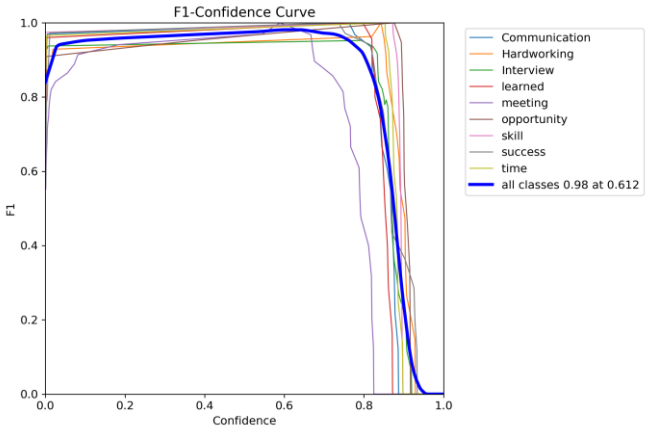

Figura 57 F1 Score modelo inicial

La curva en la *Figura 57* muestra un F1 Score relativamente alto cuando el umbral de confianza está por debajo de 0.5, lo que sugiere que el modelo tiende a predecir más objetos de manera conservadora, es decir, clasificando como positivos incluso cuando hay cierta incertidumbre.

A medida que el umbral de confianza aumenta las curvas de la mayoría de las clases caen significativamente, indicando que el modelo pierde precisión a medida que requiere mayor certeza para clasificar correctamente las imágenes. Sin embargo, las clases como Communication y Success siguen manteniendo un F1 relativamente alto hasta valores elevados de confianza.

El umbral óptimo para obtener el mejor F1 Score (0.98) es 0.612 en el modelo, lo que sugiere que ese umbral permite un buen balance entre precisión y recall sin perder demasiadas predicciones correctas.

Las clases con caídas más pronunciadas a valores altos de confianza es Meeting, lo que podría indicar que el modelo es menos confiable al realizar predicciones para esta clase, especialmente cuando se requiere mayor certeza.

5.4.1.2. Métricas de validación

Class	Images	Instances	Box(P	R	mAP50	mAP50-95):
all	160	150	0.979	0.98	0.987	0.742
Communication	160	16	0.994	1	0.995	0.705
Hardworking	160	13	0.911	1	0.995	0.85
Interview	160	32	0.963	0.938	0.928	0.725
learned	160	12	0.99	1	0.995	0.755
meeting	160	16	0.972	0.938	0.991	0.621
opportunity	160	10	0.996	1	0.995	0.754
skill	160	19	0.995	1	0.995	0.859
success	160	18	1	0.949	0.995	0.608
time	160	14	0.994	1	0.995	0.802

Figura 58 Resultados de validación del modelo inicial

5.4.1.2.1. Precisión (P)

Promedio global: 0.979, lo que significa que, en promedio, el modelo es muy preciso al identificar los objetos de las clases que ha predicho como positivas.

Clases destacadas:

- Success y Opportunity tienen una precisión perfecta de 1, lo que significa que todas las predicciones de estas clases son correctas.
- Hardworking tiene la menor precisión (0.911), lo que sugiere que hay cierta dificultad para identificar correctamente esta clase, posiblemente debido a características visuales que se asemejan a otras clases o un número insuficiente de ejemplos en los datos de entrenamiento.

5.4.1.2.2. Recall (R)

Promedio global: 0.98, lo que indica que el modelo es excelente para detectar casi todos los objetos de cada clase.

Clases destacadas:

- Hardworking tiene un recall perfecto de 1, lo que significa que el modelo detecta todos los objetos relevantes de esta clase, aunque podría no estar clasificándolos correctamente (de ahí la baja precisión).
- Interview y Meeting tienen los valores más bajos de recall (0.938), lo que sugiere que el modelo puede estar perdiendo algunos objetos relevantes de estas clases.

5.4.1.2.3. mAP

mAP50 (promedio de precisión a IoU $\geq$ 0.5): 0.987, lo que muestra que el modelo tiene un desempeño sobresaliente en la detección con un umbral de intersección sobre unión (IoU) de al menos el 50%.

mAP50-95 (promedio de precisión a IoU $\geq$ 0.5 y 0.95): 0.742, que es una métrica más exigente y muestra que, aunque el modelo sigue siendo eficiente, puede mejorar en términos de precisión en detecciones más estrictas, especialmente para clases con menos ejemplos o características más complejas.

Clases con mAP50-95 bajo:

- Success (0.608) tiene el mAP50-95 más bajo, lo que indica que, a pesar de tener una precisión perfecta (P = 1), el modelo tiene dificultades para lograr una buena precisión en detecciones más exigentes (IoU más alto).
- Hardworking (0.685) también tiene un mAP50-95 relativamente bajo, lo que refuerza la idea de que esta clase es difícil de detectar de manera precisa, especialmente con umbrales de IoU más altos.

5.4.1.3. Matriz de confusion

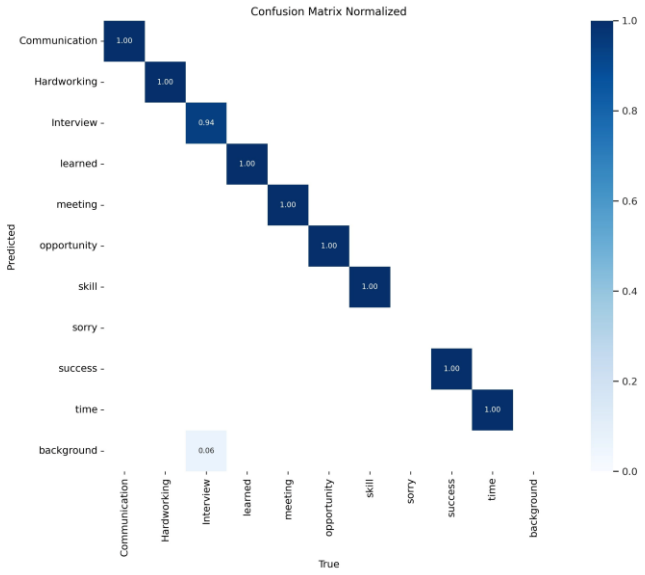

Figura 59 Matriz de confusión modelo inicial

En la *Figura 59* podemos observar la matriz de confusión normalizada en la que se puede observar:

Clases correctamente clasificadas:

- Communication, Meeting, Opportunity, Skill, Success, Time, y Learned son todas perfectamente clasificadas (con un valor de 1 en la diagonal).

Clases con confusión:

- Interview tiene una confusión del 6% con Hardworking. Esto sugiere que el modelo está confundiendo a menudo estas dos clases, lo que puede deberse a similitudes en las imágenes o características visuales comunes.
- Sorry no aparece al no tener imágenes en la carpeta valid.

5.5. Mejoras implementadas

5.5.1. Data augmentation

Para el aumento de datos en nuestro modelo, utilizamos Albumentations.

Albumentations es una potente librería de Python diseñada para realizar aumentación de datos en imágenes, ampliamente utilizada en tareas de visión por computadora como clasificación, segmentación y detección de objetos. Ofrece un conjunto extenso de transformaciones, como rotaciones, ajustes de color, ruido, desenfoques, flips, y más, las cuales se pueden combinar en pipelines flexibles mediante *A.Compose.*

Albumentations destaca por su rapidez, facilidad de uso y compatibilidad con frameworks como PyTorch y TensorFlow. Además, incluye transformaciones avanzadas como cortes de imagen, distorsiones geométricas y manipulación de espacios de color, permitiendo crear datos variados y realistas para mejorar la generalización y robustez de los modelos entrenados.

A continuación, explicaremos las transformaciones realizadas en nuestro modelo.

```
transform = A.Compose([
    #A.HorizontalFlip(p=0.5),
    #A.VerticalFlip(p=0.5),
    A.GaussNoise(var_limit=(10.0, 50.0), p=0.5),
    A.GaussianBlur(blur_limit=(3, 7), p=0.5),
    A.RandomBrightnessContrast(brightness_limit=0.2, contrast_limit=0.2, p=0.5),
    A.RandomGamma(gamma_limit=(80, 120), p=0.5),
    A.ISONoise(color_shift=(0.01, 0.05), intensity=(0.1, 0.5), p=0.5),
    A.ToGray(p=0.5),
    A.HueSaturationValue(hue_shift_limit=20, sat_shift_limit=30, val_shift_limit=20, p=0.5)
])
```

Figura 60 Albumentations

Utilizamos *A.Compose*, que permite agrupar varias transformaciones y aplicarlas de manera secuencial sobre imágenes o lotes de datos. Cada transformación tiene un parámetro p, que indica la probabilidad de que la transformación se aplique.

5.5.1.1. Transformaciones no recomendadas

Estas transformaciones nos podrían resultar contraproducentes ya que hay determinados signos como success que al ser rotado cambiaría su significado, por lo que destacamos que no serán utilizadas.

- `A.HorizontalFlip(p=0.5)`
  - Realiza una inversión horizontal de la imagen con una probabilidad del 50%.
- `A.VerticalFlip(p=0.5)`
  - Realiza una inversión vertical de la imagen con una probabilidad del 50%.

5.5.1.2. Transformaciones realizadas

1. `A.GaussNoise(var_limit=(10.0, 50.0), p=0.5)`
   - Añade ruido gaussiano a la imagen con una varianza aleatoria entre 10.0 y 50.0.
   - Se utiliza para simular condiciones de ruido en las imágenes, mejorando la robustez del modelo frente a datos ruidosos.
   - Probabilidad de aplicación: 50%.
2. `A.GaussianBlur(blur_limit=(3, 7), p=0.5)`
   - Aplica un desenfoque gaussiano a la imagen. El parámetro blur_limit especifica que el kernel de desenfoque será un tamaño aleatorio entre 3x3 y 7x7.
   - Útil para suavizar imágenes o simular efectos de movimiento/borrosidad.
3. `A.RandomBrightnessContrast(brightness_limit=0.2, contrast_limit=0.2, p=0.5)`
   - Ajusta el brillo y el contraste de la imagen.
     - brightness_limit=0.2: Aumenta o disminuye el brillo en un 20%.
     - contrast_limit=0.2: Aumenta o disminuye el contraste en un 20%.
   - Simula condiciones de iluminación variables.
4. `A.RandomGamma(gamma_limit=(80, 120), p=0.5)`
   - Cambia la gama de la imagen. El valor de gama se elige aleatoriamente entre 80 y 120.
   - Ayuda a modificar la claridad de la imagen ajustando la intensidad no lineal de sus píxeles.

5. `A.ISONoise(color_shift=(0.01, 0.05), intensity=(0.1, 0.5), p=0.5)`
    - Simula el ruido típico de cámaras digitales.
        - color_shift: Variación de color debido al ruido.
        - intensity: Intensidad del ruido.
    - Útil para entrenar modelos a manejar datos capturados con ruido ISO.
6. `A.ToGray(p=0.5)`
    - Convierte la imagen a escala de grises con una probabilidad del 50%.
    - Reduce la información del color, útil para trabajar en entornos monocromáticos.
7. `A.HueSaturationValue(hue_shift_limit=20,sat_shift_limit=30, val_shift_limit=20, p=0.5)`
    - Ajusta el matiz (hue), saturación (sat) y valor (val) de los colores de la imagen.
        - hue_shift_limit=20: Cambios en el matiz, en un rango de ±20.
        - sat_shift_limit=30: Cambios en la saturación, en un rango de ±30.
        - val_shift_limit=20: Cambios en el brillo, en un rango de ±20.
    - Útil para manejar variaciones de color en los datos.

Los parámetros p=0.5 asignados a cada transformación aseguran que no todas las transformaciones se apliquen siempre, introduciendo aleatoriedad en el pipeline.

Con estas transformaciones mejoramos la robustez del modelo ante variaciones de iluminación y contraste, diferentes niveles de ruido, transformaciones de color, cambios de desenfoque o escala de grises.

5.5.1.3. Estudio dataset aumentado

5.5.1.3.1. Carpeta train aumentada

Clase	Imágenes en train	Imágenes en valid
Communication	650	320
Hardworking	425	260
Interview	540	160
Learned	460	240
Meeting	655	320
Opportunity	415	200
Skill	740	380
Sorry	500	200
Success	720	360
Time	580	280

Tabla 6. 3 Distribución de clases en train aumentado

Como se puede observar en la *Tabla 6.3*, con el conjunto aumentado, todas las clases cuentan ahora con un número mucho mayor y más equilibrado de imágenes, tanto en entrenamiento como en validación. Por ejemplo, *Sorry* pasa de 3 a 700 imágenes (train + valid), y otras clases antes subrepresentadas, como *Opportunity* o *Hardworking*, superan las 600. Esto mejora significativamente la capacidad del modelo para aprender de todas las clases y generalizar mejor.

5.5.1.3.2. Análisis de la distribución tras el aumento

Carpeta	Nº imágenes	Porcentaje
Train	5950	66’12%
Valid	2935	32,61%
Test	114	1’27%
Total	8999	100%

Tabla 6. 4 Distribución dataset aumetado

Entrenamiento (Train)

5950 imágenes, que representan el 66.12% del total de datos, siguen siendo la mayor parte del conjunto de datos,

lo cual es típico en la mayoría de los problemas de aprendizaje automático, ya que la mayoría de las imágenes se usan para entrenar el modelo.

Esta cantidad es considerablemente grande y debería ser suficiente para permitir que el modelo aprenda representaciones robustas, incluso si algunas clases aún están desbalanceadas. Asegúrate de que las clases minoritarias estén adecuadamente representadas durante el entrenamiento para evitar sesgos en el modelo.

Validación (Valid)

2935 imágenes (32.61% del total) ahora están en la carpeta de validación. Este aumento en el número de imágenes es bastante significativo, lo que ofrece una mejor evaluación del modelo al tener más datos para validar durante el entrenamiento.

Tener un 32.61% del total de imágenes en validación es adecuado para obtener una medición precisa del rendimiento del modelo, sin ser demasiado pequeño para generar una evaluación sesgada.

Prueba (Test)

114 imágenes (1.27% del total) se encuentran en la carpeta de test. Este número sigue es bajo en comparación con el de entrenamiento y validación. Sin embargo, este conjunto sigue siendo útil para obtener una evaluación final del rendimiento del modelo en datos no vistos.

Aunque es posible que el conjunto de test no sea tan grande como los conjuntos de entrenamiento y validación, su tamaño es suficiente para medir el rendimiento en datos desconocidos, siempre y cuando las métricas sean calculadas correctamente. En este conjunto no hago data augmentation para preservar la realidad de las imágenes.

Total de imágenes

El total de 8999 imágenes es un buen número, ya que asegura que el modelo tiene suficientes datos para entrenarse y validarse adecuadamente. Sin embargo, la distribución actual muestra que el porcentaje de imágenes en el conjunto de test es relativamente pequeño en comparación con los otros dos conjuntos (train y valid).

Si el conjunto de test representara un porcentaje más grande del total, podría proporcionar una evaluación más robusta, pero esto no es estrictamente necesario si la validación está bien equilibrada.

5.5.2. Ajuste de hiperparámetros

El ajuste de hiperparámetros es una parte esencial en el proceso de entrenamiento de modelos de redes neuronales, especialmente en modelos avanzados de detección de objetos como YOLOv8. Este proceso no solo influye en el rendimiento y la precisión del modelo, sino que también puede determinar su eficiencia en términos de tiempo de entrenamiento y consumo de recursos. La optimización de los hiperparámetros adecuados es crucial para conseguir que el modelo se ajuste correctamente a los datos de entrenamiento y logre un rendimiento sobresaliente en tareas de detección de objetos en imágenes y videos.

En el caso de YOLOv8, el ajuste de tres hiperparámetros fundamentales, el optimizador, el tamaño del batch y el learning rate, tiene un impacto directo en la capacidad del modelo para aprender de los datos de manera eficiente y eficaz. Cada uno de estos hiperparámetros desempeña un papel crucial en la forma en que el modelo mejora sus predicciones durante el entrenamiento. La selección de un optimizador adecuado determina cómo el modelo actualiza sus parámetros internos, mientras que el tamaño del batch define cuántas muestras procesar de manera simultánea, afectando tanto la estabilidad del entrenamiento como el uso de memoria. Por otro lado, el learning rate, tal vez el hiperparámetro más crítico, controla la velocidad a la que el modelo ajusta sus parámetros en cada iteración, equilibrando el riesgo de sobreajuste y la necesidad de converger rápidamente.

El proceso de ajuste de estos parámetros no es trivial y requiere una comprensión profunda del modelo, los datos y los recursos computacionales disponibles. Un ajuste inadecuado puede resultar en tiempos de entrenamiento

excesivos, falta de convergencia o incluso en una baja capacidad de generalización del modelo. Por ello, encontrar la combinación óptima de estos hiperparámetros es una tarea fundamental para obtener un modelo de detección de objetos eficiente y preciso. A lo largo del proceso de experimentación, se deben tener en cuenta no solo los valores predeterminados, sino también la influencia de cada uno de estos parámetros en las métricas de desempeño, como la precisión, recall, F1 Score y pérdidas que son clave en aplicaciones reales de YOLOv8.

5.5.2.1. Ajuste de optimizadores

5.5.2.1.1. Comparativa de optimizadores

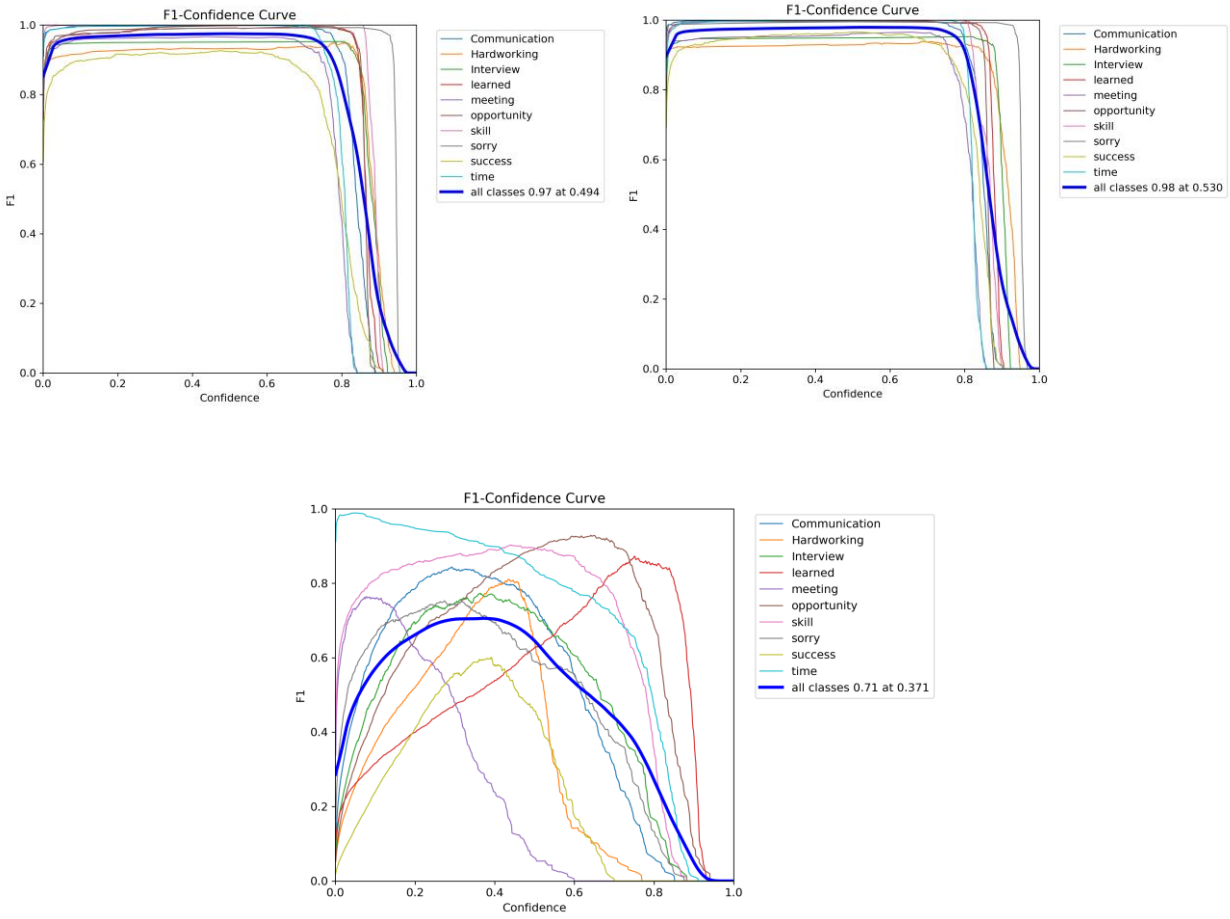

Figura 61 Comparativa F1 Score optimizadores

En la *Figura 61* se observa una clara diferencia entre los tres algoritmos evaluados:

- AdamW (gráfica superior derecha) es el que obtiene el mejor desempeño general. Alcanza un F1 Score máximo de 0.98 a una confianza de 0.53, mostrando gran estabilidad y uniformidad en el comportamiento de todas las clases. Las curvas de F1 por clase están muy agrupadas, lo que indica una generalización robusta y buen balance.
- Adam (gráfica superior izquierda) también ofrece un rendimiento notable, con un F1 Score máximo de 0.97 a una confianza de 0.49. Aunque es muy competitivo, se aprecia una ligera dispersión entre clases en comparación con AdamW, lo que podría reflejar pequeñas inconsistencias en algunas etiquetas específicas.
- RMSProp (gráfica inferior) muestra un rendimiento significativamente inferior. Su F1 Score máximo es de apenas 0.71, con una gran variabilidad entre clases y menor estabilidad a lo largo del rango de confianza. Las curvas presentan formas dispares, lo que sugiere dificultades para aprender representaciones consistentes en todas las categorías.

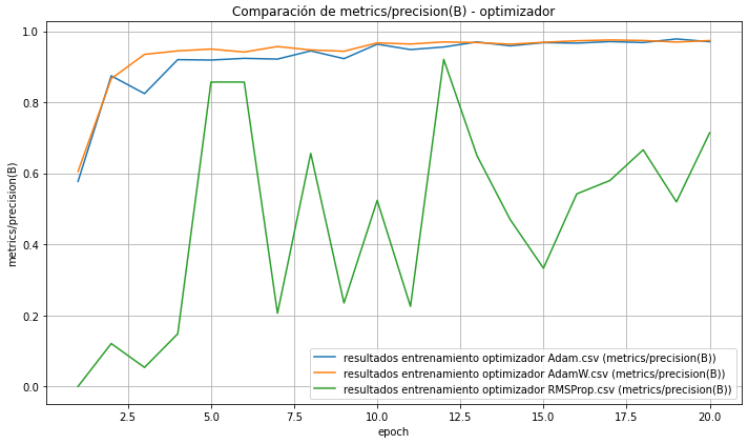

Figura 62 Comparativa de precisión de optimizadores

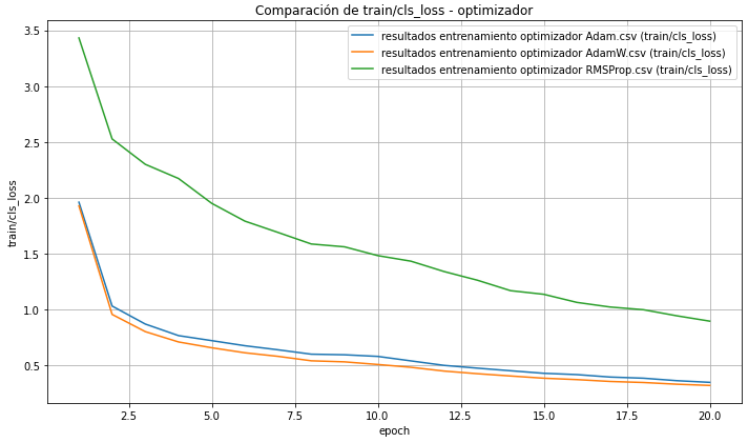

Figura 63 Comparativa de train loss de optimizadores

Como se puede observar en las *Figuras 62 y 63*, AdamW ha sido el mejor optimizador global por varias razones:

1. Estabilidad:
    - Muestra la curva de aprendizaje más suave
    - Menos oscilaciones en la precisión
    - Convergencia más rápida y estable
2. Eficiencia:
    - Alcanza menor loss con menos epochs
    - Mantiene una precisión alta de forma más consistente
    - Mejor generalización
3. Balance:
    - Encuentra un mejor equilibrio entre velocidad de convergencia y estabilidad
    - Mantiene un rendimiento consistente a través de diferentes métricas

6.4.2.1. Ajuste de tamaño de batch

Una vez elegido el optimizador AdamW, pasamos a elegir el tamaño del batch.

6.4.2.2.1. Comparativa de tamaños de batch

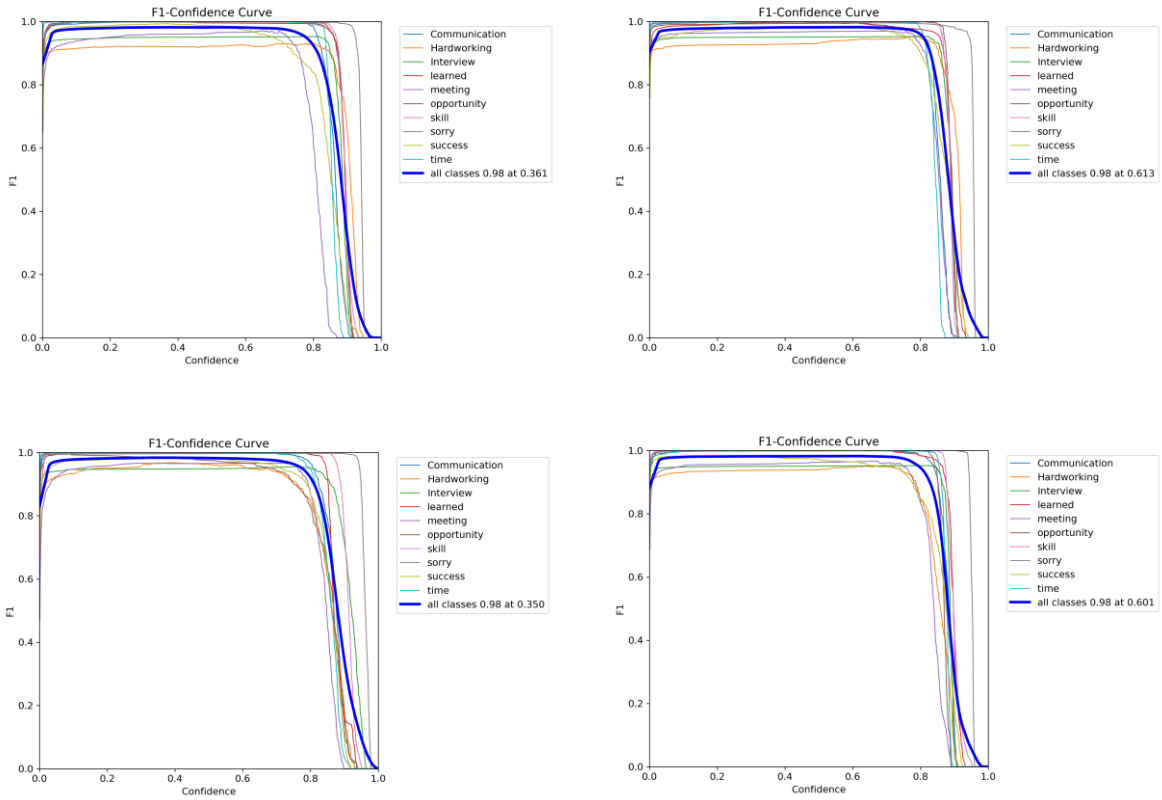

Figura 64 Comparativa tamaños de batch

En la *Figura 64* se evalúa el comportamiento del F1 Score frente a la confianza del modelo utilizando diferentes tamaños de batch: 16, 32, 64 y 128. Aunque todos los modelos alcanzan un F1 Score promedio muy alto de 0.98, existen diferencias notables en el punto de confianza en el que ese rendimiento se mantiene estable, lo que influye en la confiabilidad de las predicciones.

- Batch size 16 (arriba izquierda): El F1 Score comienza a decaer a partir de niveles relativamente bajos de confianza (alrededor de 0.36). Esto indica que el modelo realiza predicciones correctas incluso cuando no está muy seguro, pero podría ser menos confiable a altas confianzas.
- Batch size 32 (arriba derecha): Muestra una mejora clara, con el punto óptimo de confianza en 0.61. El F1 Score se mantiene alto incluso cuando el modelo exige mayor certeza para hacer predicciones, lo que indica un mejor balance entre precisión y confiabilidad.
- Batch size 64 (abajo izquierda): Similar al caso del batch 16, el punto óptimo de confianza es bajo (0.35), lo que sugiere nuevamente un modelo menos conservador en su seguridad al predecir.
- Batch size 128 (abajo derecha): Se comporta de forma muy parecida al batch 32, con un punto de confianza alto (0.60) y un F1 Score estable, lo que lo hace adecuado cuando se requiere mayor certeza en las decisiones del modelo.

En resumen, aunque el rendimiento global es excelente en todos los casos, los modelos entrenados con batches de tamaño 32 y 128 son más consistentes a niveles de confianza elevados, por lo tanto, ofrecen predicciones más seguras sin perder precisión.

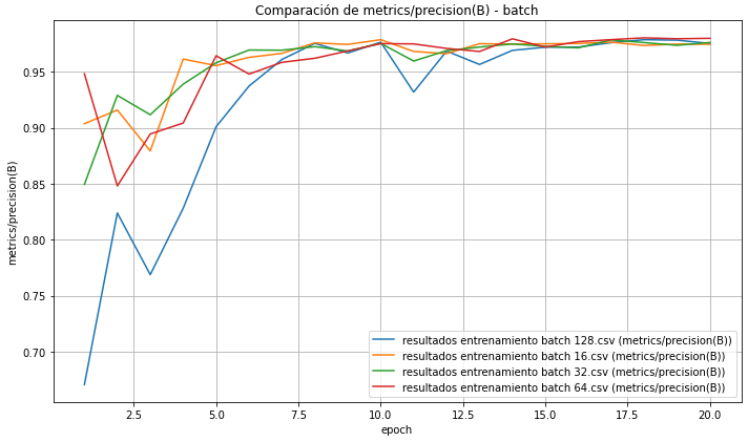

Figura 65 Comparativa de precisión por tamaño de batch

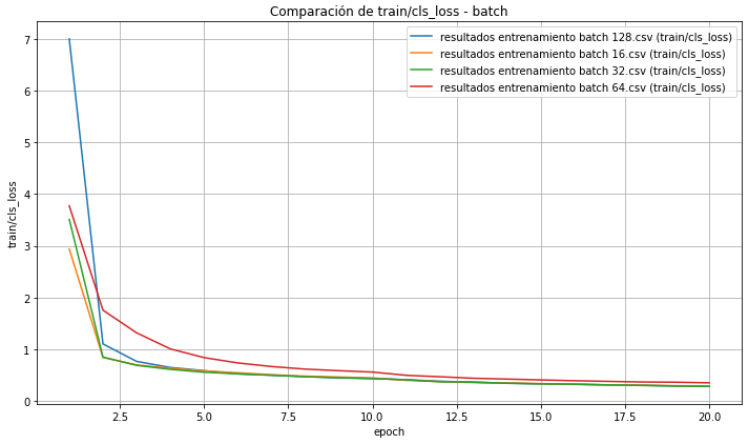

Figura 66 Comparativa de train loss por tamaño de batch

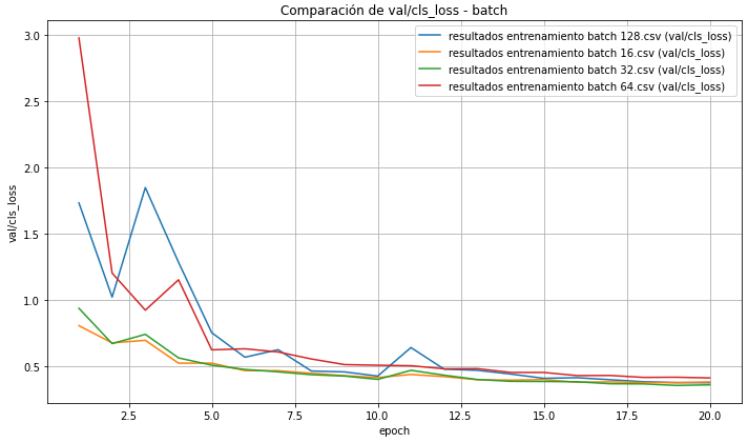

Figura 67 Comparativa de valid loss por tamaño de batch

Conclusiones de las *Figuras 65, 66 y 67*:

- Los batch 16 y 32 muestran una menor pérdida en validación en las primeras épocas, lo que podría indicar una mejor generalización inicial.
- Los batch 16, 32 y 64 alcanzan una precisión más alta en menos épocas, lo que indica que modelos entrenados con lotes más pequeños pueden aprender más rápidamente en términos de precisión
- El batch size 32 es el mejor punto óptimo en términos de estabilidad, generalización y rendimiento.
- El batch 128 no aporta mejoras significativas sobre batch 32 y podría aumentar el consumo de memoria innecesariamente.
- Batch 32 es la mejor opción.

6.4.2.3. Ajuste de learning rate

Una vez seleccionado el optimizador AdamW y tamaño de batch 32, procedemos a seleccionar el learning rate.

6.4.2.3.1. Comparativa de learning rates

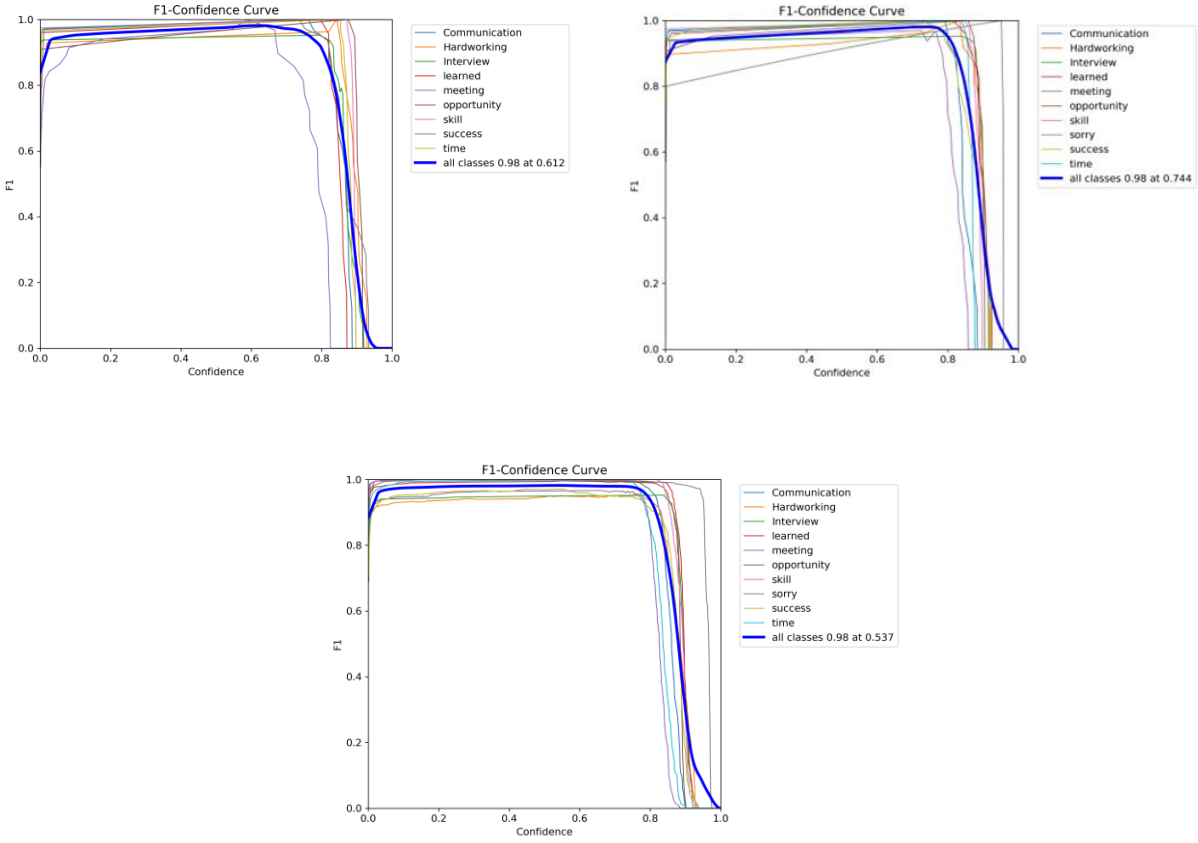

Figura 68 Comparativa de learning rate

La *Figura 68* muestra tres configuraciones de learning rate para observar su influencia en la confiabilidad de las predicciones del modelo. Aunque todas las configuraciones alcanzan un F1 score promedio de 0.98, el punto de confianza óptimo varía significativamente.

- Learning rate = 0.1 (arriba izquierda): El modelo alcanza su mejor F1 score en una confianza de

0.612, lo cual es un buen equilibrio entre rendimiento y seguridad. El F1 score se mantiene relativamente alto en un amplio rango de confianza.

- Learning rate = 0.01 (arriba derecha): Esta configuración ofrece el mejor resultado en términos de confiabilidad, alcanzando el F1 óptimo a una confianza más alta (0.744). Esto sugiere que el modelo es más conservador y solo predice con alta seguridad cuando tiene mayor certeza, lo que refuerza la fiabilidad de sus decisiones.
- Learning rate = 0.001 (abajo): Aunque mantiene el mismo F1 score, el punto óptimo de confianza cae a 0.537, indicando que el modelo comienza a perder precisión más temprano al aumentar la exigencia de seguridad. Esto podría reflejar un aprendizaje más lento o menos enfocado.

6.5. Resultados finales

Una vez hemos terminado de realizar el fine-tuning procedemos a realizar un entrenamiento al modelo con mejor configuración, AdamW, batch 32 y lr 0.01 como se indica en la *Figura 69*.

6.5.2. Resultados entrenamiento

```
!yolo task=detect mode=train model=yolov8s.pt data=Hand-Signs-Data---Original-2/data.yaml optimizer=AdamW batch=32 lr0=0.01 epochs=20 imgsz=640 freeze=9 plots=True
```

Figura 69 Código para el entrenamiento del modelo final

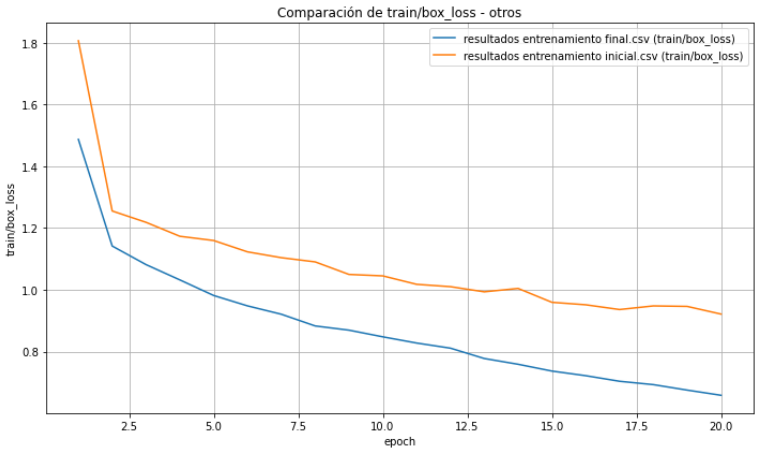

Figura 70 Train box loss

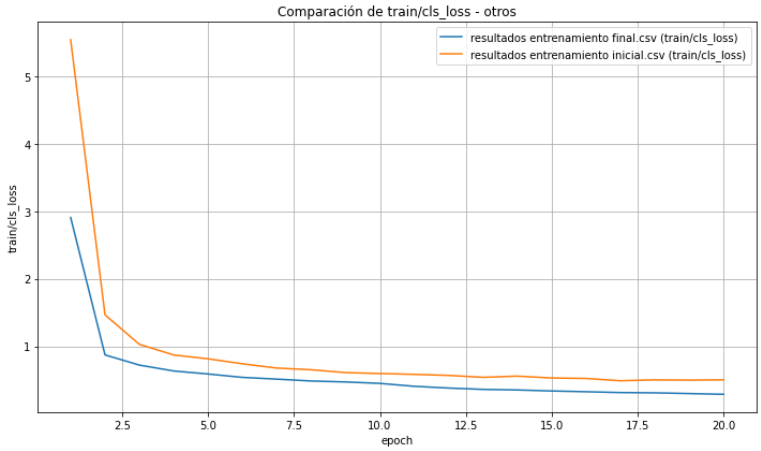

Figura 71 Train class loss

La *Figura 70* compara los resultados de entrenamientos inicial (naranja) y final (azul) mientras que la *Figura 71* muestra la evolución de la pérdida de clasificación (cls_loss), que indica como el modelo clasifica los objetos detectados.

Conclusiones

- El entrenamiento final muestra un mejor desempeño que el inicial, con pérdidas menores y una convergencia más estable.
- La pérdida en el entrenamiento final es menor en comparación con el inicial, lo que sugiere una mejora en la capacidad del modelo para clasificar correctamente.
- Las curvas sugieren que el modelo final está mejor optimizado, lo que indica mejoras en la configuración del entrenamiento (como hiperparámetros, datos o arquitectura).
- No hay signos evidentes de sobreajuste, ya que las curvas continúan descendiendo sin grandes oscilaciones.

6.5.3. Resultados validación

6.5.3.1. F1 Score

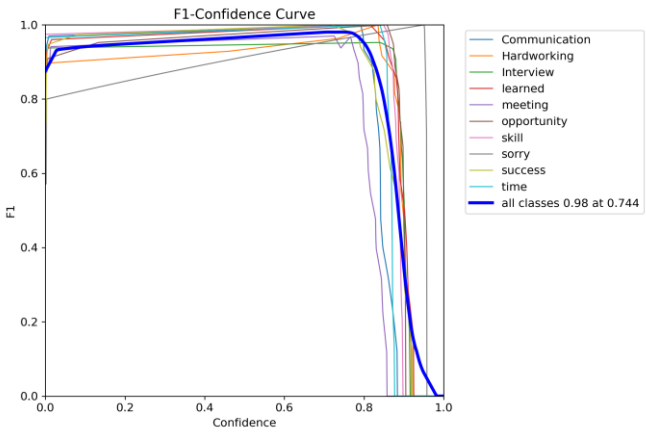

Figura 72 F1 Score modelo final

Los resultados finales son:

- F1 Score global de 0.98, lo que indica un excelente equilibrio entre precisión y recall.
- Modelo optimizado respecto a versiones anteriores, con un umbral de confianza óptimo más alto (0.744).
- Estabilidad en la mayoría de las clases, con F1 alto en la mayoría del rango de confianza.

Este modelo final demuestra un excelente rendimiento y es adecuado para su implementación con ajustes menores según el caso de uso.

6.5.3.2. Métricas de validación

6.5.3.2.1. Precisión (P)

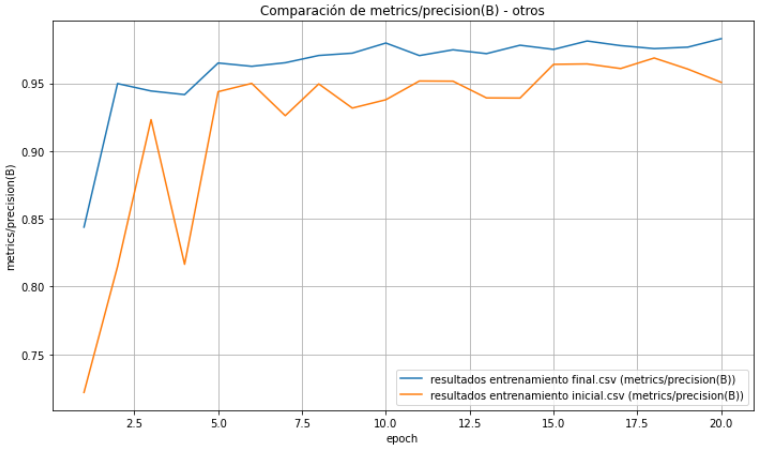

Figura 73 Comparativa de precisión

La configuración inicial tiene fluctuaciones más pronunciadas en las primeras épocas, lo que sugiere que el modelo tenía más dificultad en estabilizar su precisión.

La configuración final sube más rápido y se estabiliza en valores cercanos a 0.96–0.97, mientras que la inicial se mantiene alrededor de 0.94–0.95.

6.5.3.2.2. Recall(R)

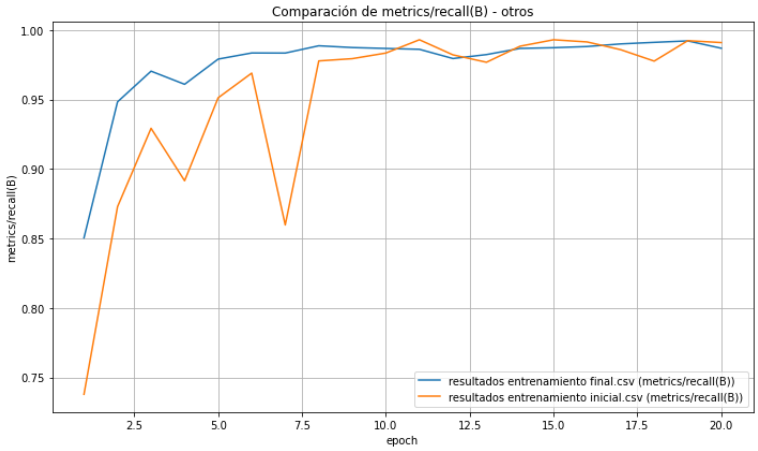

Figura 74 Comparativa de recall

El modelo inicial presenta variaciones fuertes hasta aproximadamente la época 7, lo que sugiere que no era capaz de detectar correctamente los verdaderos positivos de manera consistente.

La configuración final mantiene un crecimiento estable y alcanza valores cercanos a 1.0 desde la época 10 en adelante, lo que indica que apenas deja escapar casos positivos.

6.5.3.2.3. mAP

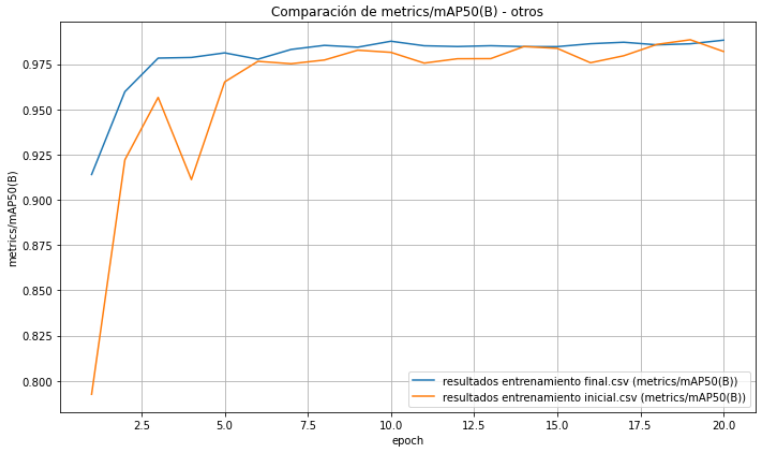

Figura 75 Comparativa mAP50

En las primeras épocas, la configuración inicial muestra más variabilidad y alcanza un plateau más tarde que la final.

Sin embargo, a partir de la época 10, ambas configuraciones están muy cerca, con valores superiores a 0.96.

Class	Images	Instances	Box(P	R	mAP50	mAP50-95):
all	2935	2720	0.983	0.987	0.988	0.78
Communication	2935	320	1	0.996	0.995	0.721
Hardworking	2935	260	0.92	0.981	0.98	0.853
Interview	2935	160	0.967	0.938	0.944	0.738
learned	2935	240	1	0.99	0.995	0.791
meeting	2935	320	0.944	0.998	0.993	0.652
opportunity	2935	200	1	0.994	0.995	0.784
skill	2935	380	0.999	1	0.995	0.87
sorry	2935	200	0.999	1	0.995	0.993
success	2935	360	1	0.975	0.995	0.624
time	2935	280	1	0.997	0.995	0.778

Figura 76 Resultados de validación modelo final

En la *Figura 76* se muestran los resultados de validación del modelo final con los siguientes resultados

El desempeño general del modelo de detección es notablemente alto, con métricas agregadas que reflejan una buena capacidad de generalización. Con un total de 2,935 imágenes y 2,720 instancias detectadas, el modelo alcanza una precisión promedio (P) de 0.983, un recall (R) de 0.987, y un mAP@50 de 0.988. La métrica más exigente, mAP@50–95, se sitúa en 0.78, lo cual indica un excelente rendimiento considerando distintos umbrales de evaluación.

Al analizar los resultados por clase, se observa que las categorías "Skill", "Hardworking" y "Success" son las mejor detectadas, con valores altos tanto en precisión como en recall, y mAP@50–95 por encima de 0.82. Estas clases demuestran una detección muy confiable y precisa.

Por otro lado, algunas clases presentan un rendimiento ligeramente inferior. Por ejemplo, "Interview" alcanza

un mAP@50–95 de 0.738, principalmente afectado por un recall más bajo en comparación con otras clases. De manera similar, la clase "Meeting" muestra el menor desempeño en esta métrica (0.652), posiblemente debido a una precisión menor (0.944) que limita su capacidad de detección con alta confianza.

En general, el modelo mantiene un equilibrio sólido entre precisión y cobertura para la mayoría de las clases. Sin embargo, las clases con menor rendimiento podrían beneficiarse de ajustes específicos, como una mayor representación en los datos de entrenamiento o un afinamiento de los umbrales de confianza, para mejorar su detección sin afectar negativamente el comportamiento global del modelo.

6.5.3.2.4. Conclusiones

Fortalezas:

- Alta precisión y recall global, lo que indica que el modelo detecta correctamente la mayoría de los objetos y comete pocos errores.
- Desempeño excelente en clases como Skill y Sorry, donde todas las instancias son detectadas y clasificadas correctamente.

Áreas de mejora:

- Success y Meeting tienen problemas de localización, lo que sugiere que el modelo puede mejorar en la precisión de los bounding boxes.
- Hardworking tiene la menor precisión, lo que indica que el modelo confunde esta clase con otras o tiene un conjunto de datos poco representativo.
- Recall bajo en Interview y Success, lo que sugiere que el modelo no detecta algunas instancias en estas clases.

6.5.3.3. Matriz de confusión

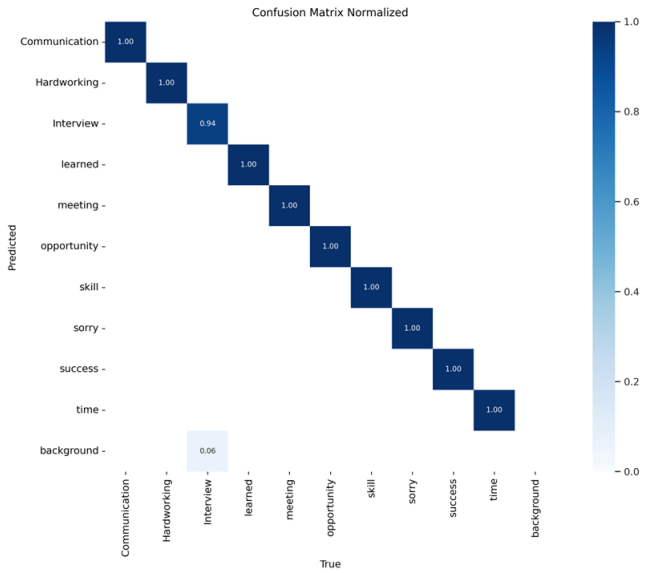

Figura 77 Matriz de confusión modelo final

Clases con alto desempeño

- Clases con clasificación perfecta (1.00 en la diagonal):
  - Communication, Hardworking, Learned, Meeting, Success, Sorry, Opportunity, Skill y Time → El modelo clasificó correctamente el 100% de las instancias de estas clases sin

confusión con otras categorías.

- Clases con rendimiento casi perfecto:
    - Interview (0.94) → El modelo detectó correctamente el 94% de las instancias, pero hubo algunos errores de clasificación con otras clases.

Errores y Confusiones

- Errores fuera de la diagonal:
    - Interview (0.94 en la diagonal): El 6% de las instancias verdaderas de esta clase fueron clasificadas incorrectamente.

Conclusiones y Recomendaciones

- Fortalezas:
    - Clasificación precisa en la mayoría de las clases, con valores de 1.00 en muchas categorías.
    - Bajas tasas de confusión en general, lo que indica que el modelo está bien entrenado.
- Áreas de mejora:
    - Mejorar detección de Interview, ya que presenta algunos errores de clasificación.
    - Reducir confusiones con el fondo (background) en Interview.
    - Revisar si los datos de entrenamiento incluyen suficientes ejemplos difíciles o variados de estas clases para mejorar la generalización del modelo.

6.5.4. Resultados test

```
!yolo task=detect mode=predict model=best.pt source=test/images save=True


!yolo task=detect mode=val model=best.pt data=data.yaml split=test
```

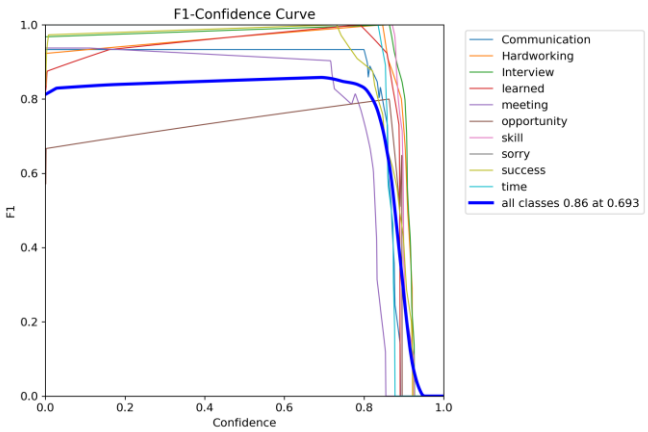

Figura 78 F1 Score Test

La curva principal de la *Figura 78* muestra que el F1 Score global alcanza un máximo de 0.86 cuando el umbral de confianza está en 0.693.

Para algunos valores bajos de confianza, el F1 Score es menor, lo que indica que el modelo comete más errores cuando se aceptan predicciones con menor certeza.

A medida que el umbral de confianza aumenta, el F1 Score tiende a disminuir en algunas clases, lo que indica que el modelo se vuelve más conservador y omite predicciones.

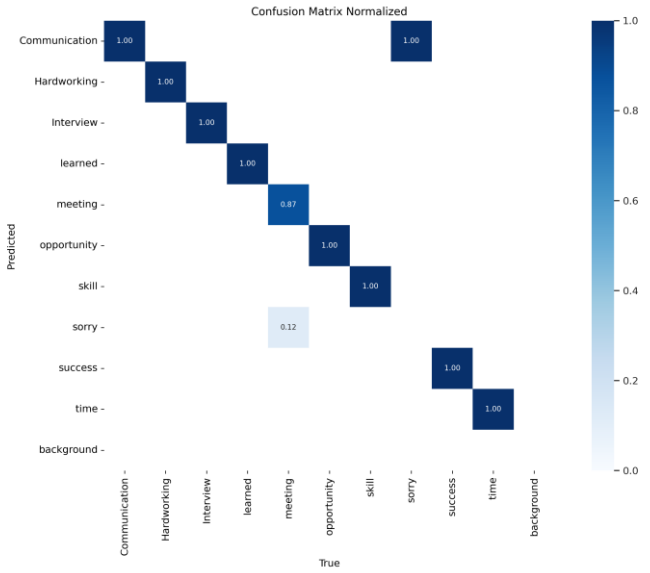

Figura 79 Matriz de confusión normalizada test

Como se observa en la *Figura 79* la mayoría de las clases presentan una precisión del 100% en la diagonal principal, indicando que las predicciones del modelo son correctas para la mayoría de los casos.

Hay un pequeño error en la clase meeting, donde el valor en la diagonal es 0.87, lo que sugiere que el 13% de las instancias de esta clase fueron clasificadas erróneamente.

La clase sorry tiene una precisión baja (0.12), lo que sugiere problemas en su detección.

Class	Images	Instances	Box(P	R	mAP50	mAP50-95):
all	114	107	0.836	0.888	0.872	0.697
Communication	114	14	0.875	1	0.967	0.818
Hardworking	114	6	0.97	1	0.995	0.897
Interview	114	15	0.986	1	0.995	0.863
learned	114	7	0.976	1	0.995	0.816
meeting	114	16	0.934	0.88	0.951	0.632
opportunity	114	2	0.629	1	0.828	0.645
skill	114	18	1	1	0.995	0.9
sorry	114	1	0	0	0	0
success	114	18	0.995	1	0.995	0.592
time	114	10	1	1	0.995	0.803

Figura 80 Resultados test

Como se observa en la *Figura 80* el modelo tiene un mAP50 de 0.872, lo que indica un rendimiento bastante bueno en términos generales.

Las clases con mejor rendimiento incluyen:

Communication, Hardworking e Interview, con valores altos de precisión y recall.

La clase sorry tiene 0 en todas sus métricas, lo que indica que el modelo no detectó correctamente ninguna instancia de esta categoría, esto requeriría de un aumento en los datos, introduciendo y etiquetando imágenes

nuevas.

Comparación entre Precisión y Recall

Algunas clases tienen alta precisión, pero bajo recall, lo que sugiere que el modelo es muy selectivo con ellas.

Otras tienen alto recall, pero baja precisión, lo que indica que el modelo predice muchos ejemplos de esa clase, pero también genera falsos positivos.

Conclusión

La media de rendimiento es buena, pero hay clases críticas con bajo rendimiento que afectan la confiabilidad del modelo.

La métrica de mAP es buena, pero la clase sorry tiene un rendimiento pobre, lo que ha generado problemas en en nuestra aplicación en tiempo real.

Abajo podemos observar una comparativa de una imagen del conjunto de test a estudiar y el resultado que se obtiene de esta predicción. Como podemos observar el resultado es bastante bueno, con un nivel de confianza del 91%.

Figura 81 Imagen a predecir Test

Figura 82 Imagen predicción Test

En general el conjunto test ha obtenido, evidentemente, peores resultados que el conjunto de validación aunque tiene unos resultados bastante buenos, el mayor problema es la clase sorry que está subrepresentada en el conjunto test y también hemos visto que es problemática. A pesar de esto los resultados siguen siendo bastante prometedores.

7. DESARROLLO DE LA APLICACIÓN

La aplicación desarrollada implementa un sistema de detección de objetos en tiempo real utilizando el modelo YOLOv8, una de las arquitecturas más avanzadas en visión por computador para tareas de detección. Este sistema permite la identificación y localización precisa de objetos en un entorno dinámico a través de una cámara local, proporcionando resultados visuales y estadísticas relevantes en tiempo real. Su diseño se orienta tanto a la demostración de capacidades de las CNN en tareas de visión artificial como a servir de base para aplicaciones prácticas en seguridad, automatización, o análisis de datos visuales.

7.1. Descripción de la aplicación

La aplicación es un sistema de detección de objetos en tiempo real que combina técnicas avanzadas de inteligencia artificial y procesamiento de imágenes. Su funcionalidad principal es capturar imágenes en directo mediante una cámara local, procesarlas utilizando el modelo YOLOv8, y mostrar los resultados de detección en pantalla con anotaciones visuales y estadísticas. Además, incorpora un cálculo dinámico de FPS, proporcionando una indicación del rendimiento en tiempo real.

El sistema está diseñado para ser intuitivo y eficiente: las detecciones se destacan con cuadros delimitadores y etiquetas que indican las categorías de los objetos reconocidos, mientras que la interfaz muestra el número total de objetos detectados y la tasa de FPS, lo que resulta útil para evaluar el desempeño del sistema en diferentes condiciones operativas.

7.2. Tecnología utilizada

- YOLOv8: Esta versión del modelo YOLO representa un avance significativo en la detección de objetos, gracias a su arquitectura optimizada que equilibra precisión y velocidad. Utiliza redes neuronales convolucionales para identificar y localizar objetos dentro de una imagen, siendo especialmente eficiente en entornos de tiempo real.
- OpenCV: Librería fundamental en el procesamiento de imágenes y video, utilizada para capturar imágenes en tiempo real desde la cámara, manipular los fotogramas y renderizar las anotaciones visuales generadas por el modelo.
- Hardware de captura: Se utiliza una cámara local con configuración de resolución ajustable (640x480), lo que asegura un rendimiento adecuado para las tareas de procesamiento en tiempo real.
- Python: Lenguaje de programación empleado para integrar las diferentes herramientas y componentes, debido a su extensa biblioteca de soporte y facilidad para trabajar con frameworks de aprendizaje profundo.

7.3. Resultados de la aplicación

El sistema ofrece detecciones rápidas y precisas en tiempo real, destacando la capacidad del modelo YOLOv8

para manejar múltiples objetos en un solo fotograma sin comprometer el rendimiento. La tasa de FPS proporciona una métrica cuantitativa del desempeño, que varía en función de la resolución del video y las capacidades del hardware subyacente. Durante las pruebas, el modelo mostró una alta precisión para identificar objetos en escenas complejas y variadas, manteniendo una tasa de detección consistente. A continuación, se muestran varios ejemplos visuales de la aplicación en funcionamiento:

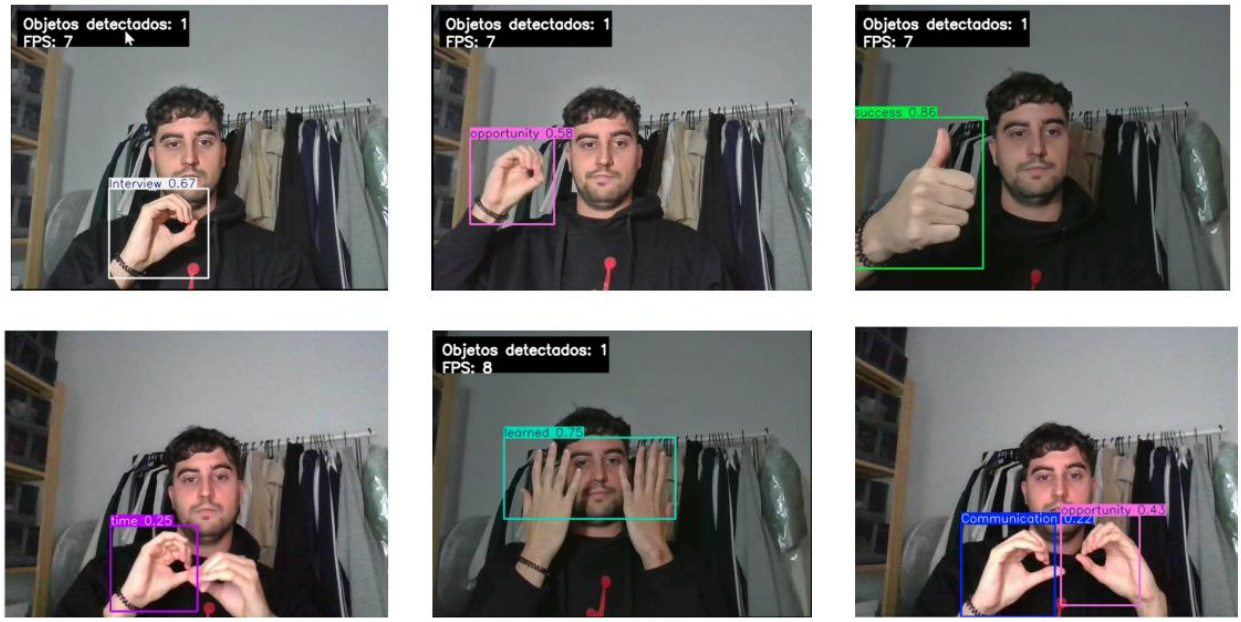

Figura 83 Ejemplo de implementación de la aplicación

7.4. Limitaciones y problemas encontrados

- Gestos problemáticos: Hay ciertos gestos como sorry o time que son una secuencia en movimiento, lo que supone un problema de detección debido a la composición en sí misma del dataset, solo compuesto por imágenes estáticas.
- Dependencia del hardware: El rendimiento del sistema, en términos de FPS, está altamente condicionado por las especificaciones del hardware, particularmente la GPU. En hardware sin aceleración por GPU, el sistema experimenta una reducción significativa en la tasa de fotogramas.
- Configuración del umbral de confianza: Si bien el umbral de confianza (conf=0.744) permite un buen balance entre precisión y detecciones falsas, su ajuste requiere pruebas exhaustivas para diferentes escenarios, ya que valores más bajos pueden generar ruido y valores más altos podrían omitir objetos relevantes.
- Entornos de baja iluminación: La calidad de las detecciones se ve afectada en condiciones de iluminación deficiente, lo que podría limitar su aplicabilidad en escenarios nocturnos o mal iluminados.
- Escalabilidad limitada: Aunque el sistema puede manejar múltiples objetos en escenas moderadamente complejas, su rendimiento podría degradarse en entornos con densidad excesiva de objetos o movimientos rápidos

8. CONCLUSIONES

En este trabajo se ha desarrollado un sistema de detección de lenguaje de signos en situaciones cotidianas utilizando el modelo YOLOv8, optimizado a través de diversas estrategias de mejora, tales como la aplicación de data augmentation, el ajuste del tamaño de batch, la tasa de aprendizaje y la selección del optimizador AdamW.

Los resultados obtenidos reflejan un rendimiento notablemente alto, con un mAP@50-95 de 0.78 para todas las clases y valores de precisión y recall superiores al 0.98 en la mayoría de los casos. La matriz de confusión normalizada muestra que las clases han sido correctamente identificadas con una tasa de error mínima, aunque se observan leves confusiones con el fondo en ciertas instancias, lo que podría indicar la necesidad de refinar el preprocesamiento de datos o ajustar aún más los umbrales de confianza.

El análisis del F1 Score evidencia que el modelo mantiene un rendimiento estable hasta umbrales de confianza cercanos a 0.8, punto en el cual se observa una caída pronunciada en la capacidad de detección. Comparando diferentes tamaños de batch, se ha identificado que un valor de 32 proporciona el mejor balance entre estabilidad y generalización, mientras que valores más altos, como 128, generan ligeras pérdidas de estabilidad y precisión.

Por otro lado, la curva de pérdida durante el entrenamiento muestra una convergencia estable, confirmando la efectividad de los ajustes realizados en los hiperparámetros. El optimizador AdamW ha demostrado ser una elección adecuada, facilitando la minimización de la pérdida sin comprometer la estabilidad del entrenamiento.

Sin embargo, es importante destacar que la obtención de un modelo óptimo en Deep Learning es un desafío altamente complejo. La interacción entre hiperparámetros, el equilibrio entre sobreajuste y generalización, y la sensibilidad del modelo a la calidad de los datos hacen que cada ajuste sea un proceso iterativo de prueba y error. Incluso pequeños cambios en la arquitectura, en la estrategia de optimización o en la distribución del dataset pueden afectar significativamente los resultados. Encontrar la combinación ideal de parámetros no solo requiere conocimiento técnico profundo, sino también un proceso meticuloso de experimentación y análisis.

En conclusión, el modelo desarrollado presenta un rendimiento sólido y una alta capacidad de generalización en la detección de signos en entornos cotidianos. No obstante, aún existen áreas de mejora, especialmente en la reducción de falsos positivos y la optimización de los umbrales de decisión, lo que plantea interesantes líneas de investigación para el futuro. Este trabajo demuestra que, aunque alcanzar el resultado óptimo en Deep Learning es una tarea difícil, el refinamiento progresivo del modelo permite obtener soluciones cada vez más robustas y eficientes.

8.1. Vías futuras de investigación

El presente trabajo ha demostrado la viabilidad de utilizar YOLOv8 para la detección de lenguaje de signos en situaciones cotidianas, así como la implementación de una herramienta en tiempo real capaz de interpretar y procesar gestos con una buena precisión. Sin embargo, aún existen diversas áreas de mejora y expansión que podrían optimizar el sistema y ampliar su aplicabilidad.

Entre las posibles líneas de desarrollo futuro se encuentran el perfeccionamiento del modelo mediante el uso de conjuntos de datos más amplios y diversificados, la integración con técnicas de procesamiento de lenguaje natural (PLN) para mejorar la interpretación del contexto y la exploración de arquitecturas más avanzadas que permitan una mayor eficiencia en dispositivos con recursos limitados.

Estos avances no solo mejorarían la precisión y usabilidad del sistema, sino que también podrían contribuir a una mayor inclusión de las personas sordas en entornos donde la comunicación mediante lenguaje de signos sigue siendo un desafío.

8.1. Propuestas de mejora

Para mejorar la precisión y la versatilidad del sistema desarrollado, se proponen las siguientes estrategias:

- Ampliación del dataset: Incorporar un mayor volumen de datos con variabilidad en iluminación, ángulos de cámara y diferentes estilos de ejecución de los gestos para mejorar la generalización del modelo.
- Integración de reconocimiento secuencial: Complementar la detección con modelos de reconocimiento de secuencias (como Transformers o LSTMs) para interpretar secuencias completas y no solo gestos individuales.
- Desarrollo de una interfaz más intuitiva: Optimizar la herramienta en tiempo real con una interfaz más accesible para usuarios sin experiencia técnica, facilitando su adopción en distintos sectores.
- Evaluación en entornos reales: Realizar pruebas en escenarios más diversos para identificar posibles limitaciones y adaptar el sistema a condiciones de uso reales.

Estas propuestas no solo mejorarían la precisión del sistema, sino que también contribuirían a su adopción en escenarios cotidianos, promoviendo una mayor inclusión de la comunidad sorda en la sociedad digital.

REFERENCIAS

[1] “Historia de la IA”, Genially, 2023

[2] Russell, S., & Norvig, P., "Artificial Intelligence: A Modern Approach." , Pearson, 2016.

[3] Goodfellow, I., Bengio, Y., & Courville, A., "Deep Learning", MIT Press, 2016.

[4] Yann LeCun, Yoshua Bengio & Geoffrey Hinton, "Deep learning", *paper*, 2015

[5] “Inteligencia Artificial”, Cisco Networking Academy, 2024

[6] “CNN Model”, Plain Concepts, 2023

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, “Attention Is All you Need”, *paper,* 2023

[8] Jeremy Jordan, “Setting the learning rate of your neural network”, 2023

[9] Raji Shanmugam , “Back Propagation Algorithm”, 2022

[10] Douglas M. Hawkins, "The Problem of Overfitting", 2003.

[11] “Data augmentation”, Albumentations, 2023

[12] F. Zhuang *et al*., "A Comprehensive Survey on Transfer Learning," in *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43-76, Jan. 2021.

[13] “Transfer Learning and Fine-Tuning a Pre-Trained DL Models”, 2023

[14] Shrey Srivastava, "Comparative analysis of deep learning image detection algorithms", *paper*, 2021

[15] Nafizul Haque, “What is CNN?”, 2019

[16] Raúl Pérez, “Clasificación de imágenes con redes profundas”, 2022

[17] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q., “Densely Connected Convolutional Networks (DenseNet)” , *paper*, 2017

[18] “Yolov8”, Ultralytics, 2023

[19] Muhammad Yaseen, "What is yolov8: an in-depth exploration of the internal features of the next-generation object detector", *paper*, 2024

[20] R. Varghese and S. M., "YOLOv8: A Novel Object Detection Algorithm with Enhanced Performance and Robustness," *2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)*, Chennai, India, 2024

[21] Nadia Ibrahim Nife, & Mohammed Chtourou. "A Comprehensive Study of Deep Learning and Performance Comparison of Deep Neural Network Models (YOLO, RetinaNet)", *International Journal of Online and Biomedical Engineering (iJOE)*, *19*(12), pp. 62–77, 2023