EVALUACIÓN DE MODELOS CONVOLUCIONALES PARA LA CLASIFICACIÓN DEL ABECEDARIO DACTILOLÓGICO DE LA LENGUA DE SEÑAS AMERICANA (ASL)

William Alberto Sinisterra Garzón

Escuela de Ingeniería Eléctrica y Electrónica
Facultad de Ingeniería
Programa de Ingeniería Electrónica
Octubre
2025

EVALUACIÓN DE MODELOS CONVOLUCIONALES PARA LA CLASIFICACIÓN DEL ABECEDARIO DACTILOLÓGICO DE LA LENGUA DE SEÑAS AMERICANA (ASL)

William Alberto Sinisterra Garzón

Documento presentado al
Programa de Ingeniería Electrónica
como requisito para optar por el título de
Ingeniero Electrónico

Director: Wilfredo Alfonso-Morales, PhD.

Codirector: Juan José Rodríguez-Rodríguez, MSc.

Escuela de Ingeniería Eléctrica y Electrónica
Facultad de Ingeniería
Programa de Ingeniería Electrónica
Octubre
2025

Aceptación

Nota de aceptación:

_____

_____

_____

_____

_____

Firma del Jurado

_____

Firma del Jurado

_____

Wilfredo Alfonso-Morales, PhD., Director

_____

Juan José Rodríguez-Rodríguez, MSc., Codirector

Cali, Octubre de 2025

Dedicatoria

Dedicado a mi familia por haber sido un importante apoyo en este camino que ha valido la pena recorrer.

A mis padres:
Solon Sinisterra Grueso (Q.E.P.D.)
Margarita Rosa Garzón Sánchez

A mis tíos y tías:
Delmer Garzón Sánchez
Vitalia Sinisterra Grueso

A mis hermanas
Mary Cenaida Guateros Garzón
Mireya Angulo Garzón
Yenny Paola Sinisterra Garzón

Agradecimientos

En mi reporte final de trabajo de grado, deseo expresar mi más sincero agradecimiento a Dios, quien ha sido mi guía y bendición en este viaje, permitiéndome acceder a la educación superior y superar las adversidades que este arduo camino ha presentado. Agradezco profundamente a mis directores de trabajo de grado, el Ingeniero Juan José Rodríguez Rodríguez y al Profesor Wilfredo Alfonso Morales, cuyo apoyo y orientación han sido fundamentales. También quiero dedicarles mis más sinceros agradecimientos a mis padres, quienes con su continua preocupación y sacrificio han hecho posible mi educación, siendo este el regalo más grande e importante que puedo recibir.

En este apartado de agradecimientos, no puedo dejar de mencionar a la persona que ha tenido un impacto significativo en mi vida y en mi trayectoria académica: Luz Aida Castillo Sinisterra. Agradezco profundamente a Dios por permitirme tener el privilegio de conocerla y aprender de ella. Luz Aida, tu influencia y orientación han sido fundamentales en mi desarrollo académico. Gracias por tus valiosos consejos y palabras de aliento que me guiaron a través de este camino desafiante y lleno de obstáculos. Tu confianza en mí ha sido un motor constante para seguir adelante y alcanzar mis metas. Tu amor y apoyo incondicional han sido la base de mi éxito. Aprecio más de lo que puedo expresar todo lo que has hecho por mí, y te estaré eternamente agradecido por ser una parte esencial de mi vida y de mi educación superior.

Glosario

AI inteligencia artificial (*Artificial Intelligence*). 2, 3, 6, 14, 18

ANN redes neuronales artificiales (*Artificial Neural Networks*). 2, 14, 19

ASL lengua de señas americana (*American Sign Language*). VII, XI, 2–4, 6, 7, 9, 14–16, 23, 25, 28, 29, 31, 32, 38, 39, 41, 52, 64–67, 69

CI inteligencia computacional (*Computational Intelligence*). 18

CNNs redes neuronales convolucionales (*Convolutional Neural Networks*). 3, 6, 14, 15, 18

DL aprendizaje profundo (*Deep Learning*). 2, 3, 6, 14, 15, 18, 19

HGR *Hand Gesture Recognition*. 2, 3

MSL lengua de señas mexicana (*Mexican Sign Language*). VII, 7, 9, 28–30, 43

Tabla de Contenido

Pág.

Lista de Figuras

FIGURA Pág.

Lista de Tablas

TABLA Pág.

Abstract

This study aims to see how well an ad hoc convolutional neural network (CNN) model can sort the signs of the American Sign Language (ASL) alphabet. Initially, we conducted an exhaustive search on Kaggle to identify an appropriate dataset that met the project's requirements. Subsequently, we performed a detailed preprocessing of the images, where dimensions were adjusted, and pixel values were normalized to ensure data consistency. Additionally, one-hot vectors were generated for the classification labels, facilitating their use in the neural network.

We set up the development environment in Kaggle using Keras with TensorFlow, leveraging the power of the GPU accelerator for efficient and rapid data processing. We designed the neural network to adapt to different numbers of ASL classes, enabling the evaluation of the model's capacity to handle various levels of classification complexity. To optimize the parameters, Optuna, a hyperparameter optimization tool, was used to explore different combinations during training efficiently. Finally, final models with optimized parameters were obtained and compared with non-optimized models in terms of performance and efficiency. This comparison highlighted the importance of optimization in the performance and efficiency of CNN models.

Keywords:– American Sign Language (ASL), Hand Gesture Recognition (HGR), Deep Learning (DL), Convolutional Neural Networks (CNN)

Resumen

Este estudio tiene como objetivo comprobar la capacidad de un modelo de red neuronal convolucional (CNN) *ad hoc* para ordenar las señas del alfabeto de la lengua de señas americana (*American Sign Language*) (ASL). Inicialmente, se realizó una búsqueda exhaustiva en *Kaggle* para identificar un conjunto de datos adecuado que cumpliera los requisitos del proyecto. Posteriormente, se realizó un preprocesamiento detallado de las imágenes, en el que se ajustaron las dimensiones y se normalizaron los valores de los píxeles para garantizar la coherencia de los datos. Además, se generaron vectores únicos para las etiquetas de clasificación, lo que facilitó su uso en la red neuronal.

Posteriormente, se configuró el entorno de desarrollo *Kaggle* utilizando *Keras* y *TensorFlow*, para aprovechar la potencia del acelerador *GPU* en el procesamiento de datos de manera eficiente y rápida. Se diseñó la red neuronal para adaptarse a diferentes números de clases de ASL, permitiendo la evaluación de la capacidad del modelo para manejar varios niveles de complejidad de clasificación. Para la optimización de hiperparámetros, se utilizó Optuna, una herramienta para explorar diferentes combinaciones durante el entrenamiento de forma eficiente. Por último, se obtuvieron modelos finales con parámetros optimizados y se compararon con modelos no optimizados en términos de rendimiento y eficacia. Esta comparación puso de manifiesto la importancia de la optimización en el rendimiento y la eficiencia de los modelos CNN.

Palabras clave:– lengua de señas americana (*American Sign Language*) (ASL), reconocimiento de gestos con la mano, aprendizaje profundo, redes neuronales convolucionales (CNN).

Capítulo 1

Introducción

Los gestos con las manos (señas) son la principal herramienta de comunicación simbólica y la forma natural en la que los humanos se expresan de manera efectiva, estos varían desde acciones simples hasta más complejas [1]. Las personas sin algún tipo de agudeza auditiva son aisladas y se crean barreras de comunicación, desarrollando un *trastorno de lenguaje* al no poder transmitir sus ideas de forma adecuada. Este grupo poblacional desarrolla ciertas habilidades de comunicación alternativas, tales como expresiones faciales, corporales y sonidos que les permiten comunicarse con los demás. Algunos de ellos se comunican a través de la Lengua de Señas, un medio de comunicación que expresa pensamientos y emociones a través de gestos realizados principalmente con las manos, utilizando una estructura con su propia gramática y léxico [1].

Los lingüistas consideran que tanto la comunicación transmitida de manera fonológica como la comunicada a través de la lengua de señas son una lengua natural, lo que indica que ambas surgieron a través de un proceso prolongado y abstracto; también consideran que éstas evolucionaron con el tiempo sin una planificación meticulosa [2].

El conjunto poblacional que hace uso de la lengua de señas, como medio de comunicación entre las comunidades con diferentes niveles de agudeza auditiva, se encuentra principalmente constituido por las personas sordas, las que poseen algún tipo de dificultad auditiva o por personas oyentes, como las que no pueden hablar físicamente o las que tienen problemas con la comunicación hablada dominante debido a una discapacidad o afección. Aunque no se conoce con precisión la cantidad de lenguas de señas en todo el mundo, la publicación de *Ethnologue* enumera 150 lenguas de señas[1], mientras que en *SIGN-HUB Atlas* enumera más de 200 estructuras e indica

[1] `https://www.ethnologue.com/guides/how-many-languages`

que hay más, que aún están sin documentar[2]. Esto se debe a que cada país tiene su propia lengua de señas nativa y algunos tienen más de una.

Por otro lado, el reconocimiento de los gestos con la mano, conocido como *Hand Gesture Recognition* (HGR), es una línea de investigación que se especializa en el desarrollo de aplicaciones donde existe una interacción entre el ser humano y la computadora. La razón principal es que casi el 65 % de la comunicación humana se asocia a gestos [3]; dentro de las técnicas HGR se encuentran dos categorías: *gestos estáticos* y *gestos dinámicos*. En los estáticos no se cuenta con cambios en la postura del brazo y la mano; por tal motivo, solo es necesario un fotograma para su procesamiento. En el caso de los dinámicos, el brazo y la mano están en continuo movimiento, y para obtener un gesto se captura una secuencia de múltiples fotogramas que se sintetizan en un vídeo.

A pesar de su constante desarrollo, HGR es un área de investigación que ha sido poco explorada; sus avances pueden facilitar el procesamiento automático en el reconocimiento de la lengua de señas y en aplicaciones tales como el control de dispositivos en interfaces humano-computadora, aplicaciones en el sector automotriz, electrónica de consumo, transporte público, juegos, domótica e incluso traductores. Todas estas aplicaciones son posibles hoy en día gracias a los avances tecnológicos que permiten la manipulación y procesamiento de una gran cantidad de datos en segundos. Las redes neuronales artificiales (*Artificial Neural Networks*) (ANN), por su habilidad para manejar grandes cantidades de información y brindar resultados precisos, han podido resolver problemas complejos, ofreciendo altos niveles de precisión que por lo general son difíciles para el ser humano. Debido al rápido desarrollo de la inteligencia artificial (*Artificial Intelligence*) (AI) con técnicas basadas en aprendizaje profundo (*Deep Learning*) (DL), este trabajo busca determinar el tipo de arquitectura de redes DL que, aplicado al reconocimiento de gestos estáticos en el alfabeto dactilológico de la lengua de señas americana (*American Sign Language*) (ASL), llene el vacío de comunicación en el paradigma HGR, donde se implementen modelos de aprendizaje de extremo a extremo (*end-to-end*); es decir, se busca que el modelo comprenda el entorno y lo interprete de manera coherente.

[2] `https://www.sign-hub.eu/sign-language/main`

Con la diversidad de movimientos con las manos y expresiones faciales presentes en la lengua de señas, los avances en las tecnologías basadas de técnicas AI permitirían una mayor inclusión social de las personas que poseen algún problema de comunicación, tales como aquellos con problemas de agudeza auditiva o dificultad para establecer una interacción en ciertos entornos sociales. Este trabajo presenta un sistema de clasificación de gestos estáticos pertenecientes al alfabeto de la lengua de señas americana (*American Sign Language*) (ASL), usando técnicas dentro del campo de la AI, especialmente aquellas relacionadas con el DL. El desarrollo de este trabajo busca servir de eje central de implementación para que futuros trabajos de grado continúen investigando sobre esta temática, relativamente nueva, que ha sido foco de estudio por la comunidad científica internacional y que no cuenta con avances importantes dentro de la comunidad latina y la Universidad del Valle, en general.

Comprender el uso y la implementación de modelos de redes neuronales en el campo del aprendizaje profundo, como las redes neuronales convolucionales (*Convolutional Neural Networks*) (CNNs), permite identificar sus potencialidades hacia otros tipos de problemas en el campo de la ingeniería. Aunque ya se han planteado soluciones como, por ejemplo, [4] y [5], en el mismo *corpus* en que se enfoca este trabajo, es decir, gestos estáticos del ASL, este trabajo planteó el siguiente interrogante:

¿Cuál es el modelo de red neuronal convolucional (CNN) óptimo para la clasificación de las 26 letras del abecedario dactilológico en la lengua de señas americana (*American Sign Language*) (ASL), considerando aspectos como la robustez y eficiencia computacional?

La mayoría de las implementaciones realizadas en el campo del HGR están orientadas hacia el reconocimiento de *gestos estáticos* en el alfabeto ASL para realizar el proceso de extracción de características y clasificación de imágenes. Sin embargo, no se ha explorado configuraciones más simples con desempeños similares y con bajo uso de recursos computacionales. Al evaluar el rendimiento de un modelo *CNN* simple *ad-hoc*, no solo se busca mantener la exactitud en la clasificación de los gestos estáticos de la ASL, sino que también se busca identificar los mejores hiper-parámetros que maximicen la robustez y su eficiencia computacional. Esto busca tener un impacto significativo en la accesibilidad y la comunicación efectiva entre personas sordas y oyentes, a través de modelos simples que puedan integrarse en equipos portátiles como celulares y tabletas.

1.1. Objetivos

1.1.1. Objetivo general

Determinar la mejor arquitectura de modelos convolucionales para la tarea de clasificación de las 26 letras del abecedario dactilológico en la lengua de señas americana (*American Sign Language*) (ASL), comparando su desempeño en términos de robustez y eficiencia computacional.

1.1.2. Objetivos específicos

- Seleccionar y pre-procesar una base de datos de la lengua de señas americana (*American Sign Language*) (ASL) para implementar modelos convolucionales en la tarea de clasificación de las 26 letras del abecedario dactilológico.
- Implementar un modelo de clasificación simple ad-hoc para el reconocimiento del alfabeto en la lengua de señas americana (*American Sign Language*) (ASL), utilizando modelos CNN y reconocimiento por clasificación.
- Optimizar los hiper-parámetros de los modelos CNN para mejorar la cantidad de parámetros del modelo, eficiencia computacional y el costo computacional en la clasificación de las 26 letras del abecedario dactilológico en la ASL.
- Evaluar los modelos optimizados con métricas de desempeño computacional, en términos de precisión, tiempo de entrenamiento, tiempo de inferencia, identificando la mejor arquitectura que maximice la robustez y eficiencia computacional.

1.2. Hallazgos

Esta investigación analizó la capacidad de los modelos de redes neuronales convolucionales (CNN) ad-hoc para clasificar los gestos estáticos del alfabeto dactilológico de la lengua de señas americana (*American Sign Language*) (ASL). Se

evaluaron diferentes arquitecturas CNN, considerando su robustez y eficiencia computacional. Un aspecto destacado fue la selección y preprocesamiento de un conjunto de datos adecuado, que incluyó imágenes del ASL ajustadas a 128x128 píxeles en escala de grises y con etiquetas codificadas mediante *one-hot encoding*. Este enfoque garantizó la uniformidad en los datos y facilitó su uso en modelos de aprendizaje profundo.

El desarrollo de modelos de redes neuronales CNN ad-hoc demostró que, incluso con configuraciones relativamente simples, es posible alcanzar altos niveles de precisión en la clasificación. La optimización de hiper-parámetros mediante Optuna jugó un papel clave, permitiendo la exploración eficiente de combinaciones de capas, filtros y tasas de aprendizaje. Por ejemplo, la exactitud del modelo de 26 clases aumentó del 95,11 % al 99,45 %. Esta mejora se refleja en las matrices de confusión optimizadas, donde se evidencia una reducción notable de los errores de clasificación, demostrando que la optimización no solo mejoró el rendimiento, sino que también aumentó la robustez del modelo.

Cabe destacar que la incorporación de optimizadores más especializados, como *RMSprop* y *SGD*, en las versiones optimizadas favoreció la estabilidad y la convergencia durante el entrenamiento, contribuyendo a modelos más robustos y escalables. En conjunto, los resultados demuestran que la optimización permite obtener un mejor desempeño computacional sin un mayor costo en recursos, lo cual resulta especialmente relevante para aplicaciones en entornos de producción con limitaciones de hardware. Por otro lado, al concentrar el análisis únicamente en el caso de $k = 1$, los valores de *F1-score* se mantuvieron estables y cercanos a 1 en la mayoría de configuraciones, confirmando su coherencia con las métricas de exactitud y *top-k accuracy*. Este resultado indica que, cuando se evalúa la predicción más probable, los modelos logran un alto equilibrio entre precisión y *recall*. Sin embargo, al aumentar el valor de k, se introducen falsos positivos adicionales que afectan la precisión y, en consecuencia, reducen el valor del *F1-score*.

Estos hallazgos subrayan el potencial de las CNN *ad-hoc* para aplicaciones de reconocimiento de señas, promoviendo la inclusión social y facilitando la comunicación entre personas sordas y oyentes. Asimismo, sientan las bases para investigaciones futuras orientadas a la implementación de sistemas más robustos y eficientes en este

campo.

1.3. Organización del Documento

El presente trabajo de grado se estructura en cinco capítulos que abordan de manera progresiva y detallada la evaluación de arquitecturas convolucionales para la clasificación del abecedario dactilológico de la lengua de señas americana (*American Sign Language*) (ASL). El Capítulo 1 presenta la introducción que establece el contexto, justificación y objetivos de la investigación. El Capítulo 2 profundiza en el marco teórico necesario para comprender las redes neuronales convolucionales (*Convolutional Neural Networks*) (CNNs) y su aplicación en la clasificación de imágenes estáticas en el contexto de la ASL; este capítulo cubre un marco de antecedentes y el marco conceptual de cada uno de los elementos que se requieren para comprender el problema que se pretende resolver, haciendo uso de implementaciones en software de técnicas de inteligencia artificial (*Artificial Intelligence*) (AI) y más específicamente en el campo del aprendizaje profundo (*Deep Learning*) (DL). El Capítulo 3 detalla la metodología seguida para implementar y evaluar los modelos convolucionales, incluyendo la construcción de modelos *ad-hoc*. El Capítulo 4 presenta los resultados obtenidos, evaluando el rendimiento de los modelos en términos de exactitud, velocidad de inferencia y eficiencia computacional, y discutiendo las ventajas y limitaciones de cada enfoque. Finalmente, el Capítulo 5 resume las conclusiones principales y propone posibles extensiones y recomendaciones para futuras investigaciones y aplicaciones prácticas en el campo de la clasificación de la ASL.

Capítulo 2

Marco Teórico

2.1. La Lengua de Señas y su relación con la dactilológia

La lengua de señas es un medio de comunicación que usa una estructura compleja de gestos con las manos para transmitir ideas [6]. Es usado principalmente por aquellas personas que tienen o desarrollan algún tipo de agudeza auditiva; por tal motivo, no les es posible establecer una comunicación verbal dominante de su entorno. Este tipo de comunicación se convierte en un medio que permite la inclusión social y accesibilidad de las personas para reducir las brechas de comunicación que han aislado a las personas que sufren de algún tipo de pérdida auditiva [7]. La lengua de señas difiere de país a país; incluso dentro de un mismo país pueden existir distintas señas de ciudad a ciudad, por lo que existen diferentes gramáticas; actualmente, la lengua de señas americana (*American Sign Language*) (ASL) crece en popularidad, tanto en personas sordas como oyentes, ya que el inglés es muy difundido en todo el mundo y su sintaxis es muy conocida [8].

Las personas con algún nivel de dificultad auditiva tienden instintivamente a desarrollar un sistema de comunicación basado en gestos y señales visuales para interactuar con su entorno. Entre los diversos sistemas de señas en todo el mundo, la lengua de señas mexicana (*Mexican Sign Language*) (MSL) es una de las lenguas de señas más estudiadas y documentadas, que surgió principalmente de la influencia de la *Lengua de Señas Francesa FSL* en el siglo XIX, cuando Laurent Clerc y Thomas Hopkins Gallaudet establecieron la primera escuela para sordos en Estados Unidos; su estructura lingüística ha sido ampliamente investigada, lo que ha contribuido a su reconocimiento como una lengua completa y compleja [9]. No obstante, existe una

tendencia en muchas comunidades sordas a mantener sus señas locales en lugar de adoptar sistemas estandarizados, como forma de preservar su identidad cultural, lo que ha dado lugar a diferentes estructuras gramaticales en distintas partes del mundo. A pesar de esta diversidad, se ha logrado establecer un consenso internacional respecto al alfabeto dactilológico, que permite deletrear palabras mediante posiciones específicas de los dedos de una mano (la dominante) [8]. Es importante mencionar que en algunos países europeos existe un alfabeto de señas con dos manos, pero el oficial que fue aprobado por la Federación Mundial de Sordos es con una sola mano.

La lengua de señas presenta una estructura que puede categorizarse en tres tipos fundamentales de señas: estáticos, dinámicos y dinámicos continuos [10]. Las señas estáticos se caracterizan por utilizar una configuración fija de la mano sin movimiento, siendo especialmente útiles para representar letras individuales del alfabeto dactilológico. Por su parte, las señas dinámicas incorporan movimientos específicos de las manos para expresar palabras completas. La categoría más compleja corresponde a las señas dinámicas continuas, las cuales consisten en secuencias fluidas de movimientos que se entrelazan para formar oraciones completas, sin presentar límites claramente definidos entre un signo y otro. Esta diversidad en la naturaleza de las señas refleja la riqueza y complejidad de la lengua de señas, lo que hace necesario desarrollar diferentes estrategias computacionales para lograr un reconocimiento efectivo de cada tipo de señas.

Los sistemas de reconocimiento de señas se clasifican en tres categorías principales: aquellos que dependen de hardware especializado, como guantes con sensores [8], que desarrollaron una interfaz para el reconocimiento de señas con una mano y la enseñanza del alfabeto), o dispositivos como Kinect [11]; cuya tecnología subyacente facilita la captura de profundidad); los que utilizan técnicas de visión artificial o aprendizaje profundo [4][5][10][12]; y los enfoques híbridos que combinan ambos. Aunque el hardware especializado puede ofrecer una captura de movimiento muy precisa, a menudo resulta costoso y poco práctico para el uso cotidiano. En contraste, los métodos basados en visión artificial o aprendizaje profundo procesan imágenes para detectar y clasificar gestos sin dispositivos adicionales, incrementando su accesibilidad.

En cuanto a la cantidad y aportes específicos, los sistemas basados en visión artificial han demostrado un gran potencial. Por ejemplo, en [4] desarrollaron una red profunda

capaz de reconocer 22 formas de manos correspondientes al alfabeto de la lengua de señas americana (*American Sign Language*) (ASL) y 10 dígitos. En [5] crearon un modelo de aprendizaje profundo que clasifica 29 clases de señas, logrando una precisión del 98.67 % en la conversión de señas a texto. En [12] consiguieron una notable precisión del 99.50 % en el reconocimiento del Lenguaje de Señas Bengalí mediante transferencia de aprendizaje. Por su parte, en [10] avanzaron en la lengua de señas mexicana (*Mexican Sign Language*) (MSL) al reconocer el alfabeto extendido y señas dinámicas como "k", rrƶ "ll", alcanzando un 91.09 % de precisión.

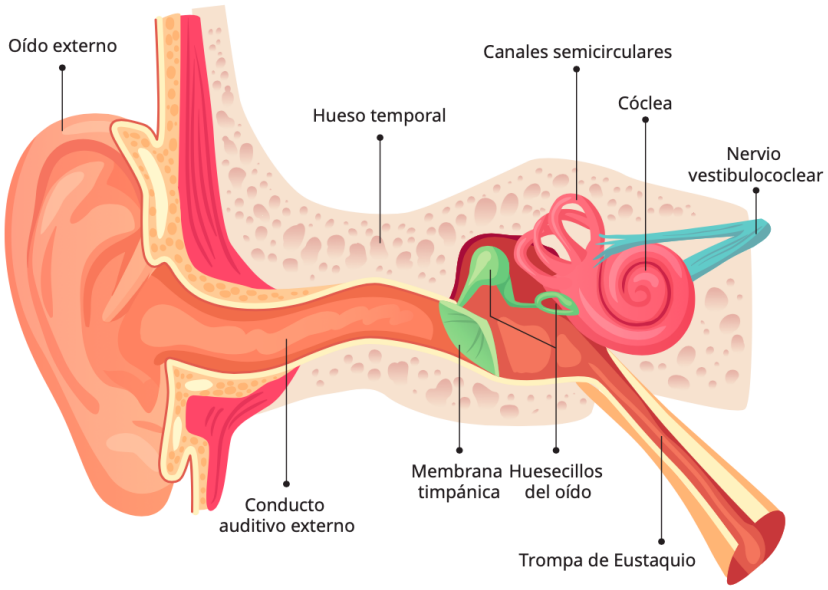
2.2. La Audición

El mecanismo de la audición es un proceso fundamental para la comunicación humana, que comienza con la captación de ondas sonoras y culmina en la percepción del sonido en el cerebro. Este proceso se desarrolla en tres etapas principales: el oído externo, el oído medio y el oído interno, tal como se presenta en la Figura 2.1.

El oído externo comienza con el pabellón auricular, que captura las ondas sonoras y las canaliza hacia el conducto auditivo externo. En este conducto, las ondas sonoras chocan con la membrana timpánica, también conocida como tímpano. La membrana timpánica, sensible a las vibraciones, comienza a vibrar en respuesta a las ondas sonoras, transmitiendo estas vibraciones al oído medio [7]. En el oído medio, las vibraciones son amplificadas y transmitidas a través de una cadena de tres huesecillos: el martillo, el yunque y el estribo, como se puede observar en la Figura 2.1. Estos huesecillos actúan como una especie de palanca, aumentando la fuerza de las vibraciones antes de transferirlas a la ventana oval, una pequeña abertura que comunica el oído medio con el oído interno. Aquí, las vibraciones son transferidas al líquido contenido en la cóclea, una estructura en forma de caracol ubicada en el oído interno [7].

La cóclea es el corazón del sistema auditivo, donde las vibraciones líquidas son convertidas en señales eléctricas. Este proceso ocurre gracias a las células ciliadas, pequeñas células sensoriales que se encuentran en la membrana basilar dentro de la cóclea. Cuando las vibraciones hacen que el líquido dentro de la cóclea se mueva, las células ciliadas se flexionan, generando impulsos eléctricos que son enviados a través

Figura 2.1: El mecanismo de la audición. Tomado de [7].



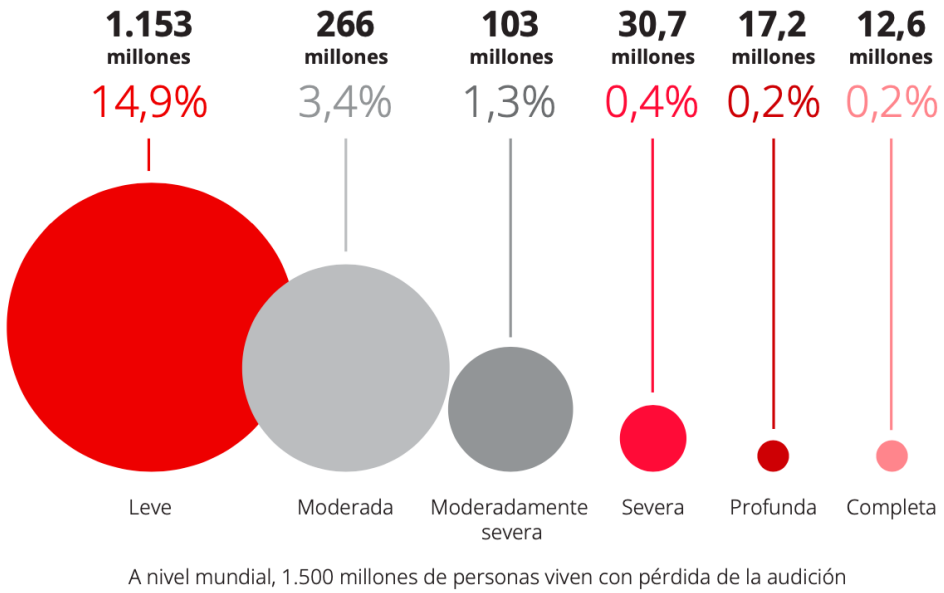
del nervio auditivo hacia el cerebro. En el cerebro, estos impulsos eléctricos son interpretados como sonido, completando así el proceso de audición [7]. Este mecanismo es esencial para la percepción del sonido y la comunicación, y su correcto funcionamiento depende de múltiples factores que pueden influir en diferentes etapas de la vida. Por ejemplo, durante los primeros años de vida, el sistema auditivo está en pleno desarrollo, lo que lo hace especialmente vulnerable a factores de riesgo como la exposición a ruidos fuertes, infecciones como la otitis media o incluso factores genéticos. Estos factores pueden tener un impacto significativo en la capacidad auditiva, especialmente si ocurren durante los períodos críticos de desarrollo [7].

A medida que envejecemos, el sistema auditivo también puede sufrir cambios degenerativos, lo que puede llevar a la pérdida auditiva relacionada con la edad. Sin embargo, esta pérdida no es simplemente un resultado inevitable del envejecimiento, sino que también puede ser influenciada por factores ambientales, modos de vida y trastornos de salud que afectan al sistema auditivo a lo largo de toda la vida. Por lo tanto, comprender el mecanismo de la audición y los factores que pueden influir en él es crucial para prevenir y tratar la pérdida auditiva en todas las etapas de la vida [7].

2.2.1. Contexto Histórico y Social de la perdida auditiva

La capacidad auditiva es un aspecto fundamental de la vida humana, ya que permite comunicarse y relacionarse con el entorno. Esta capacidad se evalúa mediante una audiometría de tonos puros, que mide los umbrales auditivos audiométricos. Cualquier disminución en la capacidad auditiva se denomina pérdida de la audición o deficiencia auditiva, y puede variar desde una pérdida leve hasta una completa, tal como se muestra en la Figura 2.2. Los efectos de la pérdida de la audición no solo dependen del grado y la naturaleza de esta pérdida, sino también en gran medida de si la deficiencia auditiva se atiende mediante intervenciones clínicas o rehabilitación.

Figura 2.2: Grados de pérdida auditiva. Tomado de [7].



Según la *Organización Mundial de la Salud (OMS)*, cerca del 5 % de la población mundial, casi 430 millones de personas sufren de pérdida auditiva alterada y necesita rehabilitación para tratar su pérdida de audición (incluidos 34 millones de niños) [1]. En el informe mundial sobre la audición publicado por la OMS se estima que para el año 2050, unos 2.500 millones de personas (1 de cada 4) sufrirán pérdida de audición, de las

[1] `https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss`

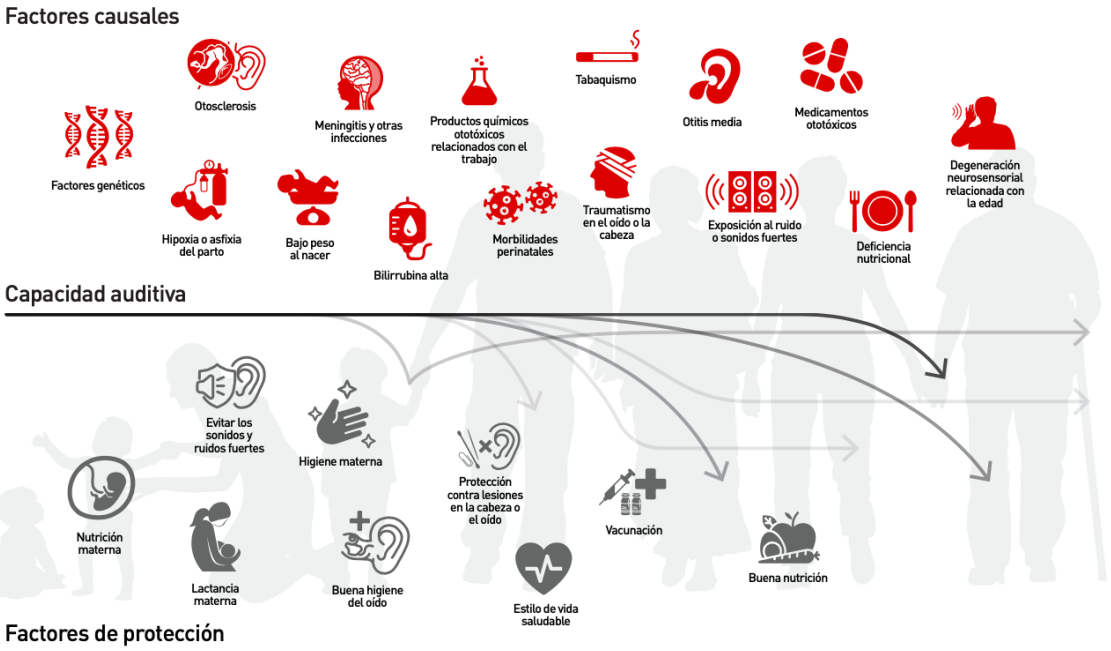
cuales casi 700 millones (1 de cada 14) padecerán pérdida moderada o superior en el oído con mejor audición [7]. Además, en el mundo, más de 1500 millones de personas sufren algún grado de pérdida de la audición [13]. Los efectos de la pérdida de la audición en una persona dependen no solo del grado y la naturaleza de esta pérdida, sino también en gran medida de si la deficiencia auditiva se atiende mediante intervenciones clínicas o de rehabilitación eficaz, y hasta en qué grado el entorno responde a las necesidades de la persona. Si no se atiende adecuadamente, la pérdida de la audición puede afectar muchos aspectos de la vida, incluyendo la comunicación, la adquisición del lenguaje y el habla, la cognición, la educación, el empleo, la salud mental y las relaciones interpersonales. La pérdida de la audición puede provocar una autoestima baja, a menudo acompañada de estigmatización, y tiene graves repercusiones en las familias y los interlocutores de quienes la padecen [7].

La pérdida de la audición no tratada conlleva un costo anual de más de 980 mil millones de dólares a nivel mundial según la OMS; esto incluye los costos relacionados con la atención de salud, la educación, las pérdidas de productividad y los costos para la sociedad [7]. La audición es un componente clave de la capacidad intrínseca del ser humano; es el sentido más necesario para comunicarse y relacionarse con los demás. Si no se atiende a tiempo, una disminución de la capacidad auditiva en cualquier momento de la vida puede afectar el funcionamiento cotidiano.

Según la OMS, la pérdida auditiva discapacitante se define como una pérdida de audición superior a 35 decibeles (dB) en el oído con mejor capacidad auditiva. Cerca del 80 % de las personas que padecen esta condición residen en países de ingresos bajos y medios. Además, la prevalencia de la pérdida auditiva incrementa con la edad, afectando a más del 25 % de las personas mayores de 60 años. En resumen, el correcto funcionamiento de la audición es esencial para la comunicación y la interacción social.

La capacidad auditiva está influenciada por una variedad de factores causales y de protección, como los mostrados en la Figura 2.3 que operan a lo largo de la vida. Los factores causales incluyen factores genéticos, bajo peso al nacer, tabaquismo, deficiencia nutricional, exposición al ruido o sonidos fuertes y degeneración neurosensorial por la edad. Estos factores pueden tener un impacto significativo en la capacidad auditiva, especialmente si ocurren durante los períodos críticos de desarrollo [7].

Figura 2.3: La audición a lo largo del curso de la vida. Tomado de [7].



Por otro lado, los factores de protección incluyen nutrición materna, lactancia materna, higiene materna, buena higiene del oído, estilo de vida saludable, vacunación y buena nutrición. Estos factores pueden ayudar a prevenir la pérdida auditiva y proteger la capacidad auditiva a lo largo de la vida según la ilustración mostrada anteriormente. La audición es crucial para comprender la importancia de la capacidad auditiva en la vida humana. La audición no solo es esencial para la comunicación y la interacción social, sino que también tiene un impacto significativo en la educación, el empleo, la salud mental y las relaciones interpersonales. La pérdida de la audición, si no se trata adecuadamente, puede tener consecuencias devastadoras en la vida de las personas, afectando su bienestar físico, mental y social, así como el de sus familias y la sociedad en general [7].

El contexto histórico y social de la audición subraya la importancia de comprender y abordar la pérdida auditiva de manera proactiva y efectiva. La capacidad auditiva es esencial para la comunicación y la interacción social, y su vulnerabilidad a diversos factores de riesgo en diferentes etapas de la vida subraya la importancia de su estudio y protección [7].

2.3. Revisión de antecedentes

Este capítulo ofrece una revisión de los principales antecedentes relacionados con la aplicación de técnicas de inteligencia artificial (*Artificial Intelligence*) (AI) orientadas al aprendizaje profundo (*Deep Learning*) (DL) para la extracción de características y la clasificación de *gestos estáticos* en el alfabeto dactilológico del lengua de señas americana (*American Sign Language*) (ASL). El capítulo también presenta una breve descripción de la población que utiliza el ASL como medio de comunicación, así como los conceptos utilizados para dar solución a esta investigación. Además, se revisa el funcionamiento y los elementos que componen las redes neuronales artificiales (*Artificial Neural Networks*) (ANN) *ad-hoc*, analizando las ventajas de cada implementación orientada a personas con algún tipo de pérdida auditiva.

El objetivo de este capítulo es contextualizar al lector en el tema bajo estudio y facilitar la comprensión del estado del arte en el reconocimiento y clasificación del alfabeto dactilológico del ASL. A continuación, se presentan diversos estudios relevantes que han contribuido significativamente al campo del reconocimiento de la lengua de señas mediante técnicas de inteligencia artificial.

En el contexto de la búsqueda de soluciones para clasificar los dígitos del 0 al 10 en la lengua de señas bengalí, en [12] se desarrollaron dos modelos pre-entrenados basados en CNNs: *InceptionV3* y *Xception*. Estos modelos fueron implementados con librerías como *TensorFlow*, *Keras*, *NumPy*, *scikit-learn* y *OpenCV* con Python 3.7 y GPU con alta RAM. Con el modelo *Xception* lograron un alcance en las métricas: precisión, exactitud, *Recall* y *F-score* del 97 %, 99,5 %, 96 % y 96 % mientras que con el modelo *InceptionV3* lograron 92 %, 97 %, 91 % y 91,5 % respectivamente. El dataset, conformado por 1075 imágenes, dividido en 925 para entrenamiento y 150 para validación, fue procesado

mediante técnicas de aumento de datos y redimensionamiento de 128×128 a 299×299 píxeles. El modelo *Xception* fue compilado con el optimizador 'Adam' y la función de costo 'categorical-crossentropy' y entrenado durante 300 épocas, demostrando la mejor precisión entre las clases, lo que lo convirtió en el modelo elegido para presentar los resultados finales. Este trabajo permite orientar esta investigación, dado que sus inicios permitieron entender la aplicabilidad de modelos CNNs preentrenados en la clasificación de señas estáticas pertenecientes a la lengua de señas bengalí.

Otro estudio presentado en [14] aplicó algoritmos de aprendizaje por transferencia pre-entrenados, específicamente VGG-16 y ResNet, para reconocer los dígitos de la ASL. Utilizando un conjunto de datos de 2180 imágenes reescaladas a 64×64 píxeles, los autores implementaron técnicas de aumento de datos para evaluar el rendimiento de los modelos. Con el optimizador Adam y un tamaño de lote de 10, probaron diferentes configuraciones de hiper-parámetros como épocas, tasas de aprendizaje y congelación de capas, encontrando que congelar todas las capas excepto las tres últimas y usar una tasa de aprendizaje de 0,1 con 50 épocas producía los mejores resultados. La exactitud alcanzada fue del 95 % para VGG-16 y 93 % para ResNet. En este trabajo se evidencia la mejora en la exactitud con el aumento de datos, ya que el desempeño de VGG-16 mejoró del 92 % al 95 % y el de ResNet del 91 % al 93 % al ser entrenados con la combinación de datos originales y aumentados.

El estudio en [15] emplea un método de segmentación basado en el color de la piel para procesar imágenes del conjunto de datos ASL *Finger Spelling benchmark*, que incluye 95.697 imágenes de 24 señas estáticas, excluyendo las letras J y Z. Estas imágenes segmentadas alimentan un modelo preentrenado de aprendizaje profundo VGG-19 para extraer características, que luego se fusiona con características locales mediante una técnica de concatenación en serie. Finalmente, una máquina de vectores de soporte (SVM) clasifica las señas, logrando una exactitud del 98,44 %, destacando la eficacia de la combinación de técnicas de segmentación y DL en la clasificación de señas.

En el trabajo [5], el modelo propuesto es capaz de reconocer el alfabeto de la ASL y convertirlo a texto simple en inglés; este alfabeto está compuesto por 24 posturas estáticas y dos posturas dinámicas. Para este trabajo se tuvo en cuenta los gestos realizados con la mano izquierda y el *dataset* tiene un total de 29 clases (26 letras,

space, *delete*, *empty*); cada clase contiene 3.000 imágenes diferentes del gesto; el dataset se dividió en conjuntos de entrenamiento y prueba con 78.300 y 8.700 imágenes en una proporción de 9:1. El modelo propuesto por los autores consiste en una CNN para el preprocesamiento y la clasificación de las imágenes: la arquitectura de la red está compuesta por dos capas convolucionales, dos *Max-Pooling* y la función de optimización *Adam*. La capa de salida tiene 29 nodos; cada uno está relacionado directamente con una clase ASL y la función de activación *SoftMax*, que arroja como resultado el valor esperado para cada clase. Durante este proceso, las imágenes se cambiaron a escala de grises, dado que estas contienen solo un sombreado; por lo tanto, es fácil de aprender para la CNN. El modelo implementado obtuvo una exactitud de entrenamiento del 98,67 %. Este estudio es relevante para esta investigación porque muestra cómo un modelo CNN ad-hoc puede ser efectivo en la clasificación de un conjunto de datos más amplio que incluye no solo letras, sino también caracteres especiales, lo que puede ser adaptado para el reconocimiento de otros tipos de señas dactilológicas.

Del mismo modo, en [4] se propone un método con una CNN que realiza tanto el proceso de extracción de características como la clasificación de las posturas estáticas de letras y números ASL. En este trabajo se entrenó una red neuronal desde cero sin aprendizaje previo y se realizó un aumento del dataset para mejorar el desempeño en el proceso de aprendizaje en casi un 20 %. La arquitectura implementada fue una CNN bastante común, la cual está compuesta por 3 bloques de 2 capas convolucionales seguidas de una capa *Max-Pooling* y una capa de *Dropout*, luego dos bloques de capas *Fully Connected (Dense)* seguidas de una capa de *Dropout* y una capa final de clasificación. Al observar el desempeño en la exactitud del sistema, primero se entrenó el modelo usando el dataset ASL, obteniendo un 82,5 % en los gestos del alfabeto y un 97 % en los dígitos. Luego se entrenó el modelo con su propio dataset, obteniendo medidas de exactitud más bajas, siendo un 67 % de exactitud en las letras del alfabeto y un 70 % en los dígitos.

La Tabla 2.1 presenta un resumen de la literatura consultada, donde se destacan los principales aportes de cada investigación.

Tabla 2.1: Resumen de antecedentes

Referencia	**Algoritmo(s) de DL empleado(s)**	**Observaciones generales**	**Resultados del mejor modelo implementado**
[12]	InceptionV3, Xception	M. R. Sadik et al. desarrollaron dos modelos de CNN, InceptionV3 y Xception, para clasificar dígitos en la lengua de señas bengalí. Utilizaron TensorFlow, Keras, NumPy, sci-kit-learn y OpenCV en Google Colab con Python 3.7 y GPU T4. El dataset de 1075 imágenes fue procesado mediante técnicas de aumento de datos y redimensionamiento. El modelo Xception, entrenado durante 300 épocas, demostró la mejor exactutud.	Exactitud = 99,50 %
[14]	VGG-16, ResNet	Alrobah N. y Selmi A. aplicaron algoritmos de aprendizaje por transferencia, VGG-16 y ResNet, para reconocer dígitos en la lengua de señas americana. Utilizaron un conjunto de datos de 2180 imágenes reescaladas a 64×64 píxeles y aplicaron técnicas de aumento de datos. Encontraron que congelar todas las capas excepto las tres últimas con una tasa de aprendizaje de 0.1 y 50 épocas producía los mejores resultados.	Exactitud = 95,00 %
[15]	VGG-19 + SVM	Rajan R y Judith Leo M. emplearon un método de segmentación basado en el color de la piel para procesar imágenes del conjunto de datos ASL Finger Spelling benchmark. Estas imágenes segmentadas alimentaron un modelo de aprendizaje profundo VGG-19 para extraer características, que luego se fusionaron con características locales artesanales mediante una técnica de concatenación en serie. Finalmente, una máquina de vectores soporte (SVM) clasificó las señas.	Exactitud = 98,44 %
[5]	CNN	Kartik et al. propusieron un modelo de CNN para reconocer el alfabeto de la Lengua de Señas Americana (ASL) y convertirlo a texto simple en inglés. El modelo consistió en una CNN para el pre-procesamiento y la clasificación de las imágenes, con una arquitectura compuesta por dos capas convolucionales, dos Max-Pooling y la función de optimización ADAM. La capa de salida tenía 29 nodos, cada uno relacionado directamente con una clase del ASL.	Exactitud = 98,67 %
[4]	CNN	Bheda et al. propusieron una CNN para clasificar posturas estáticas de letras y números en el ASL. Entrenaron la red desde cero sin aprendizaje previo y aplicaron aumento de datos para mejorar el desempeño. La arquitectura incluye 3 bloques de capas Convolucionales, Max-Pool y Dropout, seguidos de capas Fully Connected y una capa final Output.	Exactitud = 97,00 %

2.4. Inteligencia Computacional

La inteligencia computacional (*Computational Intelligence*) (CI) está relacionada con el estudio de mecanismos adaptativos que dan lugar a un comportamiento inteligente en ambientes complejos y cambiantes [16]. Dentro de la CI se pueden encontrar los siguientes paradigmas: redes neuronales artificiales (*Artificial Neural Networks* o ANN), computación evolutiva (*Evolutionary Computation* o EC), inteligencia de enjambres (*Swarm Intelligence* o SI), sistemas inmunitarios artificiales (*Artificial Immune Systems* o AIS) y sistemas difusos (*Fuzzy Systems* o FS). Las técnicas estudiadas y usadas en este trabajo forman parte del primero de estos paradigmas.

Aunque las redes neuronales son consideradas dentro de las técnicas de la AI, específicamente de *machine learning*, estas han evolucionado hacia arquitecturas más complejas, que dependen de la cantidad de capas y elementos neuronales con que se asocian dentro de lo que se conoce como DL. Esta sección presenta el funcionamiento de aquellas técnicas que, de acuerdo con investigaciones previas, se conocen que proporcionan resultados importantes, como las redes neuronales convolucionales (*Convolutional Neural Networks*) (CNNs).

2.4.1. Aprendizaje Profundo (*Deep Learning - DL*)

El aprendizaje profundo (DL) es un subconjunto del Aprendizaje Automático (*Machine Learning - ML*), donde las redes multicapa aprenden a partir de una gran cantidad de datos. La posibilidad de contar con este tipo de algoritmos aparece luego de establecerse nuevos algoritmos de aprendizaje, mejoras en las capacidades de cómputo, en los esquemas de inicialización de los pesos sinápticos de las redes de dos o más capas ocultas y la adopción de nuevos conceptos como la convolución y la recurrencia. Además, estas redes surgen de la necesidad de procesar gran cantidad de información, sin perder la capacidad de generalizar en la clasificación, para resolver el problema de estancamiento en mínimos locales que sufrían los métodos utilizados anteriormente en *ML* [17].

El DL no requiere ninguna regla diseñada por humanos para operar; más bien, utiliza una gran cantidad de datos para asignar a la entrada dada unas etiquetas específicas. DL está diseñado para utilizar numerosas capas con algoritmos en las ANN, donde cada capa proporciona una interpretación diferente de los datos que se les ha proporcionado [18].

2.4.2. Redes Neuronales Artificiales aplicadas al reconocimiento de gestos

Las redes neuronales artificiales (*Artificial Neural Networks*) (ANN), son sistemas de computación inteligentes que tienen la habilidad de aprender y generalizar a partir de un conjunto de ejemplos dados en su etapa de entrenamiento, y están inspirados en la estructura y funcionamiento de las redes neuronales biológicas que constituyen el cerebro humano[1]. Estos sistemas tienen como unidades básicas de procesamiento las neuronas, las cuales se encuentran conectadas entre sí a través de canales de comunicación, o conexiones, que cuentan con distintos pesos que están asociados con la importancia que tenga cada conexión en la red.

Cuando se emplean redes neuronales para resolver algún problema en particular, lo que se pretende es construir un sistema que pueda *aprender* una representación útil de un conjunto de datos. Para lograr esto, la red es entrenada de manera iterativa con muchos ejemplos compuestos por datos de entrada y por las salidas que se esperan obtener cuando se ingresan cada uno de estos datos; de esta manera se busca reducir la llamada *función de costo* que determina la diferencia entre las predicciones que está haciendo la red y las salidas que se deberían obtener. Hecho esto, el sistema tendrá la capacidad de predecir las salidas del mismo, de manera automática, ante un nuevo conjunto de datos similares [19]. Ejemplos de aplicación son la clasificación de imágenes, reconocimiento facial y detección de objetos, por nombrar algunos.

En una red neuronal se suelen encontrar los siguientes componentes:

- *Capas:* Son los bloques que conforman el modelo de una red neuronal. A su vez, cada capa está conformada por un conjunto de neuronas o unidades de procesamiento.

[1] `https://cis.ieee.org/about/what-is-ci`

- *Pesos de las capas:* Contienen la información aprendida por la red en la etapa de entrenamiento. Durante esta fase los pesos se actualizan repetidamente buscando minimizar una función objetivo o función de costo.

- *Predicciones:* Son las salidas arrojadas por la red neuronal. Durante la fase de entrenamiento se busca que estas predicciones se acerquen tanto como sea posible a las salidas esperadas.

- *Salidas esperadas:* Son las respuestas esperadas ante un conjunto de datos presentes en la entrada de la red. Esta información es la que usa el modelo en su etapa de entrenamiento.

- *Función de costo:* Es la encargada de definir el error que hay entre las predicciones que está haciendo la red neuronal y las salidas esperadas.

- *Error:* Es la señal de realimentación que usará el optimizador para ajustar los valores de los pesos de cada capa.

- *Optimizador:* Es el encargado de actualizar los valores de los pesos de cada capa con el fin de reducir la diferencia entre las predicciones y las salidas esperadas.

2.4.3. Perceptrón multicapa- MLP

Las redes neuronales perceptrón multicapa, o MLP por sus siglas en inglés (*Multilayer Perceptron*), son quizás los modelos más básicos que se pueden encontrar en el mundo de las redes neuronales [20]. También se conocen como redes completamente conectadas (*fully connected*) debido a la forma en que están conectadas las neuronas que conforman las diferentes capas del modelo: cada neurona de una determinada capa de la red está conectada a **todas** las neuronas presentes en la capa inmediatamente posterior. Sin embargo, no hay conexiones entre las neuronas de una misma capa. Como consecuencia, una MLP aprende características globales de los datos de entrada; en contraste, una CNN aprende características locales (esto es una propiedad relevante cuando se desea procesar imágenes).

En términos generales, la estructura de una MLP se compone de una capa de entrada encargada de recibir los datos que el modelo procesará, un número de capas ocultas

que varía de acuerdo al problema que se desea resolver y cuya tarea es procesar la información de entrada para buscar una representación adecuada de la misma, mediante la fase de entretenimiento, que facilite su posterior clasificación. Por último, se encuentra la capa de salida que mostrará los resultados finales (como clasificación de los datos en diferentes categorías). Es posible usar una MLP de manera independiente, pero también en compañía de otro tipo de arquitecturas; por ejemplo, este tipo de redes son las que se conectan a las salidas de una CNN, por lo que sus entradas serán las características que hayan sido extraídas de los datos ingresados a esta última, después de convertirlas a un formato adecuado (vector columna). También vale la pena destacar que el número de parámetros en una MLP es bastante grande en comparación con una CNN, lo cual tiene implicaciones en el tiempo que tarda el modelo en ser entrenado.

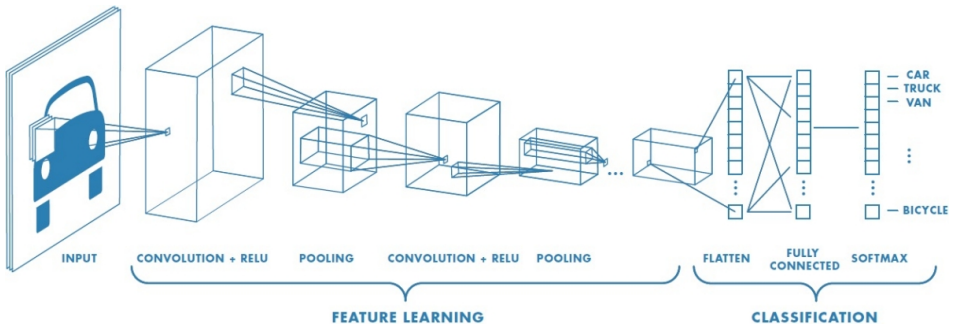
2.4.4. Redes Neuronales Convolucionales (CNNs)

Una CNN toma una imagen de entrada y asigna una importancia (mediante el uso de parámetros aprendidos en la fase de entrenamiento de la red) a varias características de ésta con el fin de clasificarla dentro de alguna categoría en particular, dependiendo del problema en cuestión, y diferenciarla de otras imágenes. En consecuencia, una CNN conserva las características del aprendizaje profundo al tomar los datos crudos y procesarlos para ejecutar diferentes etapas de extracción de características de manera autónoma durante el proceso de aprendizaje. Una ventaja importante de este tipo de redes es su capacidad para aprender dependencias espaciales y temporales entre los diferentes píxeles que conforman una imagen, lo cual es un aspecto omitido en una red neuronal tradicional MLP.

En la Figura 2.4 se presentan los componentes de una red neuronal convolucional utilizada para la clasificación de imágenes en diferentes categorías (multiclase). De manera general, las CNNs están compuestas por capas convolucionales en las que se lleva a cabo la operación matemática denominada convolución a través de filtros o *kernels* que almacenan los parámetros aprendidos durante el entrenamiento; con esta operación la red extrae características de los datos de entrada a medida que el *kernel* "recorre" cada uno de estos a lo largo de sus dimensiones. También se tienen las funciones de activación que permiten a la red aprender patrones complejos en los datos (en la CNN mostrada se usa la función ReLU, la cual es una de las funciones de activación

más comúnmente usadas). Además de la convolución, en una CNN se suele llevar a cabo otra operación que recibe el nombre de *pooling*. Su tarea es reducir el tamaño espacial de los mapas de características que se obtienen a partir de la convolución entre los datos de entrada de una capa convolucional y los *kernels* asociados, para lo cual se emplea una ventana que recorre el mapa de principio a fin y solo se toma un valor dentro de la ventana (como el valor máximo, por ejemplo) a medida que ésta se desplaza. Con el *Pooling* se logra una disminución en la capacidad de cómputo requerida para procesar la información y también que el modelo retenga las características más importantes de los datos procesados.

Figura 2.4: Estructura genérica de una CNN. Tomado de MathWorks[2].



La anterior descripción hace parte de la etapa conocida como aprendizaje de características (***Feature Learning***). Después viene la etapa de clasificación (***Classification***), en la cual se tomará la salida de la etapa 1 (matriz 2D de características aprendidas de la imagen) y, después de convertirla en un vector columna de datos (1D), se usa como entrada para una red completamente conectada que tendrá como salida la clasificación final de la red. Es interesante observar la forma en que se va reduciendo el tamaño de la imagen de entrada a medida que pasa por las diferentes capas del modelo.

La arquitectura mostrada corresponde a una red neuronal convolucional de dos dimensiones (2D CNN) porque los datos de entrada son imágenes. Sin embargo, es posible usar series temporales (información de una variable a lo largo del tiempo) como datos de entrada. En este caso se emplea una red neuronal convolucional de una dimensión (1D CNN), la cual extraerá características que se usarán en una etapa de clasificación posterior, como se describió antes, pero esta vez las operaciones se harán

solo en una dimensión (el *kernel* solo se mueve en un sentido, por ejemplo). Ejemplos de series temporales hay muchos, como las curvas de luz de una estrella que representan la información de la variación del brillo de ésta en función del tiempo.

2.5. Métricas de Desempeño

Las métricas de desempeño se emplean para cuantificar el rendimiento de un algoritmo en la tarea para la cual ha sido creado. A continuación se describen brevemente aquellas que, de acuerdo con la literatura consultada, son comúnmente utilizadas para evaluar la capacidad de clasificación de los modelos generados.

2.5.1. Matriz de confusión

La matriz de confusión es una herramienta clave para evaluar el desempeño de un modelo de clasificación, mostrando cómo se distribuyen las predicciones en relación con las etiquetas reales [21]. En este trabajo, donde se clasifican las 26 letras del alfabeto dactilológico de la lengua de señas americana (*American Sign Language*) (ASL), la matriz de confusión permite visualizar los aciertos y errores del modelo. Las filas representan las clases reales (las letras del ASL), mientras que las columnas indican las clases predichas. La diagonal principal refleja las predicciones correctas, mientras que las celdas fuera de la diagonal muestran confusiones entre letras, lo que ayuda a identificar patrones de error. Esta herramienta es especialmente útil para detectar confusiones frecuentes entre letras visualmente similares, lo que puede guiar mejoras en el modelo.

2.5.2. Exactitud

La exactitud es la fracción de clasificaciones correctas hechas por el algoritmo. Matemáticamente se expresa como la relación entre la suma de verdaderos positivos y verdaderos negativos (VN), y el total de predicciones del modelo (correctas e incorrectas), es decir, la suma de los verdaderos positivos, verdaderos negativos, falsos

positivos y falsos negativos. Por lo tanto,

$$Exactitud = \frac{VP + VN}{VP + VN + FP + FN}. \tag{2.1}$$

Esta métrica no es un buen punto de referencia para evaluar el desempeño del modelo cuando se trabaja con un conjunto de datos desbalanceados.

2.5.3. Top-k

El top-K es una métrica utilizada para evaluar el rendimiento de modelos de clasificación, especialmente en tareas donde se predice una lista de posibles resultados ordenados por probabilidad [21]. En términos simples, la métrica top-K mide la exactitud del modelo al verificar si la verdadera etiqueta se encuentra dentro de las K predicciones más probables.

$$\text{Top-}k = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}\{y_i \in \hat{Y}_i^{(k)}\}$$

donde:

- N: número total de muestras.
- y_i: etiqueta verdadera de la muestra i.
- $\hat{Y}_i^{(k)}$: conjunto de las k clases más probables predichas por el modelo.
- $\mathbf{1}\{\cdot\}$: función indicadora que vale 1 si la condición se cumple y 0 en caso contrario.

2.5.4. F1-score

El F1-score es una métrica utilizada para evaluar el rendimiento de modelos de clasificación en machine learning. Combina dos medidas importantes: precisión (cuántos de los casos predichos como positivos son realmente positivos) y recall (cuántos de los casos positivos reales fueron correctamente identificados)[3].

[3] `https://ellielfrank.medium.com/understanding-the-f1-score-55371416fbe1`

El F1-score es el promedio armónico de ambas, lo que le da igual importancia a la precisión y al recall. Su valor oscila entre 0 y 1, donde 1 indica un modelo perfecto (alta precisión y recall) y 0 un rendimiento pobre. Es útil para equilibrar ambos aspectos, especialmente en casos donde hay desbalance de clases, ayudando a tomar decisiones informadas sobre el modelo.

$$F1 = 2 \cdot \frac{Precisión \cdot Recall}{Precisión + Recall}$$

donde:

$$P = \frac{TP}{TP + FP}, \quad R = \frac{TP}{TP + FN}$$

- TP: verdaderos positivos.
- FP: falsos positivos.
- FN: falsos negativos.

2.6. Síntesis

El presente capítulo ha proporcionado una revisión de los antecedentes para comprender la aplicación de técnicas de inteligencia artificial (AI) y aprendizaje profundo (DL) en el reconocimiento y clasificación del alfabeto dactilológico de la lengua de señas americana (*American Sign Language*) (ASL). Se han revisado trabajos previos que han logrado altas exactitudes en la clasificación de dígitos y letras en ASL utilizando modelos de redes neuronales convolucionales (CNN) ad-hoc y técnicas de aprendizaje por transferencia. Además, se ha explorado el mecanismo de la audición, el contexto histórico y social de la pérdida auditiva, y la importancia de la inclusión social y accesibilidad para las personas sordas.

Este capítulo ha establecido una base para el desarrollo y evaluación de modelos de DL en el reconocimiento de señas, destacando la relevancia de estas técnicas en la mejora de la comunicación y la inclusión social.

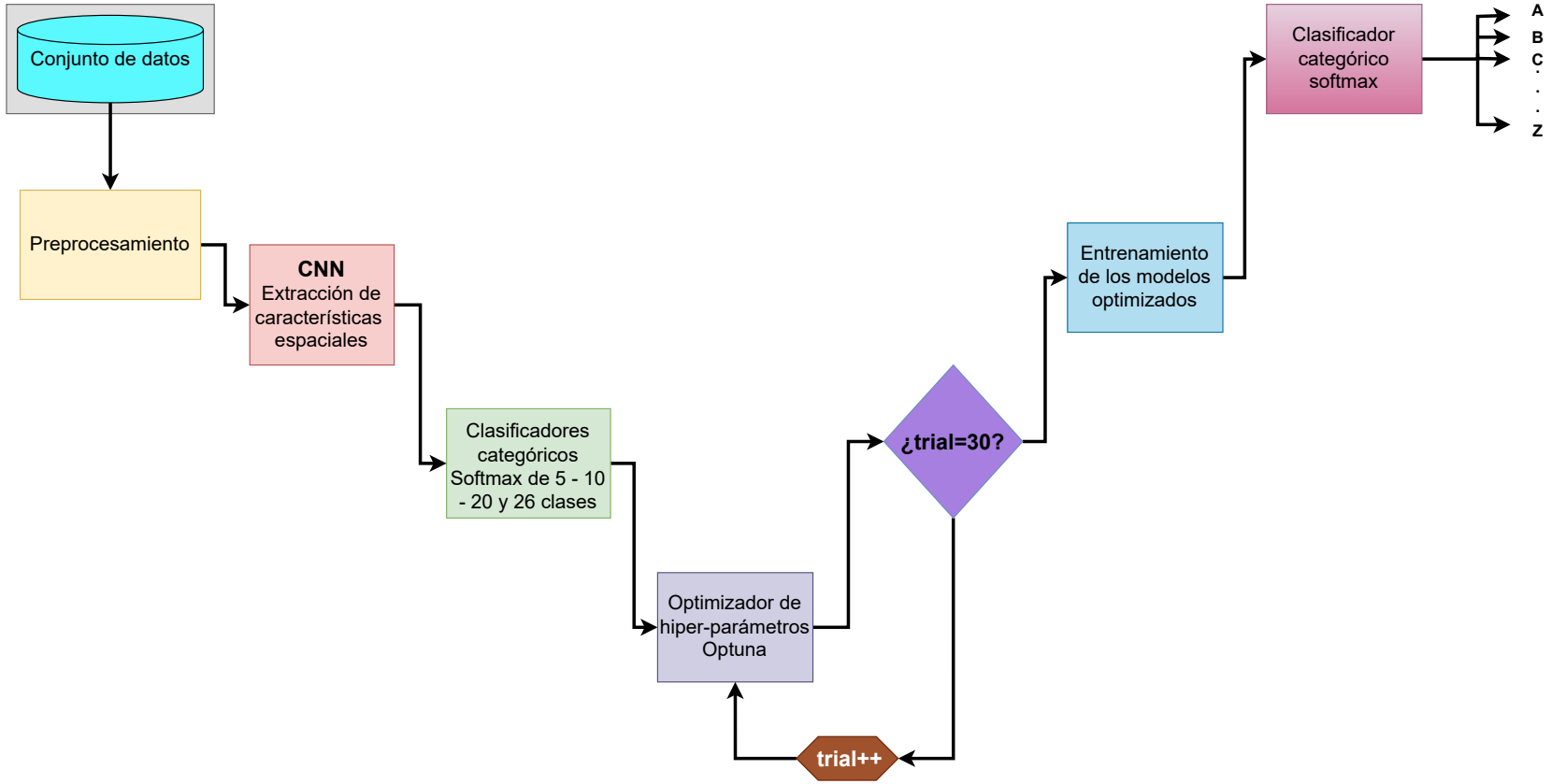
Capítulo 3

Metodología

En el desarrollo de este trabajo de grado, se llevó a cabo una serie de pasos meticulosos que abarcaron desde la búsqueda y selección de la base de datos en *Kaggle* hasta la implementación y optimización de modelos de aprendizaje profundo basados en redes neuronales convolucionales *CNNs*. En primer lugar, se realizó una búsqueda en Kaggle para identificar una base de datos adecuada que cumpliera con los requisitos del proyecto. Posteriormente, se llevó a cabo un preprocesamiento detallado de las imágenes, donde se convirtieron a escala de grises (blanco y negro) para reducir la complejidad y el tamaño de los datos, se normalizaron los valores de los píxeles para asegurar que estuvieran en un rango consistente, y se generaron vectores one-hot para las etiquetas de clasificación, lo que facilitó su uso en la red neuronal.

A través del uso de la librería Keras de Python, se logró la construcción y entrenamiento del modelo de la red neuronal, donde además se configuró de manera adaptable para clasificar 5, 10, 20 y 26 clases del *ASL*, lo que permitió evaluar la capacidad de los modelos para manejar diferentes niveles de complejidad en la clasificación. Para optimizar estos parámetros, se utilizó Optuna, una herramienta de optimización de hiper-parámetros que permitió ajustar de manera eficiente los valores de manera automática. Optuna se configuró para realizar 32 pruebas, explorando diferentes combinaciones de hiper-parámetros durante un entrenamiento de 10 épocas (epochs) con un tamaño de lote (batch size) de 32. Los hiper-parámetros que se optimizaron incluyen el número de capas convolucionales, el número de filtros, la tasa de Dropout, número de capas densas, el optimizador y la tasa de aprendizaje. Finalmente, se obtuvieron modelos optimizados, los cuales se compararon con los modelos sin optimizar en términos de cantidad de parámetros, peso en megabytes, tiempo de entrenamiento, exactitud (accuracy), exactitud de validación (val-accuracy), operaciones de punto flotante por segundo (FLOPS), tiempo de evaluación y tiempo de inferencia. La Figura 3.1 presenta la metodología del trabajo.

Figura 3.1: Diagrama de bloques de la metodología propuesta. Elaboración propia.



3.1. Descripción de la Base de Datos

Para este trabajo de investigación se utilizó el conjunto de datos 'ASL Alphabet Image Dataset' disponible en la plataforma *Kaggle*[1]. Esta base de datos fue seleccionada siguiendo criterios específicos establecidos para el reconocimiento del alfabeto dactilológico de la Lengua de Señas: cantidad de datos, número de personas participantes y número de clases disponibles. La base de datos contiene un total de 87.000 imágenes, cada una con una resolución de 200x200 píxeles. Estas imágenes están organizadas en 29 clases, correspondientes a las letras del alfabeto inglés (A-Z) y tres clases adicionales: SPACE, DELETE y NOTHING. En este estudio, se utilizaron las 26 clases correspondientes a las letras A-Z, descartando las clases adicionales.

Las imágenes fueron capturadas en un entorno controlado, lo que garantiza una calidad y consistencia visuales adecuadas para el entrenamiento de modelos de aprendizaje profundo. Específicamente, este control se evidencia en la uniformidad de la iluminación, la estandarización del fondo, la consistencia en el ángulo de captura y la resolución fija de 200x200 píxeles en todas las imágenes. Además, cada clase contiene 3.000 imágenes, proporcionando un conjunto de datos balanceado que es esencial para el entrenamiento y evaluación de modelos de clasificación.

La Figura 3.2 corresponde a una pequeña comparación de algunos gestos del alfabeto de la lengua de señas americana (*American Sign Language*) (ASL) y la lengua de señas mexicana (*Mexican Sign Language*) (MSL). Se puede notar que existen diferencias entre algunos de los gestos, y que por lo tanto es necesario realizar un proceso de ajuste de los modelos para que comprendan el abecedario dactilológico de la lengua de señas americana (*American Sign Language*) (ASL)

3.1.1. Procesamiento de la base de datos

La base de datos se importó directamente a un cuaderno *notebook* de Kaggle desde el repositorio de la misma plataforma, ambiente en el cual se desarrolló y ejecutó posteriormente el algoritmo. Para el manejo de las imágenes, se utilizó la librería `cv2`

[1] `https://www.kaggle.com/datasets/grassknoted/asl-alphabet`

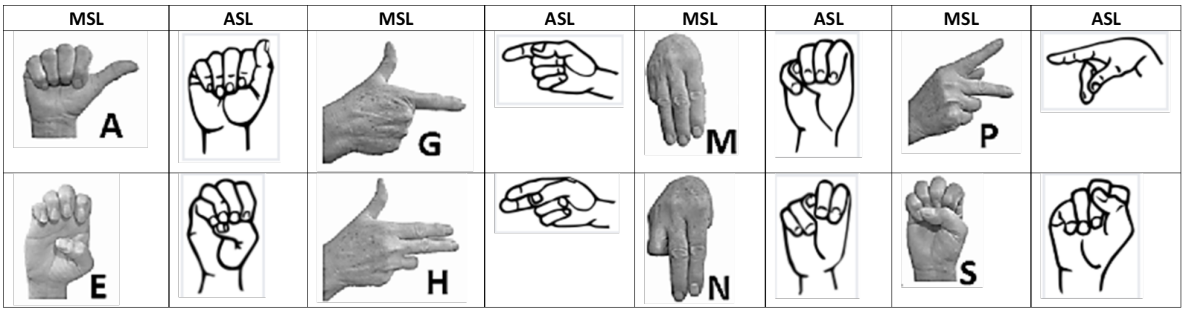


Figura 3.2: Diferencias entre la lengua de señas americana (*American Sign Language*) (ASL) y la lengua de señas mexicana (*Mexican Sign Language*) (MSL) para el alfabeto dactilológico

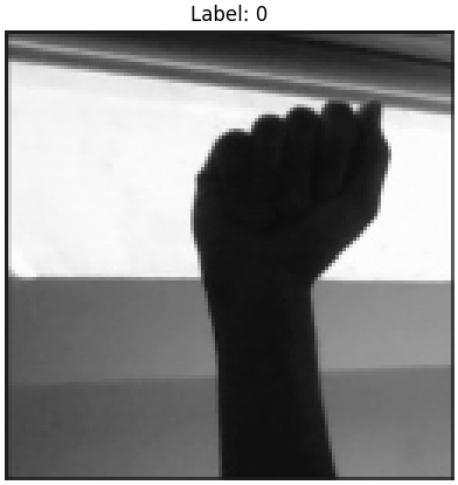
de OpenCV, la cual permitió cargar las imágenes en escala de grises y redimensionarlas a un tamaño de 128x128 píxeles como se muestra en la Figura 3.3. Estas imágenes se almacenaron en listas junto con sus respectivas etiquetas, que corresponden a las categorías de las imágenes. Para facilitar el procesamiento en tareas de clasificación multiclase, las etiquetas se convirtieron a una representación categórica mediante el método de *one-hot encoding*, también conocido como codificación categórica. La codificación one-hot es un formato muy utilizado para datos categóricos, también llamado codificación categórica [19]. Esta técnica es ampliamente utilizada para transformar datos categóricos en un formato que los modelos de aprendizaje profundo pueden procesar eficientemente, utilizando, por ejemplo, la función `to_categorical` de *Keras*. Este enfoque garantiza que cada clase sea representada de manera única y discreta, lo cual es crucial para el entrenamiento de modelos de clasificación multiclase.

A continuación, se utilizó la función `train_test_split`[2] de la librería `sklearn` para dividir el conjunto de datos de manera aleatoria en tres subconjuntos: un 80 % para entrenamiento, un 5 % para validación y un 15 % para prueba. Esta división permite que el modelo aprenda de una amplia variedad de datos, se valide su desempeño durante el entrenamiento y se evalúe de manera imparcial en un conjunto de datos nunca antes visto.

Finalmente, para optimizar el procesamiento, las listas de imágenes y etiquetas

[2] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`

Figura 3.3: Base de datos pre-procesada. Tomada de kaggle.



(entrenamiento, validación y prueba) se convirtieron en arreglos utilizando la librería `numpy`, especificando que los valores fueran de tipo *float64*. Además, las imágenes se normalizaron dividiendo los valores de píxeles entre 255, lo que asegura que los datos estén en un rango de 0 a 1, lo cual es adecuado para su uso en modelos de aprendizaje profundo. Este proceso asegura que los datos estén en un formato adecuado para su uso en modelos de aprendizaje profundo, y que la división de los conjuntos de entrenamiento, validación y prueba se realice de forma representativa. Para ello, las muestras fueron sometidas previamente a la función *shuffle*[3], la cual baraja aleatoriamente los datos antes de la partición, evitando sesgos y garantizando que cada subconjunto contenga ejemplos diversos y proporcionales de todas las clases.

3.2. Arquitectura de Aprendizaje Profundo Implementada (Modelos Base)

La arquitectura de los modelos base implementados en este estudio se construyó tomando como referencia directa la investigación de [10], cuyo diseño se enfocó en el reconocimiento del alfabeto de la lengua de señas mexicana (*Mexican Sign Language*) (MSL). En consecuencia, los parámetros iniciales de configuración no fueron definidos de manera arbitraria, sino replicados de dicha publicación y adaptados para el

[3] `https://www.tensorflow.org/api_docs/python/tf/keras/random/shuffle`

reconocimiento del alfabeto de la lengua de señas americana (*American Sign Language*) (ASL).

La red neuronal consta de tres bloques convolucionales conectados secuencialmente con 64, 32 y 16 filtros, respectivamente. Cada bloque combina operaciones de convolución, normalización y activación, lo que permite extraer características progresivamente más abstractas de las imágenes de entrada. Posteriormente, se incorpora una capa de *Flatten* para transformar la representación espacial en un vector unidimensional, seguido de tres capas densas con 64, 32 y 32 unidades neuronales. Estas capas densas cumplen la función de integrar la información extraída y facilitar la clasificación. Finalmente, se añade una capa de salida con activación *softmax*, cuyo número de neuronas se ajusta a la cantidad de clases en cada experimento (5, 10, 20 y 26), permitiendo obtener probabilidades de pertenencia para cada categoría.

A continuación se hace una breve explicación de las capas implementadas y las técnicas de regularización utilizadas en la arquitectura general de los modelos base descrita anteriormente:

- ***Capa Convolucional*** Las capas convolucionales son el núcleo de las redes neuronales convolucionales (CNNs), aplicando filtros (kernels) a las imágenes de entrada para extraer características locales como bordes, texturas y formas. Estas capas capturan patrones espaciales en los datos de entrada, lo que es crucial para la identificación de características relevantes en imágenes.

- ***Flatten*** La capa de Flatten es una operación que aplana los datos, convirtiendo una matriz o tensor en una secuencia lineal de valores, esto es necesario porque, después de la extracción de características en las capas convolucionales, la información debe ser pasada a capas densas (fully connected) para realizar tareas de clasificación.

- ***Dropout*** La capa de Dropout es una técnica de regularización que se utiliza para prevenir el sobreajuste, "apagando" aleatoriamente algunas neuronas durante el entrenamiento con una probabilidad dada. Esto fuerza a la red a aprender características más robustas y reduce la dependencia de cualquier

neurona individual, mejorando la generalización del modelo.

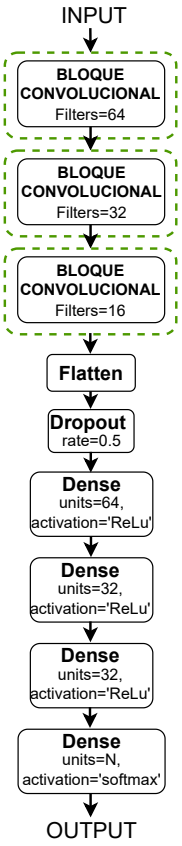
- ***Capas Densas*** Las capas densas, también conocidas como capas completamente conectadas, son capas donde cada neurona está conectada a todas las neuronas de la capa anterior. Estas capas realizan la clasificación final de las características extraídas por las capas convolucionales y de pooling, proporcionando la salida del modelo.

- ***Capa Softmax*** La capa softmax se utiliza comúnmente como capa de salida en problemas de clasificación multiclase. Su función es transformar las activaciones de la última capa densa en una distribución de probabilidades, asignando a cada clase un valor entre 0 y 1 cuya suma total es igual a 1. De esta forma, cada salida puede interpretarse como la probabilidad de pertenencia de la muestra a una categoría específica, siendo la predicción final aquella clase con mayor probabilidad asignada. Esta característica convierte a la función softmax en una herramienta esencial para la toma de decisiones en tareas donde las clases son mutuamente excluyentes.

En total se armaron cuatro grupos de modelos secuenciales, correspondientes a 5, 10, 20 y 26 clases, siguiendo un criterio de selección basado estrictamente en el orden alfabético de los gestos, sin tener en cuenta el tipo, forma o configuración de la mano. Este enfoque permitió evaluar progresivamente la complejidad del problema a medida que aumentaba el número de clases incluidas en el clasificador. Aunque no se diferenciaron los gestos por similitud visual, los resultados obtenidos demuestran que las configuraciones propuestas fueron lo suficientemente robustas para mantener un buen desempeño incluso al abordar las 26 clases del alfabeto completo.

En síntesis, los modelos base se diseñaron siguiendo la propuesta [10] y se adaptaron para clasificar distintos subconjuntos de letras del alfabeto dactilológico en la ASL, hasta abarcar el conjunto completo de 26 clases. La Figura 3.4 ilustra esta arquitectura general, donde se observan los bloques convolucionales, la etapa densa y la configuración de salida.

Figura 3.4: Arquitectura de modelos base ad-hoc implementados. Elaboración propia.



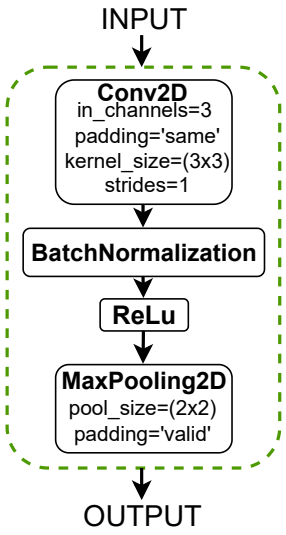
3.2.1. Descripción Bloque Convolucional de los modelos base

El bloque convolucional implementado en los modelos base se construyó siguiendo la propuesta de [10], manteniendo los parámetros definidos en dicha investigación. En particular, la capa convolucional utilizó filtros con tamaño de kernel 3×3, un *stride* de 1 y un *padding* de tipo "same". Estas configuraciones permiten un desplazamiento detallado de los filtros sobre la imagen y aseguran que las dimensiones espaciales se conserven después de la convolución, lo que resulta esencial para no perder información en los bordes. A continuación, se incorporó una operación de *MaxPooling* con ventana 2×2, también tomada de la configuración original, con el fin de reducir la dimensionalidad de los mapas de características y mantener las propiedades más relevantes para la clasificación.

Como se muestra en la Figura 3.5, cada bloque está compuesto por cuatro

componentes principales: una capa convolucional, una capa de normalización por lotes, la función de activación *ReLU* y una capa de *MaxPooling2D*. Este diseño permite extraer características progresivamente más complejas, al tiempo que estabiliza y agiliza el proceso de entrenamiento.

Figura 3.5: Estructura Bloque Convolucional modelos base. Elaboración propia.



A continuación se explica cada una de las capas implementadas en el bloque convolucional explicado anteriormente:

- ***BatchNormalization*** Técnica que normaliza las activaciones de la capa convolucional, ajustando la media y la desviación estándar de las entradas. Esto acelera el entrenamiento, mejora la estabilidad del aprendizaje y mitiga el desvanecimiento del gradiente, permitiendo emplear tasas de aprendizaje más altas.

- ***Función de Activación ReLU*** La activación *ReLU* introduce no linealidad al modelo devolviendo cero para valores negativos y la entrada misma para valores positivos. Su simplicidad la hace eficiente en la práctica y permite que la red aprenda relaciones más complejas sin problemas de gradiente desvanecido.

- ***MaxPooling*** Operación de reducción espacial que aplica una ventana de 2×2, tomando el valor máximo de cada región. Con ello se disminuye el número de parámetros, se reducen los cálculos y se controla el sobreajuste, sin perder las características principales extraídas en las primeras etapas.

En resumen, **el bloque convolucional** utilizado en este trabajo está diseñado para realizar una representación semántica de las imágenes de entrada. Cada bloque convolucional está compuesto por una capa convolucional, una capa de normalización por lotes, una función de activación *ReLU* y una capa de *MaxPooling2D*. Estas capas trabajan juntas para capturar patrones espaciales en los datos de entrada, reducir la dimensionalidad de la representación semántica y mejorar la estabilidad y generalización del modelo.

3.3. Entrenamiento

Para la implementación de los diferentes modelos de clasificación categórica *ad hoc* para 5, 10, 20 y 26 clases se utilizó el entorno interactivo de programación Kaggle. Esta plataforma permite ejecutar código Python directamente desde un navegador web, aprovechando recursos computacionales en la nube, tales como memoria RAM y almacenamiento, además de aceleradores de hardware como GPU T4 x2, GPU P100 y TPU VM v3-8. La optimización de hiper-parámetros se llevó a cabo también en el entorno de implementación de kaggle con la ayuda del optimizador *Optuna*. Estas herramientas son ideales para ejecutar algoritmos de aprendizaje profundo de alta demanda computacional, especialmente cuando no se dispone de equipos locales con capacidades avanzadas.

En Kaggle, se trabaja con cuadernos o notebooks, los cuales son documentos interactivos que combinan código, texto descriptivo, ecuaciones matemáticas y elementos multimedia. Esto facilita el desarrollo de proyectos de manera estructurada, permitiendo a los desarrolladores agregar contexto y explicaciones adicionales junto al código. Este enfoque no solo mejora la comprensión del trabajo para terceros, sino que también fomenta la colaboración y el mantenimiento eficiente de los proyectos.

Para construir, entrenar y evaluar las redes neuronales convolucionales (CNN) empleadas en el diseño, se utilizó la librería Keras, que forma parte del ecosistema TensorFlow. Esta herramienta destaca por su simplicidad y flexibilidad, al proporcionar una interfaz de alto nivel para definir arquitecturas neuronales complejas con pocas líneas de código. Además, Keras integra múltiples opciones para la optimización del entrenamiento, como el uso de optimizadores personalizados, técnicas de regularización y herramientas de evaluación, lo que la convierte en una elección estándar dentro de la comunidad de aprendizaje profundo.

En el entrenamiento de cada uno de los modelos base se hizo uso de una función especial proporcionada por *Keras*: *ModelCheckpoint*[4] que automáticamente guarda un archivo con extensión *.keras* en donde se almacena el modelo (arquitectura y pesos sinápticos) con el mejor rendimiento, de acuerdo con la métrica que se le indique (en este caso la función de pérdida obtenida con los datos de validación), que se tenga a lo largo de las iteraciones que se llevan a cabo en la fase de aprendizaje de la red.

3.3.1. Bucle de entrenamiento de los modelos base

Durante el entrenamiento de los modelos base se utilizó el optimizador *Adam*[5] con su configuración por defecto; por otra parte, la función de costo empleada para compilar los modelos base y optimizados fue la conocida como *Categorical cross-entropy*[6], la cual es la que se usa por lo general en problemas de clasificación de múltiples clases [19]. Esta métrica permite cuantificar la diferencia entre la distribución real de las clases y la distribución de probabilidades predicha por el modelo.

Su formulación matemática se expresa como:

$$L = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\,\log(\hat{y}_{i,c})$$

donde:

[4] `https://keras.io/api/callbacks/model_checkpoint/`
[5] `https://keras.io/api/optimizers/adam/`
[6] `https://keras.io/api/losses/probabilistic_losses/#categoricalcrossentropy-class`

- N: número total de muestras en el lote de entrenamiento.
- C: número de clases posibles (26 en el caso del alfabeto dactilológico de la ASL).
- $y_{i,c}$: valor binario que indica si la muestra i pertenece a la clase c (codificación *one-hot*).
- $\hat{y}_{i,c}$: probabilidad predicha por el modelo de que la muestra i corresponda a la clase c, obtenida mediante la función *softmax*.

Esta función de pérdida penaliza con mayor severidad aquellas predicciones que asignan baja probabilidad a la clase correcta, incentivando al modelo a generar distribuciones de salida que se aproximen lo más posible a las etiquetas reales. Su uso asegura un entrenamiento más estable y efectivo en tareas de clasificación de múltiples categorías, como la planteada en este trabajo.

Adicionalmente se implementó un *callback* de *checkpoint* que monitoreó esta métrica durante el entrenamiento. El proceso se configuró con un tamaño de lote (*batch size*) de 32 y un total de 15 épocas (*epochs*). Durante el entrenamiento se registraron las métricas de pérdida (*loss*) y exactitud (*accuracy*) tanto en los datos de entrenamiento como en el conjunto de validación, lo que permitió evaluar el comportamiento del modelo en cada época.

Cabe resaltar que, en la publicación [10], la arquitectura base fue entrenada durante 20 épocas. Sin embargo, en el presente trabajo se observó que a partir de la época número 10 el modelo ya alcanzaba niveles de exactitud superiores al 90 %. Por esta razón, se decidió ajustar el entrenamiento de los modelos base a 15 épocas, buscando un balance adecuado entre desempeño y eficiencia computacional.

3.3.2. Teoría y Funcionamiento de Optuna

Para la optimización de *hiper-parámetros* se utilizó *Optuna*[7], un marco de trabajo basado en optimización bayesiana que permite la búsqueda eficiente en espacios o rangos de *hiper-parámetros* complejos. La idea principal consiste en definir una

[7] `https://optuna.org/`

función objetivo $f(\theta)$, donde θ representa el vector de *hiper-parámetros* a evaluar, y encontrar la combinación que minimice la función de pérdida o maximice la métrica de interés:

$$\theta^* = \underset{\theta \in \Theta}{\operatorname{argmin}}\, f(\theta).$$

En este estudio, $f(\theta)$ corresponde al valor de la pérdida de validación obtenida al entrenar el modelo con los *hiper-parámetros* θ. Optuna emplea el algoritmo *Tree-structured Parzen Estimator (TPE)*, que modela la distribución condicional $p(y \mid \theta)$ para distinguir entre *hiper-parámetros* que producen buenos resultados ($l(\theta)$) y aquellos que producen resultados inferiores ($g(\theta)$). El optimizador concentra el muestreo en $l(\theta)$, acelerando la búsqueda de configuraciones más prometedoras. Además, incorpora técnicas de *pruning* o parada temprana, lo que permite descartar configuraciones no competitivas y reducir el costo computacional.

3.3.3. Optimización de Hiper-parámetros

La optimización de hiper-parámetros es un paso crucial en el desarrollo e implementación de los modelos de aprendizaje profundo, ya que permite encontrar configuraciones que mejoran el rendimiento de los modelos. En este estudio se empleó el optimizador *Optuna*, una herramienta que ofrece un rango amplio de opciones para la búsqueda de hiper-parámetros y que puede integrarse con varios frameworks como *TensorFlow*, *Keras*, *PyTorch* o *Scikit-Learn*. La optimización se llevó a cabo en el entorno de desarrollo de *Kaggle*, configurando *Optuna* para realizar 8 pruebas (*trials*) por cada uno de los 4 modelos implementados, con un entrenamiento de 10 épocas por prueba y un tamaño de lote de 32 recomendados por optuna. Durante este proceso se exploraron distintas combinaciones de hiper-parámetros relevantes para maximizar el rendimiento de los modelos en el reconocimiento del abecedario en la lengua de señas americana (*American Sign Language*) (ASL). Los hiper-parámetros considerados fueron:

- ***Número de capas convolucionales (n_conv_layers):*** Se sugirió un rango entre 2 y 4 capas.
- ***Número de filtros convolucionales (c_kernels):*** Se consideraron valores de 16, 32, 64, 128.

- ***Tasa de Dropout (dropout_rate):*** Se exploró un rango entre 0.1 y 0.5.
- ***Numero de capas densas (n_dense_layers):*** Se consideraron entre 1 y 3 capas densas.
- ***Número de Unidades neuronales densas (d_kernels):*** Se consideraron valores de 16, 32, 64, 128.
- ***Optimizador (optimizer_name):*** Se evaluaron tres optimizadores: Adam, RMSprop y SGD.
- ***Tasa de aprendizaje (learning-rate):*** Se exploró un rango logarítmico entre 1e-5 y 1e-1.

Para establecer los rangos de búsqueda en cada hiper-parámetro, se tomó como referencia la arquitectura propuesta por [10]. En dicha investigación, el modelo base utilizó tres capas convolucionales con configuraciones de 64, 32 y 16 filtros. A partir de esta referencia, se definieron los intervalos de exploración en *Optuna*, ampliando los valores tanto por encima como por debajo de los propuestos en Lara-Cázares. Por ejemplo, para el número de filtros se consideraron valores desde 16 hasta 128, lo que permitió abarcar un rango más amplio sin perder la coherencia con la arquitectura inicial. Este mismo criterio se aplicó al resto de hiper-parámetros, de manera que la optimización no se realizó de manera arbitraria, sino siguiendo una estrategia fundamentada en la arquitectura base ya validada en la literatura. De esta forma, se garantizó un proceso de búsqueda controlado y pertinente, orientado a mejorar el desempeño de los modelos implementados.

3.4. Discusión

En este capítulo, se ha delineado la metodología empleada para el desarrollo y evaluación de modelos de aprendizaje profundo destinados al reconocimiento del alfabeto de la lengua de señas americana (*American Sign Language*) (ASL). Esta investigación busca cerrar la brecha de comunicación entre personas oyentes y aquellas que tienen dificultades auditivas, proporcionando una herramienta efectiva para la interpretación de señas. Se ha optado por el uso de redes neuronales convolucionales

(CNNs) debido a su efectividad en la extracción de características visuales de las imágenes de las señas, así como por su auge y gran utilización en los últimos tiempos en diversas aplicaciones de reconocimiento visual.

La metodología incluye la preparación de datos mediante técnicas de "one-hot encoding" y la división del conjunto de datos en subconjuntos de entrenamiento, validación y prueba. La arquitectura de los modelos se implementó con *Bloques Convolucionales*, capas de normalización por lotes *BatchNormalization*, funciones de activación *ReLU* y *softmax*, y capas de *MaxPooling2D* para capturar características progresivamente más complejas. Además, se incorporó una capa de *Dropout* y *Flatten* para mejorar la generalización y reducir el sobreajuste. El entrenamiento se llevó a cabo utilizando el optimizador *Adam* y se implementó un proceso de optimización de hiper-parámetros con Optuna para encontrar la configuración óptima de los diferentes modelos implementados.

Capítulo 4

Análisis de Resultados

En esta sección, se aborda un análisis de los resultados obtenidos durante la implementación de los modelos base de clasificación del abecedario dactilológico de la lengua de señas americana (*American Sign Language*) (ASL). El objetivo principal fue evaluar la capacidad de las arquitecturas implementadas para identificar correctamente las 26 clases correspondientes a cada letra del abecedario dactilológico de la lengua de señas americana (*American Sign Language*) (ASL). Para ello, se realizaron diversas investigaciones que incluyen la configuración inicial de los modelos como versiones optimizadas mediante ajustes avanzados de hiper-parámetros con el uso de Optuna, con el fin de mejorar su rendimiento.

La evaluación de los resultados se llevó a cabo empleando métricas ampliamente aceptadas en el campo de la clasificación de imágenes, como la matriz de confusión, exactitud, el F1-score y el top-k accuracy. Estas métricas permitieron medir tanto el desempeño global de los modelos como su capacidad para discriminar entre clases con similitudes visuales. Adicionalmente, se analizaron patrones de error para identificar las clases que presentaron mayores dificultades en su clasificación. Este enfoque crítico permite no solo resaltar los logros del sistema, sino también proponer mejoras específicas que podrían implementarse en estudios futuros.

4.1. Desempeño computacional de los modelos base

La Tabla 4.1 presenta los resultados obtenidos durante la implementación de modelos base implementados para clasificar gestos estáticos del alfabeto dactilológico en la lengua de señas americana (*American Sign Language*) (ASL). Estos modelos, sin optimización, fueron configurados con una arquitectura base compuesta por tres capas convolucionales seguidas de una operación de *Flatten*. Este enfoque permitió reducir la

dimensionalidad de manera eficiente al extraer las características más representativas de las imágenes procesadas. Posteriormente, se incluyeron dos capas densas con 64 y 32 unidades neuronales, respectivamente, utilizando la función de activación ReLU. Para minimizar el sobreajuste, se aplicó una capa de *Dropout* con un valor fijo de 0.05 antes de la etapa final de clasificación. La salida del modelo fue una capa densa cuyo tamaño varió según el número de clases en cada configuración (5, 10, 20 o 26), con una función de activación softmax para asignar probabilidades a cada clase.

Tabla 4.1: Métricas de desempeño computacional de modelos base

Modelo	**Tiempo de entrenamiento (segundos)**	**Costo computacional del modelo (FLOPS)**	**Cantidad de parámetros**	**Tiempo de evaluación (segundos)**	**Tiempo de inferencia (segundos)**
5 clases	300,41	$1,558731454 \times 10^{-10}$	923.921	2,78	0,0008
10 clases	604,12	$1,558731804 \times 10^{-10}$	924.416	4,25	0,0006
20 clases	1106,21	$1,558732504 \times 10^{-10}$	925.406	5,55	0,0005
26 clases	1387,28	$1,558732924 \times 10^{-10}$	926.000	6,64	0,0004

Los datos de la Tabla 4.1 destacan aspectos clave del desempeño computacional de los modelos base. El **tiempo de entrenamiento** aumenta de manera considerable conforme se incrementa el número de clases, pasando de 300,41 segundos en el modelo de 5 clases a 1387,28 segundos en el de 26 clases. De igual forma, el **tiempo de evaluación** se eleva progresivamente de 2,78 a 6,64 segundos, lo que refleja la mayor exigencia al procesar conjuntos de prueba más amplios. En contraste, métricas como el costo computacional (FLOPS) y la cantidad de parámetros se mantienen prácticamente constantes, ya que la arquitectura no cambia significativamente entre modelos, salvo en el número de neuronas de salida.

Un aspecto llamativo es el comportamiento del **tiempo de inferencia**, que disminuye de 0,0008 segundos en el modelo de 5 clases a 0,0004 segundos en el de 26 clases. Esto indica que, aunque el entrenamiento y la evaluación global requieren más tiempo al aumentar la complejidad de la tarea, la predicción individual sigue siendo altamente eficiente. Este resultado muestra que la arquitectura propuesta no solo escala de manera controlada con el número de clases, sino que además mantiene tiempos de inferencia muy bajos, lo cual la hace adecuada para aplicaciones de reconocimiento en tiempo real.

La Tabla 4.2 presenta el uso de recursos computacionales de los modelos base

durante su entrenamiento. En todos los casos, se utilizó una tarjeta gráfica **GPU T4 en configuración doble (x2)**, con un número constante de **15 épocas**, un **tamaño de lote (batch size) de 32** y el optimizador **Adam** tomados de [10] lengua de señas mexicana (*Mexican Sign Language*) (MSL). También se observa que el **tamaño final del modelo** se mantiene invariable en **1,18 MB** para las configuraciones de 5, 10, 20 y 26 clases, lo que indica que la arquitectura de red propuesta conserva el mismo número de parámetros independientemente del número de categorías a clasificar. Sin embargo, el **consumo de memoria RAM** muestra un aumento progresivo a medida que se incrementa el número de clases: de **10,2 GB** en el modelo de 5 clases, pasa a **14,6 GB** en el de 10 clases, **19,5 GB** en el de 20 clases y alcanza **22,9 GB** en el de 26 clases. Este comportamiento evidencia que, aunque el modelo mantiene una estructura compacta en términos de almacenamiento, el crecimiento en la cantidad de clases incrementa significativamente la demanda de recursos en memoria durante el entrenamiento. En consecuencia, la tabla permite inferir que el costo computacional no se ve reflejado en el tamaño del modelo final, sino en la **memoria necesaria para procesar los datos**, aspecto crucial al escalar la arquitectura hacia escenarios de mayor complejidad.

Tabla 4.2: Uso de recursos computacionales de los modelos base

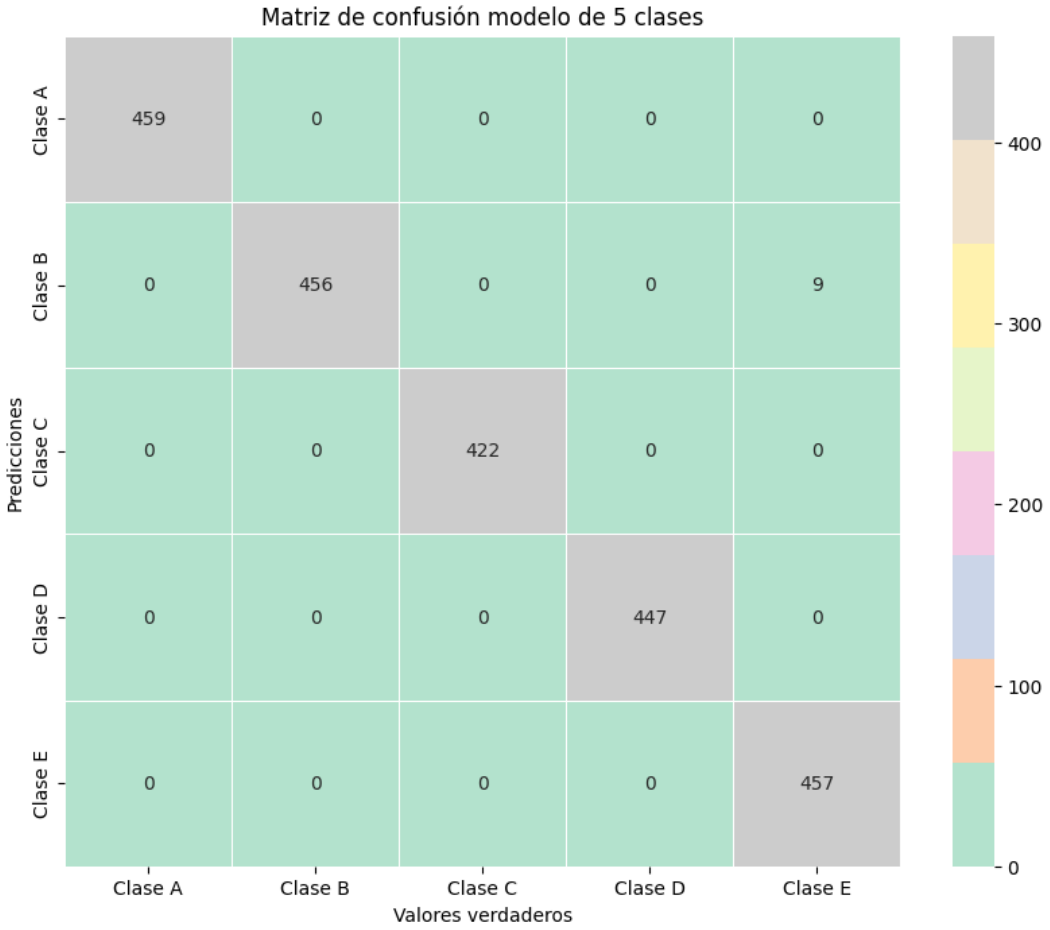
Modelo	**Tarjeta**	**Consumo de ram (GB)**	**# Epoch**	**Batch size**	**Tamaño del modelo (MB)**	**Optimizador**
5 clases	GPU T4 x2	10,2	15	32	1,18	adam
10 clases	GPU T4 x2	14,6	15	32	1,18	adam
20 clases	GPU T4 x2	19,5	15	32	1,18	adam
26 clases	GPU T4 x2	22,9	15	32	1,18	adam

4.2. Resultados y Análisis de los Modelos Base

Las métricas de desempeño utilizadas en este estudio (exactitud, *Top-k Accuracy* y *F1-score*) fueron calculadas sobre el *set de prueba*, el cual corresponde al 15 % del total de los datos (aproximadamente 450 muestras por clase). Esto asegura que la evaluación se haya realizado sobre datos completamente independientes del entrenamiento y de la validación, lo que permite una valoración objetiva de la capacidad de generalización de los modelos. El conjunto de validación (5 % de los datos) fue empleado únicamente para

activar mecanismos de *early-stopping*[1], sin ser considerado para la evaluación final.

Figura 4.1: Tabla de confusión modelo base de 5 clases con el conjunto de prueba. Elaboración propia.



El análisis de las matrices de confusión (Figura 4.1, Figura 4.2, Figura 4.3, Figura 4.4) complementa las métricas cuantitativas al mostrar gráficamente los aciertos y errores en la clasificación de cada clase. En el caso del modelo de 5 clases se observa un rendimiento casi perfecto, con un único error relevante en la confusión entre las letras E y B. A medida que el número de clases aumenta, los errores tienden a distribuirse en mayor medida: por ejemplo, en el modelo de 10 clases la letra A se confundió en 12 ocasiones con B y la letra I en 52 casos con E, mientras que en el modelo de 20 clases la confusión más notoria fue entre P y Q con 63 muestras.

[1] `https://keras.io/api/callbacks/early_stopping/`

Figura 4.2: Tabla de confusión modelo base de 10 clases con el conjunto de prueba. Elaboración propia.

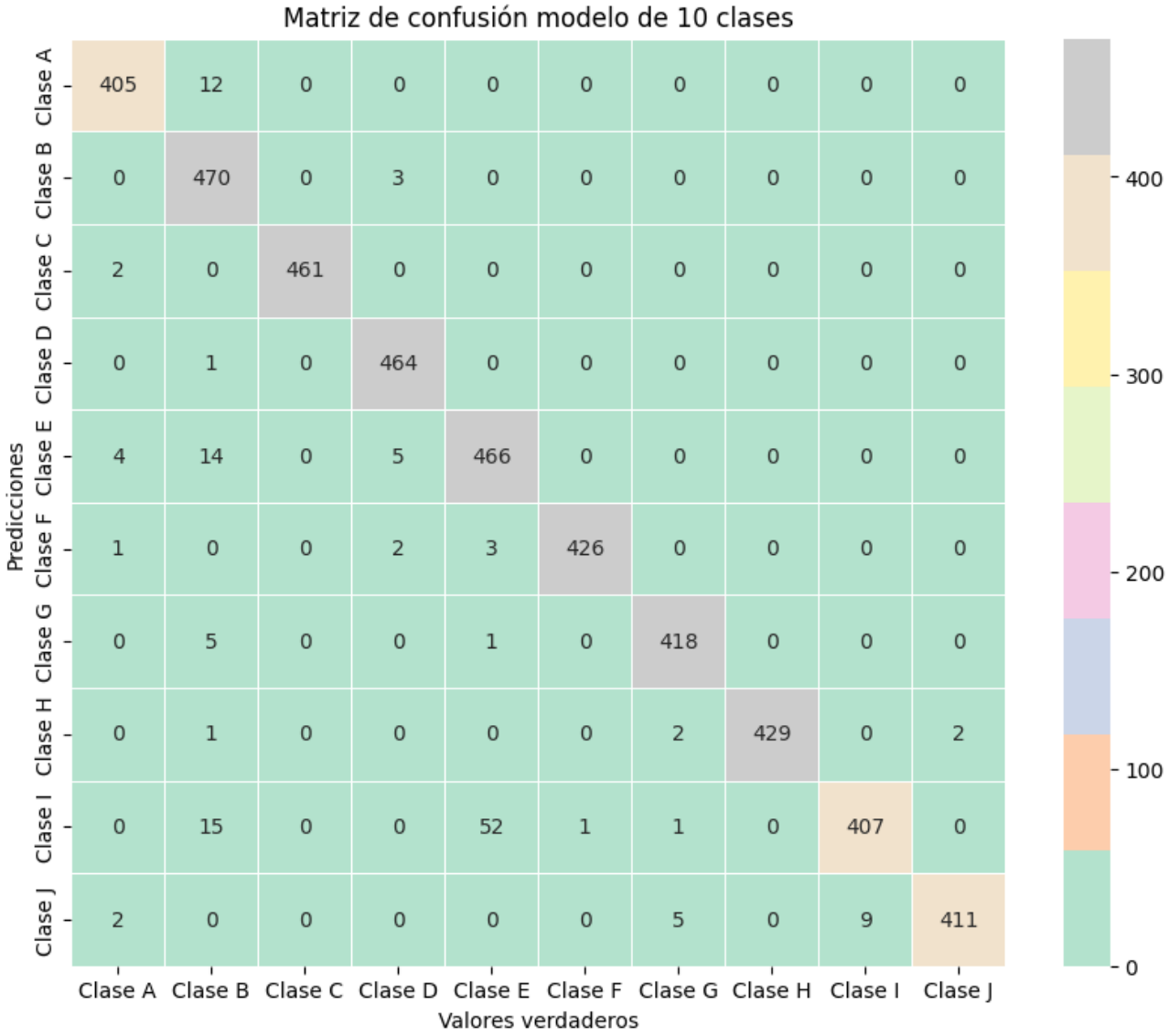


Figura 4.3: Tabla de confusión modelo base de 20 clases con el conjunto de prueba. Elaboración propia.

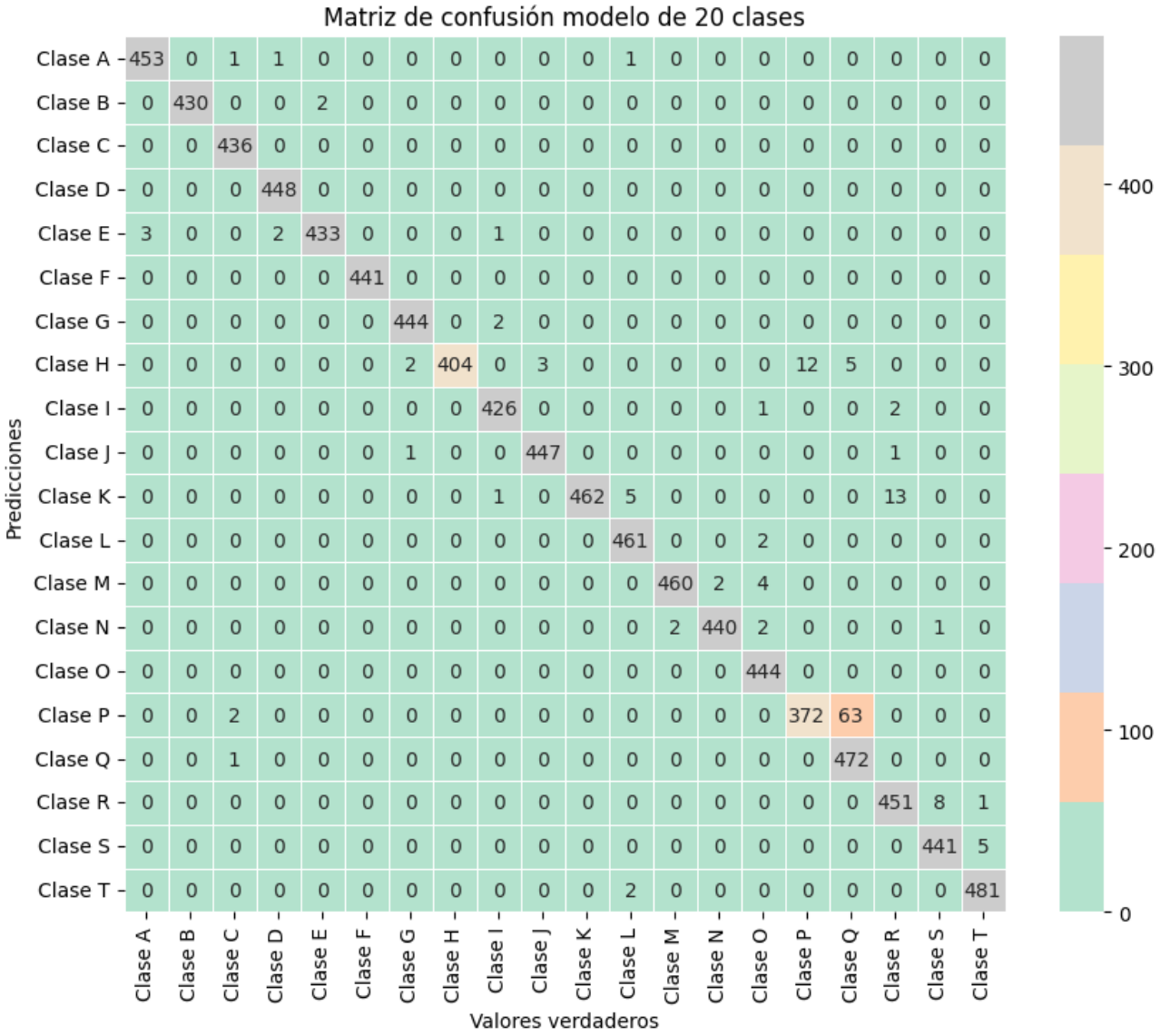
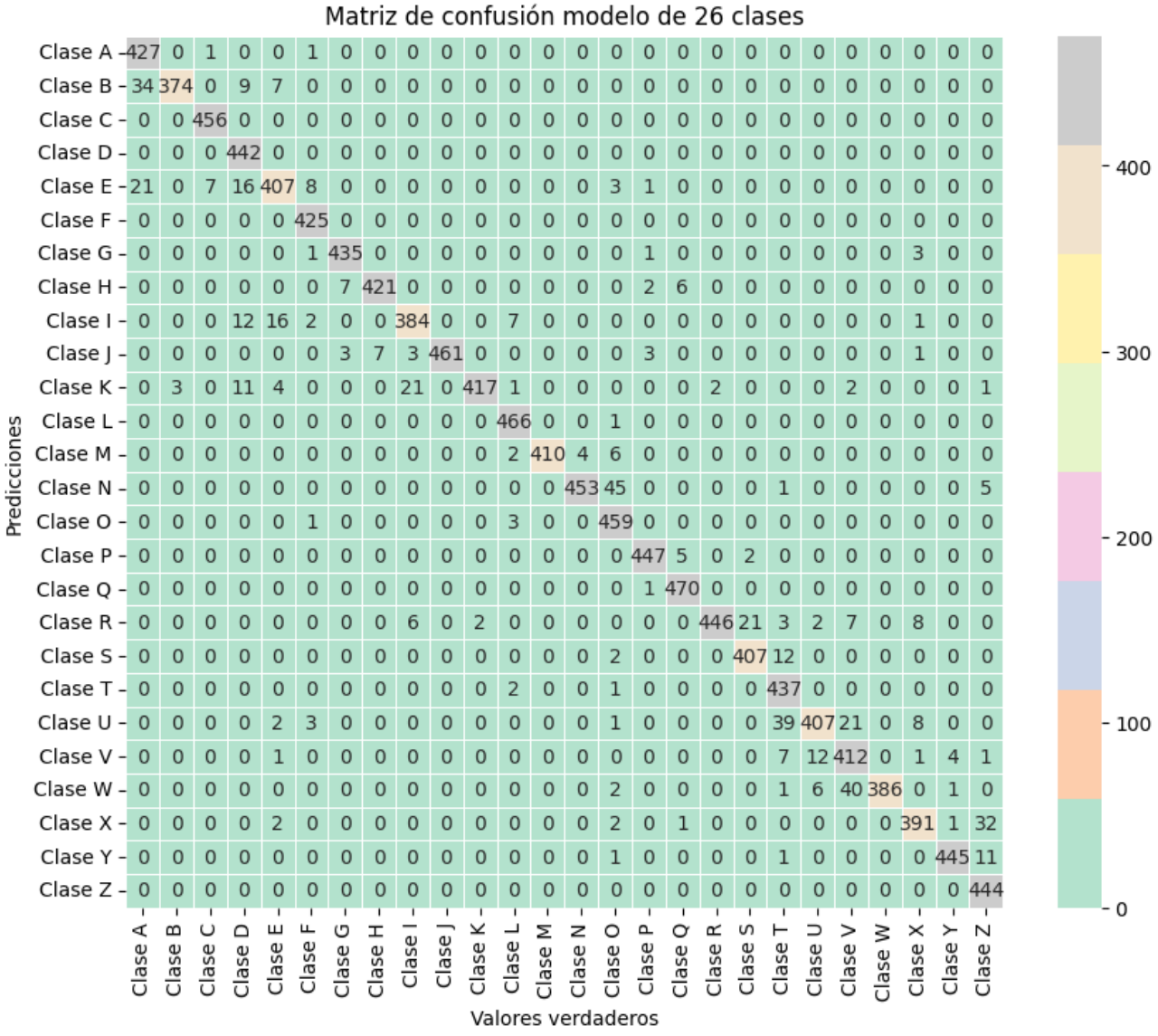


Figura 4.4: Tabla de confusión modelo base de 26 clases con el cojunto de prueba. Elaboración propia.

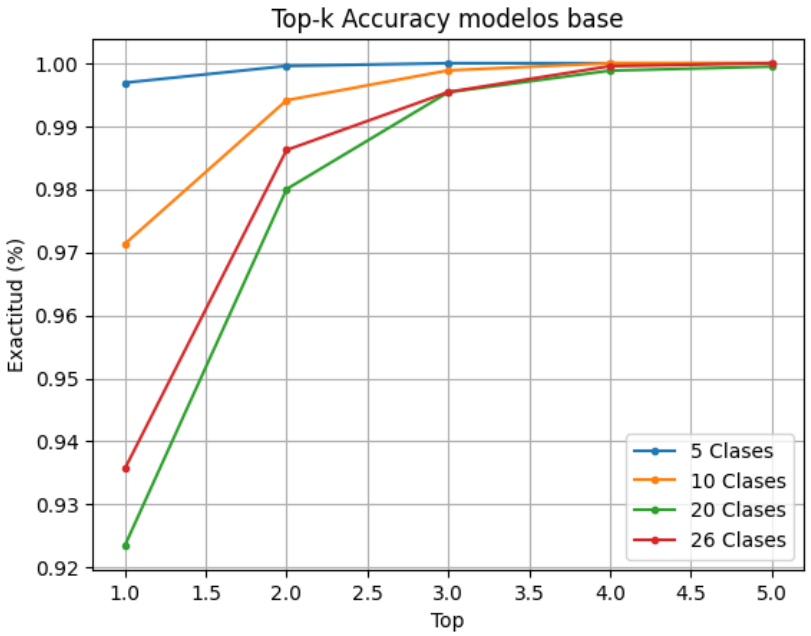


Finalmente, en el modelo de 26 clases —correspondiente al alfabeto completo— se mantiene un rendimiento sólido en general, aunque los errores se hacen más frecuentes y dispersos: la clase B se confundió en 34 ocasiones con A, la clase I en 16 ocasiones con C y 20 con E, y la clase X en 32 casos con Z. Estos patrones evidencian que el incremento en el número de clases aumenta la complejidad del problema, reduciendo la robustez del modelo. En consecuencia, resulta necesario aplicar estrategias de optimización y regularización que mitiguen estas confusiones y fortalezcan el desempeño en escenarios de clasificación multiclase.

Tabla 4.3: Desempeño de los modelos base con el conjunto de prueba

Métrica	**Modelo 5 clases**	**Modelo 10 clases**	**Modelo 20 clases**	**Modelo 26 clases**
Exactitud [%]	99,60	96,82	98,28	95,11
Top-k [%]	99,92	99,64	99,90	99,74

Figura 4.5: Exactitud Top-k de los modelos base para diferentes modelos con el conjunto de prueba



En Tabla 4.3 presenta el desempeño de los modelos base evaluados con el conjunto de prueba en términos de *exactitud* y *Top-k accuracy*. En la métrica de exactitud, el modelo de 5 clases obtuvo el mejor resultado con un 99,60 %, seguido por el modelo

de 20 clases con 98,28 %. A medida que aumenta el número de clases, se observa una ligera disminución en el rendimiento, siendo el modelo de 26 clases el que presenta la menor exactitud con un 95,11 %. Esto refleja la mayor complejidad del problema de clasificación al incrementar el número de categorías a reconocer.

La métrica *top-k accuracy* presentada en la Figura 4.5 muestra valores consistentemente altos en todos los modelos, superiores al 99 %. Esto indica que, aunque la predicción principal no siempre coincide con la clase correcta (como refleja la exactitud), la clase verdadera suele encontrarse dentro de las k predicciones más probables. En particular, los modelos de 5 y 20 clases alcanzan los valores más altos (99 %,92 % y 99 %,90 %), lo que evidencia una alta capacidad de los modelos para priorizar las clases correctas entre sus opciones de salida, incluso en escenarios con mayor complejidad de clasificación.

4.3. Optimización de Modelos Base

La optimización de hiper-parámetros fue llevada a cabo en el entorno de desarrollo de *Kaggle*, incorporando la librería de *Optuna*. Primero, se hizo una selección de los hiper-parámetros en los cuales se iba a realizar la búsqueda. La Tabla 4.4 presenta los resultados de las 8 pruebas que se realizaron para cada uno de los modelos implementados con diferentes combinaciones de los hiper-parámetros seleccionados en la Subsección 3.3.3 respectivamente.

La optimización se llevó a cabo mediante 8 pruebas para cada uno de los cuatro modelos implementados utilizando los optimizadores *Adam*, *RMSprop* y *SGD*, cada uno seleccionado por su capacidad para manejar diferentes características de los datos y adaptarse a distintas dinámicas de aprendizaje [19]. Adam combina las ventajas de AdaGrad y RMSprop, ajustando automáticamente las tasas de aprendizaje en cada parámetro, lo que lo hace especialmente eficiente en problemas con gradientes ruidosos. RMSprop, por su parte, controla el tamaño de los pasos en función de las medias móviles de los gradientes recientes, lo que resulta útil para estabilizar el entrenamiento en redes profundas. Finalmente, SGD, aunque más simple, proporciona robustez y un control granular en el ajuste de parámetros, especialmente cuando se

combina con un buen ajuste de la tasa de aprendizaje. Esta estrategia permitió explorar un rango diverso de combinaciones para identificar la configuración óptima que maximice la exactitud y minimice la pérdida, mejorando tanto el rendimiento como la capacidad de generalización de los modelos.

Para la optimización de los modelos base de 5, 10, 20 y 26 clases, se realizaron 32 pruebas totales en Optuna, con el objetivo de encontrar la configuración que ofreciera el mejor rendimiento en términos de exactitud en la validación (val_acc). Entre estas pruebas *Trials*, se eligió el sexto *Trial* para el modelo de 5 clases con un val_acc=99,33 %, para el modelo de 10 clases se escogió el tercer *Trial* con un val_acc=98,93 %, para el modelo de 20 clases se escogió el segundo *Trial* con un val_acc=99,06 %. Finalmente, para el modelo de 26 clases se escogió el séptimo *Trial*, que destacó por ofrecer un val_acc=99,41 %.

Tabla 4.4: Pruebas del Optimizador de Hiper-parámetros (Optuna)

Modelo	**Trial**	**val_acc**	**n_conv_layers**	**c_kernels**	**dropout_rate**	**n_dense_layers**	**d_kernels**	**Optimazer**	**learning_rate**
5 clases	0	0,9413	4	128-128-64-16	0,4632	2	128-32	Adam	$5,0502 \times 10^{-4}$
5 clases	1	0,2026	2	128-64	0,4469	1	32	Adam	$8,9095 \times 10^{-4}$
5 clases	2	0,2026	4	128-128-64-64	0,4660	3	64-32-16	RMSprop	$2,8669 \times 10^{-2}$
5 clases	3	0,8040	2	128-32	0,4714	3	128-64-16	SGD	$2,7871 \times 10^{-4}$
5 clases	4	0,9893	3	64-32-32	0,2571	3	128-16-16	Adam	$9,0265 \times 10^{-5}$
5 clases	5	0,9813	2	64-16	0,3080	2	128-32	Adam	$4,9027 \times 10^{-5}$
5 clases	6	0,9933	4	128-128-32-32	0,4296	2	128-64	RMSprop	$4,6860 \times 10^{-4}$
5 clases	7	0,9666	3	128-16-16	0,3971	3	128-64-16	SGD	$8,2341 \times 10^{-2}$
10 clases	0	0,8026	2	128-128	0,1023	3	128-32-16	SGD	$9,3565 \times 10^{-5}$
10 clases	1	0,6579	4	128-32-32-16	0,2188	2	128-64	SGD	$7,7290 \times 10^{-5}$
10 clases	2	0,1013	2	16-16	0,1618	2	128-32	RMSprop	$8,1873 \times 10^{-3}$
10 clases	3	0,9893	4	128-64-16-16	0,3122	3	64-64-32	SGD	$4,3622 \times 10^{-3}$
10 clases	4	0,6346	2	128-32	0,1191	2	128-16	SGD	$9,4079 \times 10^{-5}$
10 clases	5	0,8939	2	64-32	0,4666	3	64-32-32	SGD	$1,0758 \times 10^{-3}$
10 clases	6	0,1073	2	16-16	0,2650	3	128-128-32	Adam	$2,3159 \times 10^{-2}$
10 clases	7	0,9440	2	32-32	0,1467	1	16	Adam	$7,0347 \times 10^{-5}$
20 clases	0	0,0480	3	16-16-16	0,1101	2	32-16	Adam	$9,9392 \times 10^{-3}$
20 clases	1	0,0553	3	64-64-64	0,3127	3	128-128-64	Adam	$1,9887 \times 10^{-2}$
20 clases	2	0,9906	3	128-64-32	0,4296	1	128	SGD	$3,0151 \times 10^{-2}$
20 clases	3	0,9866	2	64-16	0,2491	2	128-32	SGD	$8,6986 \times 10^{-3}$
20 clases	4	0,0473	4	128-64-32-32	0,4460	1	64	Adam	$7,9193 \times 10^{-2}$
20 clases	5	0,9079	2	64-16	0,4426	2	128-128	SGD	$2,5511 \times 10^{-4}$
20 clases	6	0,9219	3	64-16-16	0,2236	3	64-64-16	SGD	$3,8593 \times 10^{-3}$
20 clases	7	0,9883	2	64-16	0,4233	1	128	RMSprop	$3,9803 \times 10^{-5}$
26 clases	0	0,7607	3	128-64-16	0,2562	3	64-32-16	RMSprop	$1,2417 \times 10^{-5}$
26 clases	1	0,8953	2	128-128	0,1686	1	128	RMSprop	$7,3691 \times 10^{-3}$
26 clases	2	0,9917	4	128-128-32-32	0,2565	2	128-128	SGD	$1,2080 \times 10^{-3}$
26 clases	3	0,9148	3	64-16-16	0,4926	3	128-16-16	RMSprop	$2,2192 \times 10^{-4}$
26 clases	4	0,3235	2	64-16	0,2975	3	32-16-16	RMSprop	$1,5557 \times 10^{-5}$
26 clases	5	0,9617	3	128-32-16	0,1969	1	32	RMSprop	$2,0284 \times 10^{-4}$
26 clases	6	0.9934	3	128-128-64	0,4210	2	128-32	SGD	$1,3435 \times 10^{-2}$
26 clases	7	0,9941	3	128-64-64	0,3310	1	32	SGD	$4,2895 \times 10^{-2}$

Las Tabla 4.5, Tabla 4.6, Tabla 4.7 y Tabla 4.8 presentan las arquitecturas seleccionadas para cada uno de los 4 modelos base optimizados para la tarea de la clasificación del abecedario de la lengua de señas americana (*American Sign Language*) (ASL).

4.3.1. Arquitecturas de los Modelos Optimizados con Optuna

Para el diseño del modelo de 5 clases se implementó la prueba número 6, cuya arquitectura se detalla en la Tabla 4.5. Este modelo está conformado por cuatro capas convolucionales con la configuración de filtros 128-128-32-32, todas con función de activación $ReLU$ y un tamaño de kernel de 3×3, cada una seguida de normalización por lotes (*Batch Normalization*) y una capa de agrupamiento máximo (*Max Pooling*) de 2×2. Posteriormente, la extracción semántica de las imagenes se aplanan mediante una capa *Flatten* y se aplica una tasa de *Dropout* del 42.96 % para reducir el sobreajuste. Luego, se incorporan dos capas densas completamente conectadas con 128 y 64 unidades neuronales, respectivamente, ambas con activación $ReLU$, y finalmente una capa de salida con función de activación *Softmax*, que permite la clasificación en 5 clases. El entrenamiento de este modelo se llevó a cabo utilizando el optimizador *RMSprop* con una tasa de aprendizaje de 4.6860×10^{-4}.

En el caso del modelo de 10 clases se implementó la prueba número 3, cuya arquitectura se detalla en la Tabla 4.6. El diseño está compuesto por cuatro capas convolucionales con configuraciones de 128, 64, 16 y 16 filtros, respectivamente, todas con función de activación $ReLU$ y un tamaño de kernel de 3×3, cada una seguida de normalización por lotes (*Batch Normalization*) y una capa de agrupamiento máximo (*Max Pooling*) de 2×2. Tras el aplanamiento de las características mediante una capa *Flatten*, se aplicó un *Dropout* del 31.22 % para reducir el sobreajuste. Posteriormente, se añadieron tres capas densas con 64, 64 y 32 unidades neuronales, todas con activación $ReLU$, y finalmente una capa de salida con activación *Softmax* para la clasificación de las 10 clases. El entrenamiento de este modelo se realizó utilizando el optimizador *SGD* con una tasa de aprendizaje de 4.3622×10^{-3}, lo que permitió un ajuste eficiente y estable.

Tabla 4.5: Arquitectura modelo de 5 clases con optimizacion de hiper-parametros

Capa	Unidades Neuronales	Función de Activación	Tamano de kernel	Dropout
Convolutional	128	ReLu	3×3	N/A
Batch Normalization	N/A	N/A	N/A	N/A
Max Pooling	N/A	N/A	2×2	N/A
Convolutional	128	ReLu	3×3	N/A
Batch Normalization	N/A	N/A	N/A	N/A
Max Pooling	N/A	N/A	2×2	N/A
Convolutional	32	ReLu	3×3	N/A
Batch Normalization	N/A	N/A	N/A	N/A
Max Pooling	N/A	N/A	2×2	N/A
Convolutional	32	ReLu	3×3	N/A
Batch Normalization	N/A	N/A	N/A	N/A
Max Pooling	N/A	N/A	2×2	N/A
Flatten	N/A	N/A	N/A	N/A
Dropout	N/A	N/A	N/A	0.4296
Fully Connected	128	ReLu	N/A	N/A
Fully Connected	64	ReLu	N/A	N/A
Salida	5	Softmax	N/A	N/A

Tabla 4.6: Arquitectura modelo de 10 clases con optimizacion de hiper-parametros

Capa	Unidades Neuronales	Función de Activación	Tamano de kernel	Dropout
Convolutional	128	ReLu	3×3	N/A
Batch Normalization	N/A	N/A	N/A	N/A
Max Pooling	N/A	N/A	2×2	N/A
Convolutional	64	ReLu	3×3	N/A
Batch Normalization	N/A	N/A	N/A	N/A
Max Pooling	N/A	N/A	2×2	N/A
Convolutional	16	ReLu	3×3	N/A
Batch Normalization	N/A	N/A	N/A	N/A
Max Pooling	N/A	N/A	2×2	N/A
Convolutional	16	ReLu	3×3	N/A
Batch Normalization	N/A	N/A	N/A	N/A
Max Pooling	N/A	N/A	2×2	N/A
Flatten	N/A	N/A	N/A	N/A
Dropout	N/A	N/A	N/A	0.3122
Fully Connected	64	ReLu	N/A	N/A
Fully Connected	64	ReLu	N/A	N/A
Fully Connected	32	ReLu	N/A	N/A
Salida	10	Softmax	N/A	N/A

En el caso del modelo de 20 clases se implementó la prueba número 2, cuya arquitectura se detalla en la Tabla 4.7. Este diseño es más compacto y consta de tres capas convolucionales con 128, 64 y 32 filtros, respectivamente, todas con activación *ReLU* y un tamaño de kernel de 3×3, cada una seguida de normalización por lotes (*Batch Normalization*) y una capa de agrupamiento máximo (*Max Pooling*) de 2×2. Posteriormente, las características extraídas se aplanan mediante una capa *Flatten*, tras lo cual se aplica un *Dropout* del 42.96 % para reducir el sobreajuste. La red concluye con una capa densa de 128 unidades con activación *ReLU* y una capa de salida con activación *Softmax* para la clasificación de las 20 clases. El entrenamiento se realizó utilizando el optimizador *SGD* con una tasa de aprendizaje de 3.0151×10^{-2}, lo que permitió un ajuste rápido y efectivo del modelo.

Tabla 4.7: Arquitectura modelo de 20 clases con optimizacion de hiper-parametros

Capa	**Unidades Neuronales**	**Función de Activación**	**Tamano de kernel**	**Dropout**
Convolutional	128	ReLu	3×3	N/A
Batch Normalization	N/A	N/A	N/A	N/A
Max Pooling	N/A	N/A	2×2	N/A
Convolutional	64	ReLu	3×3	N/A
Batch Normalization	N/A	N/A	N/A	N/A
Max Pooling	N/A	N/A	2×2	N/A
Convolutional	32	ReLu	3×3	N/A
Batch Normalization	N/A	N/A	N/A	N/A
Max Pooling	N/A	N/A	2×2	N/A
Flatten	N/A	N/A	N/A	N/A
Dropout	N/A	N/A	N/A	0.4296
Fully Connected	128	ReLu	N/A	N/A
Salida	20	Softmax	N/A	N/A

Finalmente, para el modelo de 26 clases se seleccionó la prueba número 7, cuya arquitectura se muestra en la Tabla 4.8. Esta red está compuesta por tres capas convolucionales con 128, 64 y 64 filtros, respectivamente, todas con activación *ReLU* y un tamaño de kernel de 3×3, seguidas de normalización por lotes (*Batch Normalization*) y una capa de agrupamiento máximo (*Max Pooling*) de 2×2 después de cada convolución. Posteriormente, las características se aplanan mediante una capa *Flatten* y se aplica un *Dropout* del 33.10 % para reducir el sobreajuste. La arquitectura continúa con una capa densa de 32 unidades neuronales con activación *ReLU*, y concluye con una capa de salida con activación *Softmax*, que permite la

clasificación en las 26 clases. Para el entrenamiento se empleó el optimizador *SGD* con una tasa de aprendizaje de 4.2895×10^{-2}, garantizando un ajuste eficiente en un escenario de mayor complejidad clasificatoria.

Tabla 4.8: Arquitectura modelo de 26 clases con optimizacion de hiper-parametros

Capa	**Unidades Neuronales**	**Función de Activación**	**Tamano de kernel**	**Dropout**
Convolutional	128	ReLu	3×3	N/A
Batch Normalization	N/A	N/A	N/A	N/A
Max Pooling	N/A	N/A	2×2	N/A
Convolutional	64	ReLu	3×3	N/A
Batch Normalization	N/A	N/A	N/A	N/A
Max Pooling	N/A	N/A	2×2	N/A
Convolutional	64	ReLu	3×3	N/A
Batch Normalization	N/A	N/A	N/A	N/A
Max Pooling	N/A	N/A	2×2	N/A
Flatten	N/A	N/A	N/A	N/A
Dropout	N/A	N/A	N/A	0.3310
Fully Connected	32	ReLu	N/A	N/A
Salida	26	Softmax	N/A	N/A

Esta elección de hiper-parámetros permitió definir arquitecturas optimizadas en los modelos de distintas clases, mejorando considerablemente y de manera consistente el desempeño de los modelos, consumiendo una menor cantidad de recursos computacionales.

4.4. Desempeño computacional de los modelos optimizados

En el análisis de los resultados obtenidos en los modelos optimizados, se observan mejoras notables en la eficiencia computacional frente a los modelos base. Como se muestra en la Tabla 4.9, los tiempos de entrenamiento presentan reducciones consistentes en todos los escenarios. Por ejemplo, el modelo de 5 clases disminuye de 300,41 segundos en el modelo base Tabla 4.1 a 295,71 segundos en el modelo optimizado, tendencia que se repite en los modelos de 10, 20 y 26 clases. De igual forma, los tiempos de evaluación e inferencia también se reducen, lo que indica un procesamiento más rápido y eficiente. Estos resultados reflejan que la optimización

logra un impacto positivo en la velocidad de ejecución sin comprometer la precisión de los modelos.

Tabla 4.9: Métricas de desempeño computacional de modelos optimizados

Modelo	**Tiempo de entrenamiento (segundos)**	**Costo computacional (FLOPS)**	**Cantidad de parámetros**	**Tiempo de evaluación (segundos)**	**Tiempo de inferencia (segundos)**
5 clases	295,71	$1,558727646 \times 10^{-10}$	918.161	2,70	0,0008
10 clases	571,79	$1,558728316 \times 10^{-10}$	919.136	3,86	0,0006
20 clases	1131,99	$1,558729656 \times 10^{-10}$	921.086	5,26	0,0005
26 clases	1510,86	$1,558730460 \times 10^{-10}$	922.256	7,59	0,0005

Por otro lado, en términos de uso de recursos, la comparación entre la Tabla 4.2 y la Tabla 4.10 muestra que tanto el consumo de memoria RAM como el tamaño de los modelos se mantienen prácticamente iguales entre las versiones base y optimizadas. Además, el costo computacional en FLOPS y la cantidad de parámetros no presentan incrementos significativos tras la optimización, lo que evidencia un equilibrio entre eficiencia y consumo de recursos. Cabe destacar que la incorporación de optimizadores más especializados, como RMSprop y SGD en las versiones optimizadas, favoreció la estabilidad y la convergencia durante el entrenamiento, contribuyendo a modelos más robustos y escalables. En conjunto, los resultados demuestran que la optimización permite obtener un mejor desempeño computacional sin un mayor costo en recursos, lo cual resulta especialmente relevante para aplicaciones en entornos de producción con limitaciones de hardware.

Tabla 4.10: Uso de recursos computacionales de los modelos optimizados

Modelo	**Tarjeta**	**Consumo de ram (GB)**	**# Epoch**	**Batch size**	**Tamaño del modelo (MB)**	**Optimizador**
5 clases	GPU T4 x2	10,2	15	32	1,17	RMSprop
10 clases	GPU T4 x2	14,8	15	32	1,17	SGD
20 clases	GPU T4 x2	19,4	15	32	1,17	SGD
26 clases	GPU T4 x2	22,9	15	32	1,17	SGD

4.5. Resultados y Análisis de los Modelos Optimizados

El análisis de las matrices de confusión de los modelos optimizados (Figura 4.6, Figura 4.7, Figura 4.8, Figura 4.9) evidencia mejoras significativas respecto a los modelos base. En el caso del modelo de 5 clases, tanto en la versión base como en la optimizada, los errores son prácticamente inexistentes; sin embargo, en el modelo optimizado se eliminan por completo las confusiones puntuales observadas inicialmente, logrando un rendimiento perfecto.

Figura 4.6: Tabla de confusión modelo optimizado de 5 clases con el conjunto de prueba. Elaboración propia.

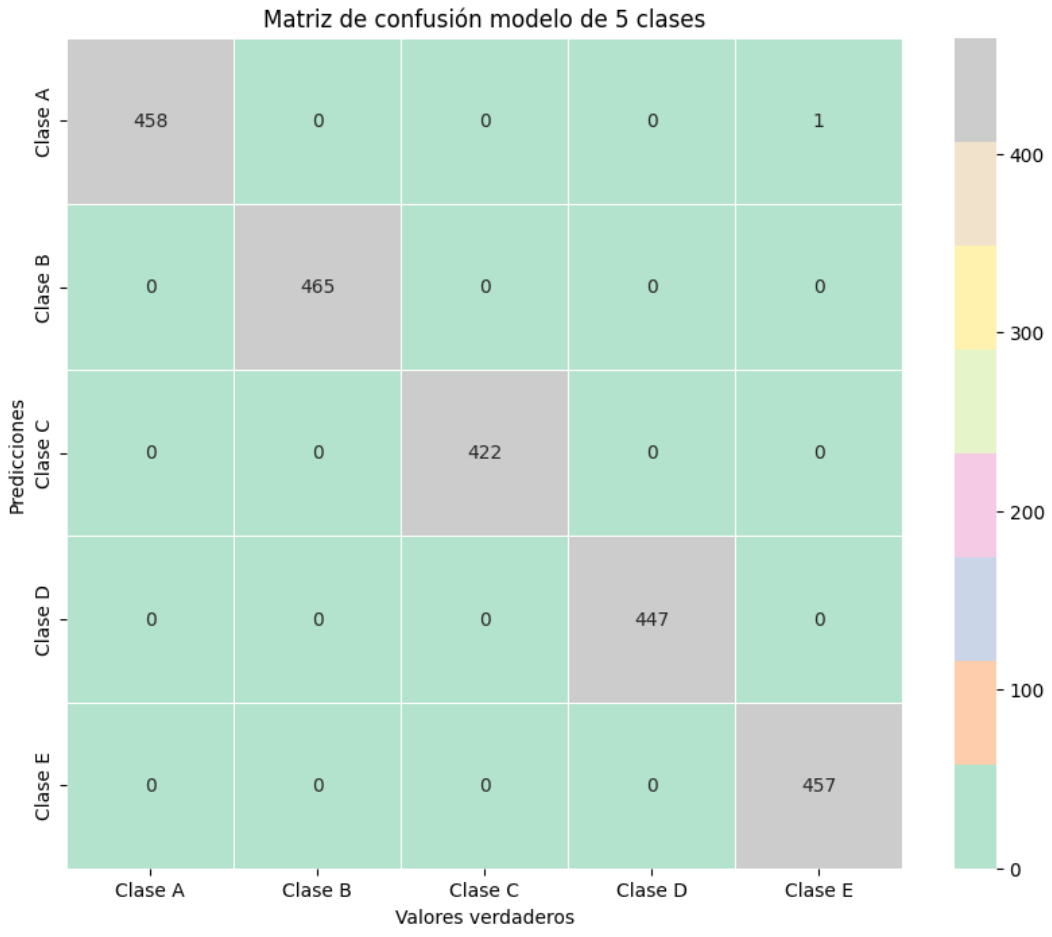


Figura 4.7: Tabla de confusión modelo optimizado de 10 clases con el conjunto de prueba. Elaboración propia.

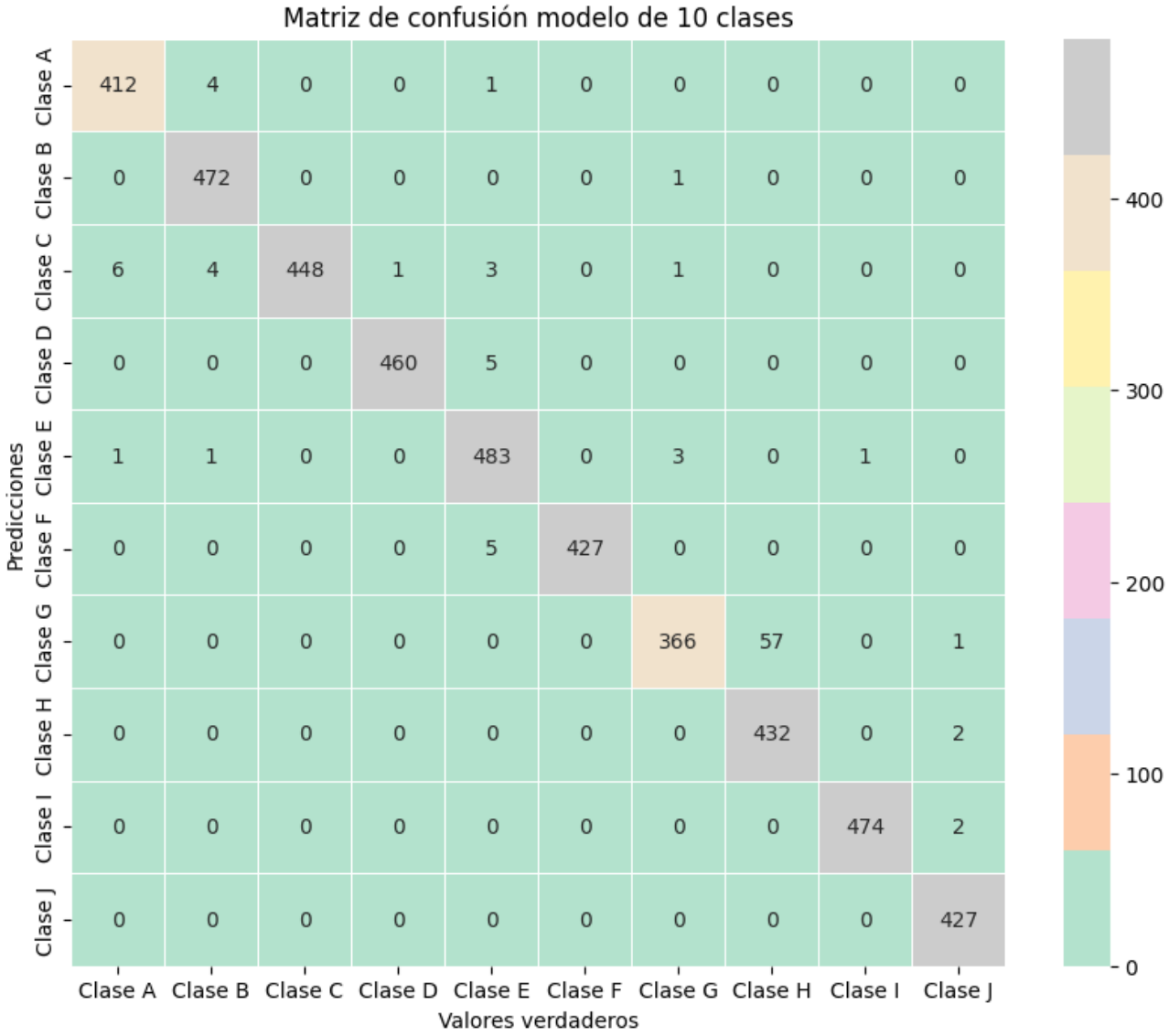


Figura 4.8: Tabla de confusión modelo optimizado de 20 clases con el conjunto de prueba. Elaboración propia.

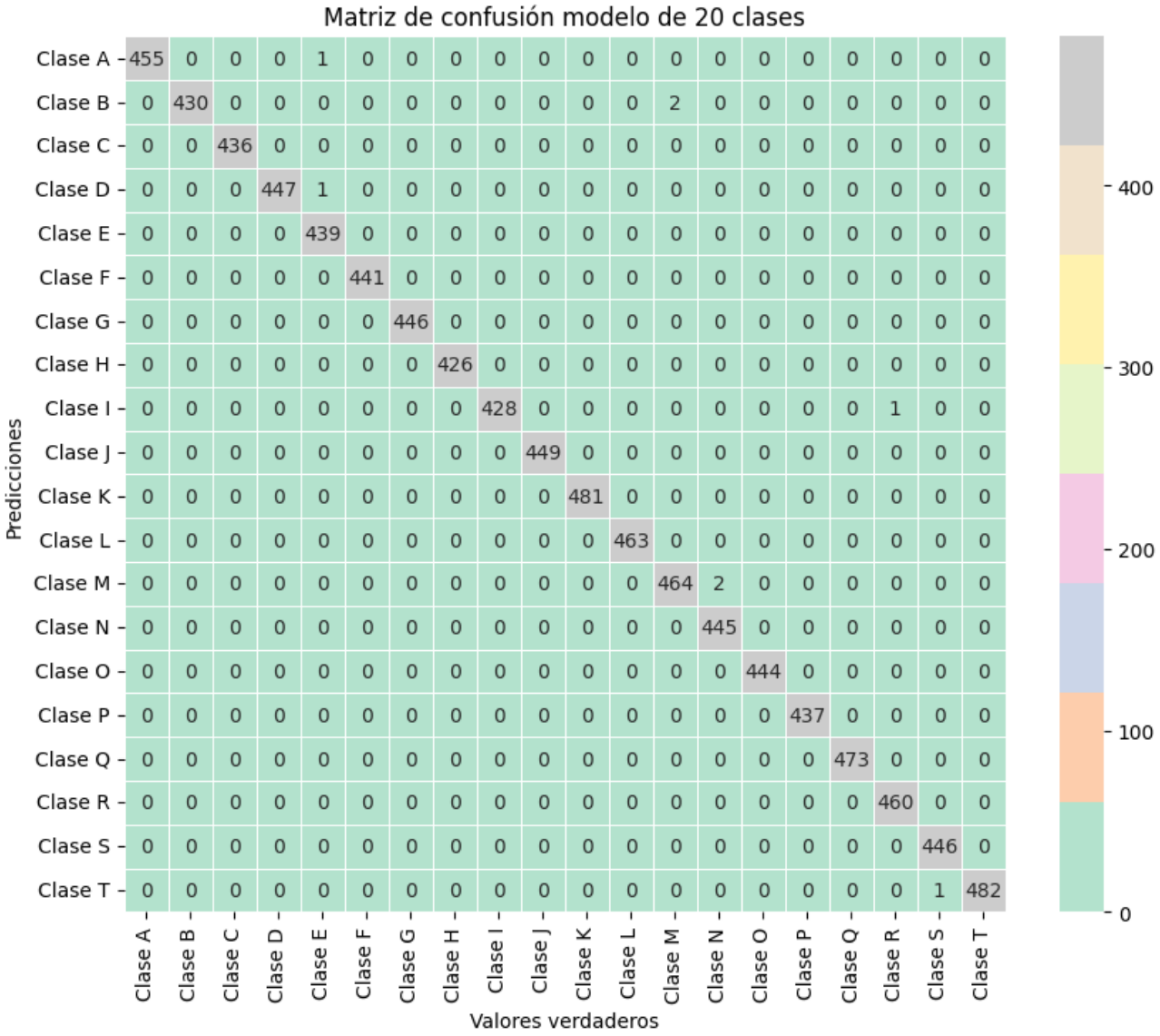
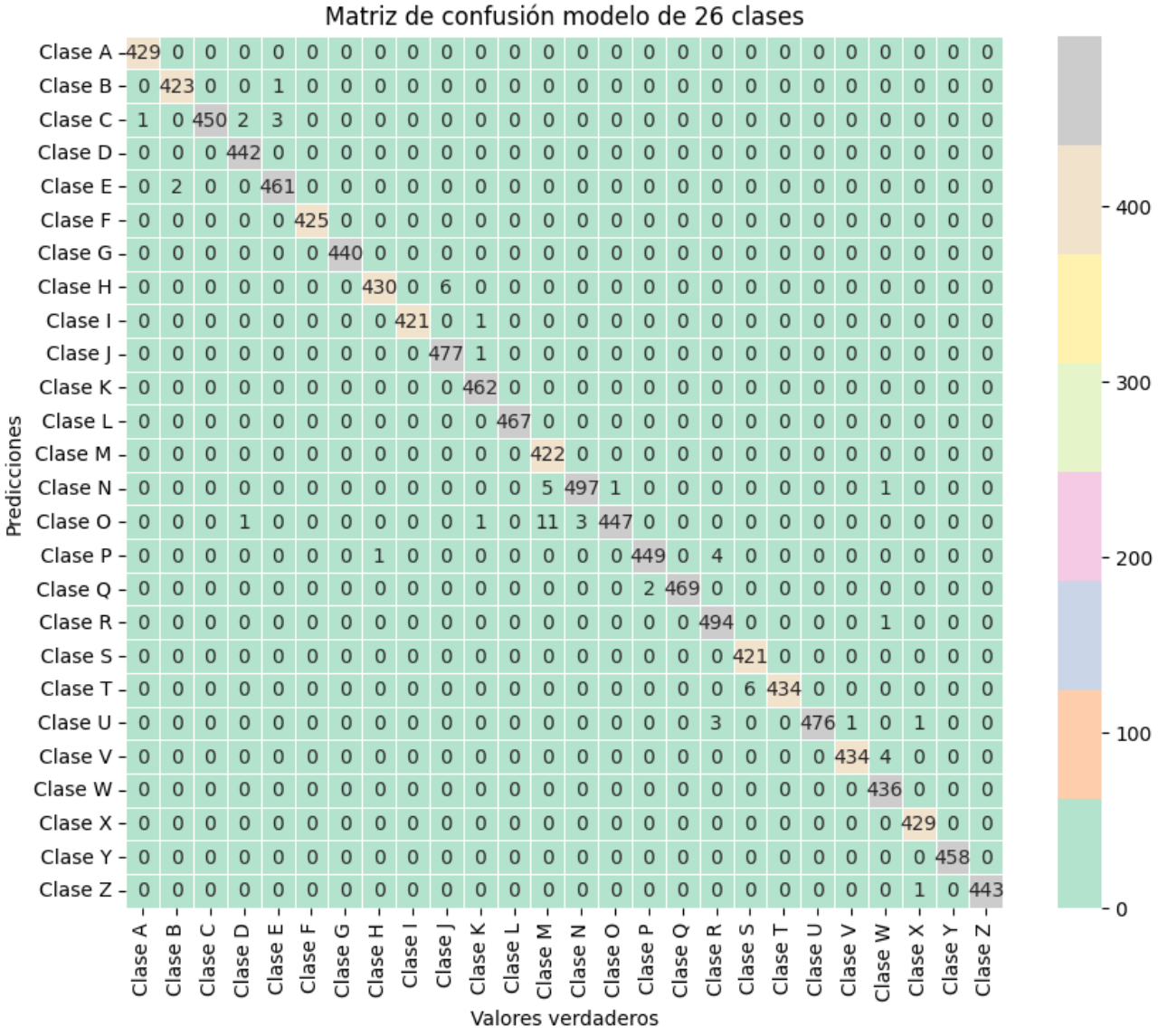


Figura 4.9: Tabla de confusión modelo optimizado de 26 clases con el conjunto de prueba. Elaboración propia.



En el modelo de 10 clases (Figura 4.7), las confusiones más relevantes del modelo base, como la mezcla entre las letras I y E, se reducen de manera considerable en el modelo optimizado, mostrando una distribución de errores más controlada y localizada. De forma similar, en el modelo de 20 clases (Figura 4.8) se observa una mejora clara: la confusión marcada entre las clases P y Q en el modelo base disminuye significativamente en la versión optimizada, lo que refleja un aprendizaje más robusto y estable.

Finalmente, en el modelo de 26 clases (Figura 4.9), los errores dispersos que afectaban a letras como B, I o X se reducen notablemente en el modelo optimizado. Aunque persisten confusiones puntuales, estas son mucho menores en frecuencia e impacto que en el modelo base. En conjunto, estos resultados confirman que la optimización no solo mejora las métricas globales, sino que también corrige patrones de error específicos, fortaleciendo la capacidad del sistema para realizar una clasificación más precisa en escenarios multiclase.

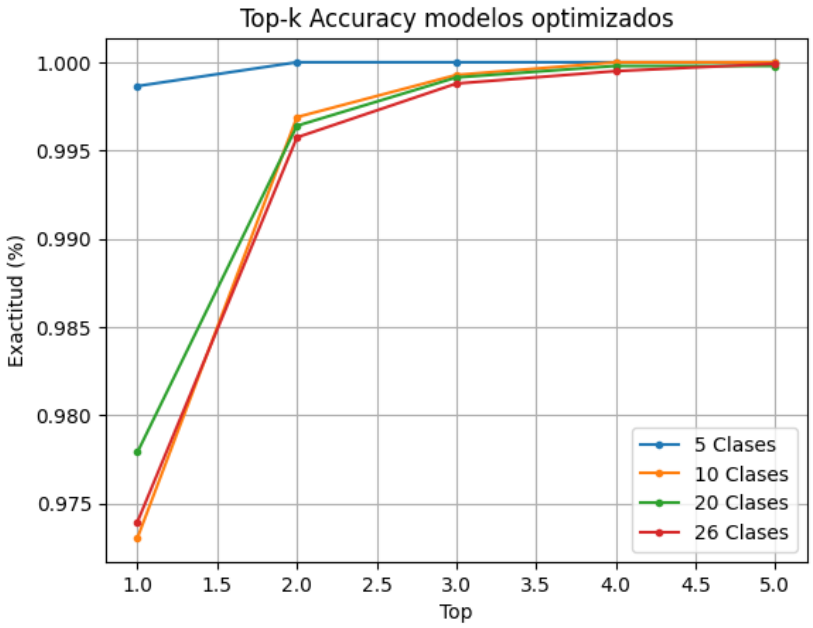
Tabla 4.11: Métricas de los modelos optimizados con el conjunto de prueba

Métrica	**Modelo 5 clases**	**Modelo 10 clases**	**Modelo 20 clases**	**Modelo 26 clases**
Exactitud [%]	99,95	97,80	99,91	99,45
Top-k [%]	99,99	99,69	99,99	99,97

En la Tabla 4.11 se presentan los resultados de exactitud alcanzados por los modelos optimizados en el conjunto de prueba. Se observa que, en general, todos los modelos logran un desempeño elevado, con valores superiores al 97 %. El modelo de 5 clases alcanza la exactitud más alta (99,95 %), seguido muy de cerca por el de 20 clases (99,91 %). Por otro lado, el modelo de 10 clases presenta el valor más bajo (97,80 %), lo cual sugiere que el incremento en la complejidad de clases no siempre implica un deterioro del rendimiento, sino que depende de cómo los datos se distribuyen entre las categorías. Finalmente, el modelo de 26 clases mantiene una exactitud considerable (99,45 %), demostrando la robustez de los algoritmos incluso en escenarios de mayor granularidad.

En cuanto al análisis del top-k, los resultados evidencian una mejora sustancial en la capacidad de los modelos para incluir la clase correcta dentro de sus primeras predicciones. Tal como se ilustra en la Figura 4.10, la métrica top-5 muestra valores cercanos al 100 % en todos los casos, lo que significa que la verdadera clase del ejemplo

Figura 4.10: Exactitud Top-k de los modelos base para diferentes modelos con el conjunto de prueba



de prueba se encuentra casi siempre entre las cinco etiquetas con mayor probabilidad asignada por el modelo. Esto resalta la utilidad del top-k como métrica complementaria a la exactitud, ya que refleja un mejor panorama del rendimiento en aplicaciones donde no necesariamente se requiere que la primera predicción sea la correcta, sino que esté dentro de un rango reducido de opciones confiables.

4.6. Discusión de resultados modelos base y optimizados con el conjunto de prueba

Los resultados obtenidos para los modelos base y sus versiones optimizadas corresponden exclusivamente al **conjunto de prueba**, el cual representa aproximadamente el 15 % del total de los datos (alrededor de 450 muestras por clase). Es importante destacar que el conjunto de validación fue utilizado para la activación de mecanismos de *parada temprana* en función del número de iteraciones. De este modo, aunque existe la posibilidad de que los datos de validación introduzcan cierto

sesgo, la muestra del conjunto de prueba es lo suficientemente representativa como para garantizar que los resultados reportados reflejen la capacidad real de generalización de los modelos y eviten el sobreentrenamiento.

Es importante aclarar que las versiones optimizadas de las matrices de confusión mostradas en las Figura 4.6, Figura 4.7, Figura 4.8 y Figura 4.9 se revelan mejoras significativas en el desempeño del modelo tras el proceso de optimización. Estas mejoras se reflejan principalmente en la reducción de errores de clasificación, un aumento en la exactitud de las predicciones y una menor confusión entre clases similares. Uno de los aspectos más notables es el incremento en la cantidad de predicciones correctas, representadas por valores más altos en la diagonal principal de las matrices optimizadas. Esto indica que el modelo es capaz de reconocer correctamente un mayor número de ejemplos en cada categoría, reduciendo así la cantidad de falsos positivos y falsos negativos.

El análisis de las métricas de desempeño evidencia claramente las ventajas de la optimización de los modelos. En la Figura 4.11 se muestra que en términos de la exactitud, los modelos optimizados superan a los modelos base en todos los casos, destacándose el modelo de 26 clases con un 99,45 % frente al 95,11 %. Esta mejora se puede observar en la Figura 4.12 con la métrica *Top-k*, donde los modelos optimizados alcanzan valores cercanos al 100 %, demostrando una mayor capacidad para clasificar correctamente dentro de las primeras predicciones.

Figura 4.11: Exactitud Top-k de los modelos base para diferentes modelos con el conjunto de prueba

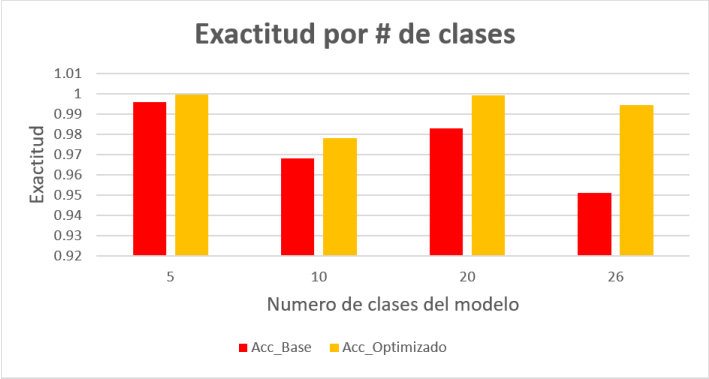
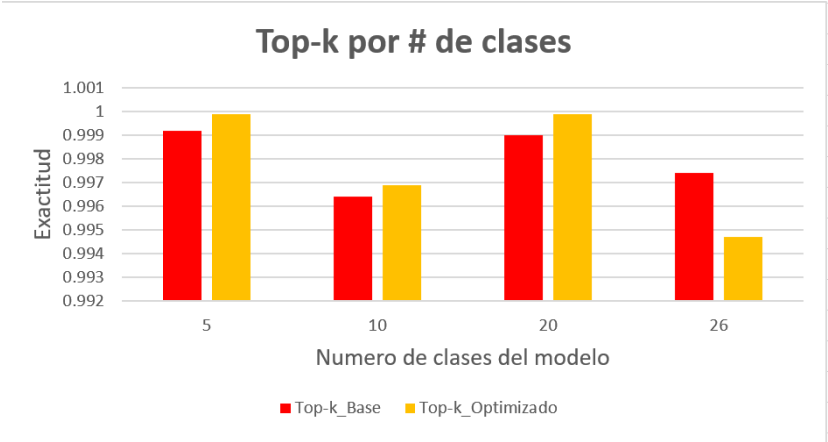


Figura 4.12: Exactitud Top-k de los modelos base para diferentes modelos con el conjunto de prueba



El proceso de optimización aporta ventajas significativas. Primero, permite adaptar los modelos a diferentes niveles de complejidad (5, 10, 20 o 26 clases) sin comprometer la exactitud, incluso en escenarios con limitaciones de hardware. Segundo, al reducir el tiempo de inferencia y el tamaño del modelo, se facilita la implementación en dispositivos con recursos limitados. Estos resultados evidencian cómo la optimización no solo mejora la robustez de los modelos, sino que también potencia su aplicabilidad en contextos reales, promoviendo una mayor accesibilidad para soluciones de clasificación del alfabeto dactilológico de la lengua de señas americana (*American Sign Language*) (ASL).

En el análisis de la métrica *F1-score* se identificaron limitaciones en la librería *Torchmetrics* utilizada, particularmente al calcular esta métrica con valores de *top-k* superiores a 1. Se evidenció que los resultados presentaban inconsistencias, especialmente porque el *recall* se mantenía en 1 mientras que la *precisión* descendía de forma anómala, lo cual no era coherente con el comportamiento esperado. Tras varias verificaciones, incluyendo pruebas de escritorio y consultas apoyadas por la AI a través de Grok, se concluyó que el problema provenía de la implementación de la librería, por lo que se optó por excluir los resultados de *F1-score* para valores de $k > 1$.

Al concentrar el análisis únicamente en el caso de $k = 1$, los valores de *F1-score* se mantuvieron estables y cercanos a 1 en la mayoría de configuraciones, lo que confirma

la coherencia con las métricas de exactitud y *top-k accuracy*. Este resultado indica que, cuando se evalúa la predicción más probable, los modelos muestran un alto balance entre precisión y recall. Sin embargo, al aumentar el valor de k, se introducen falsos positivos adicionales que afectan la precisión y, en consecuencia, reducen el valor del *F1-score*.

En síntesis, el comportamiento observado refleja que el *F1-score* es una métrica sensible a los errores de precisión en escenarios con múltiples predicciones posibles, lo cual explica la tendencia decreciente en los valores representados en la Figura correspondiente. Por este motivo, el análisis final se centra en los resultados de $k = 1$, donde se garantiza consistencia y se puede evaluar de forma más confiable el desempeño real de los modelos base. Esto sugiere que, aunque los modelos base son capaces de manejar un mayor número de clases gradualmente, su capacidad para predecir correctamente dentro de las primeras k opciones se ve ligeramente afectada por la complejidad añadida. En otras palabras, se puede concluir que a mayor k, mayor es la probabilidad de que el elemento esté dentro del vector de predicciones.

4.7. Selección del modelo final

Durante la búsqueda de un modelo convolucional adecuado se presentaron diferentes dificultades, especialmente en la definición de arquitecturas que ofrecieran un equilibrio entre exactitud y eficiencia computacional. La implementación inicial propuesta por Lara-Cázares sirvió como punto de partida, ya que permitió identificar configuraciones base robustas sobre las cuales se aplicaron ajustes y procesos de optimización de hiper-parámetros. A partir de este enfoque, se logró establecer una arquitectura final capaz de clasificar satisfactoriamente las 26 letras del alfabeto dactilológico de la lengua de señas americana (*American Sign Language*) (ASL), superando los retos asociados al incremento progresivo del número de clases.

En este trabajo, la métrica priorizada para guiar la optimización de hiper-parámetros fue la *accuracy* (exactitud), dado que el conjunto de datos utilizado presentaba un balance adecuado entre clases, con diferencias mínimas en el número de muestras por categoría. Esto permitió que la exactitud fuera una medida representativa y confiable

del desempeño global del clasificador. No obstante, para complementar y profundizar el análisis, también se incluyeron matrices de confusión y tablas de resultados que evidencian de manera más detallada los aciertos y errores en cada clase, garantizando así una evaluación integral del modelo.

Los resultados de esta investigación confirman la efectividad de los modelos CNN *ad-hoc* en la clasificación de gestos estáticos del alfabeto de la lengua de señas americana (*American Sign Language*) (ASL). El preprocesamiento de las imágenes, incluyendo la normalización y el uso de "one-hot encoding" para las etiquetas, fue clave para garantizar la estabilidad y precisión durante el entrenamiento. Asimismo, la optimización de hiper-parámetros mediante Optuna demostró ser una herramienta eficaz, permitiendo obtener modelos con configuraciones que maximizaron la exactitud sin superar significativamente el costo computacional, haciendo viables sus aplicaciones en dispositivos de recursos limitados. Finalmente, el modelo seleccionado corresponde a la prueba número 7 de 26 clases, cuya arquitectura se muestra en la Tabla 4.8. Este modelo no solo alcanzó un desempeño competitivo en términos de exactitud, sino que además mantuvo un consumo controlado de recursos y una buena capacidad de generalización, lo que lo convierte en la mejor respuesta al problema planteado en esta investigación.

Capítulo 5

Conclusiones

5.1. Conclusiones Generales

El presente trabajo tuvo como objetivo evaluar la capacidad de modelos ad hoc basados en redes neuronales convolucionales (CNN) para clasificar las letras del alfabeto dactilológico de la lengua de señas americana (*American Sign Language*) (ASL). A lo largo del desarrollo de la investigación, se implementaron estrategias de optimización y preprocesamiento, logrando resultados prometedores en términos de desempeño y adaptabilidad del modelo a diferentes niveles de complejidad con 5, 10, 20 y 26 clases respectivamente. El diseño e implementación de los modelos demostró que es posible abordar con éxito problemas de clasificación en un entorno con múltiples clases utilizando arquitecturas convolucionales ajustadas a las necesidades del conjunto de datos. La normalización de las imágenes y la generación de etiquetas codificadas en vectores únicos ad-hoc jugaron un papel crucial para garantizar la consistencia y la estabilidad durante el entrenamiento.

En particular, se hizo énfasis en la forma en que, a partir de una secuencia de imágenes (frames), se realizó la clasificación de las imágenes pertenecientes al alfabeto dactilológico de la lengua de señas americana (*American Sign Language*) (ASL), en las cuales se implementaron diversos modelos de 5, 10, 20 y 26 clases consecuentemente y fueron presentados en el Capítulo 4. Esta primera etapa en la investigación permitió alcanzar el objetivo general.

Posteriormente, se hizo una revisión a la literatura con el fin de determinar las estrategias que habían sido utilizadas por diferentes autores para la clasificación de gestos estáticos del ASL mediante técnicas de inteligencia artificial, más específicamente en el área del aprendizaje profundo, con la implementación de modelos *ad-hoc* simples,

obteniendo así un punto de partida para el desarrollo de la metodología que se seguiría y que se presentó en el Capítulo 3. Asimismo, se presentaron las herramientas teóricas que se necesitaron para abordar el problema en cuestión.

Teniendo claro el funcionamiento general del sistema que se pretendía construir, se continuó con el diseño del software que ejecutaría cada una de las funciones especificadas previamente por el diagrama de bloques que se presentó en la Figura 3.1. Después, diversas pruebas fueron llevadas a cabo utilizando los datos de la base de datos; primero se entrenó la red convolucional *ad hoc*, luego se implementó la optimización de hiper-parámetros con el optimizador Optuna, y finalmente se volvieron a entrenar los modelos, obteniendo una mejora en los rendimientos y en la simplificación de las mismas.

Una de las principales contribuciones de este trabajo fue demostrar la efectividad de las arquitecturas *ad-hoc* para capturar características discriminativas en un conjunto de datos de imágenes estáticas, logrando una clasificación robusta incluso en clases visualmente similares. El análisis de las métricas de desempeño, presentado en la Tabla 4.1 y Tabla 4.9 del documento, evidenció una mejora significativa en la cantidad de parámetros, costo computacional, tiempo de entrenamiento y otros indicadores clave tras la optimización de hiper-parámetros mediante Optuna. Estas mejoras subrayan la importancia de explorar configuraciones personalizadas y la ventaja de realizar ajustes iterativos en los modelos. En comparación con los modelos base sin optimizar, los modelos optimizados no solo lograron un desempeño superior, sino que también destacaron por su eficiencia en términos de tiempo de inferencia y uso de recursos computacionales.

Finalmente, se concluye que los modelos de redes neuronales convolucionales basados en modelos ad-hoc son una herramienta robusta y eficiente para abordar problemas de reconocimiento de secuencias en imágenes, como la clasificación del alfabeto ASL. Este trabajo no solo contribuye al desarrollo de herramientas que pueden facilitar la inclusión social de personas con discapacidad auditiva, sino que también sienta las bases para futuras investigaciones en el campo del reconocimiento de gestos dinámicos, donde la interacción humano-computadora puede verse profundamente enriquecida por el avance en técnicas de aprendizaje profundo.

5.2. Trabajos a Futuro

- Si bien este trabajo se centró en gestos estáticos, una extensión natural sería abordar el reconocimiento de gestos dinámicos, que son esenciales para captar movimientos complejos en la lengua de señas americana (*American Sign Language*) (ASL). Esto podría lograrse mediante el uso de modelos híbridos que combinen redes convolucionales (CNN) para la extracción de características espaciales con redes neuronales recurrentes (RNN) o *transformers* (un tipo de arquitectura de redes de aprendizaje profundo) para capturar la información temporal en secuencias de video.

- Los modelos desarrollados podrían ser optimizados para aplicaciones en tiempo real, integrándolos en dispositivos móviles o sistemas embebidos. Esto requeriría un enfoque en la reducción de la complejidad computacional, utilizando técnicas como la poda de redes neuronales y la cuantización, para garantizar tiempos de inferencia rápidos y menor consumo de recursos.

- Se podría ampliar el conjunto de datos con imágenes capturadas en entornos no controlados y con variabilidad en la iluminación, perspectiva y contextos de fondo. Esto permitiría entrenar modelos más robustos y generalizables, mejorando su desempeño en condiciones reales. Además, se podrían explorar métodos de aumento de datos más avanzados y generar conjuntos de datos sintéticos para entrenar modelos en gestos menos representados.

Referencias

[1] M. Karam, «A framework for research and design of gesture-based human-computer interactions,» Tesis doct., University of Southampton, 2006.

[2] M. Marschark, P. Spencer y P. Nathan, *The Oxford Handbook of Deaf Studies, Language, and Education, Vol. 2* (Oxford Library of Psychology). Oxford University Press, 2010, ISBN: 9780199741816.

[3] E. T. Hall, *The Silent Language*, Vol 3. USA: Doubleday y Company, New York, 1959, ISBN: 1617294438.

[4] V. Bheda y D. Radpour, «Using Deep Convolutional Networks for Gesture Recognition in American Sign Language,» 2017, ISSN: 2331-8422. arXiv: 1710.06836.

[5] P. V. S. M. S. Kartik, K. B. V. N. S. Sumanth, V. N. V. S. Ram y P. Prakash, *Sign language to text conversion using deep learning*. Springer Singapore, 2021, vol. 145, págs. 219-227, ISBN: 9789811573446. DOI: 10.1007/978-981-15-7345-3_18.

[6] H. Cooper, B. Holt y R. Bowden, «Sign Language Recognition,» en Springer London, 2011, págs. 539-562. DOI: 10.1007/978-0-85729-997-0_27.

[7] G. W. H. Organization, *World report on hearing*. Wiley Publishing, 2021, https://www.univalle.edu.co/, ISBN: 0470035617.

[8] E. Chacón[1], D. Aguilar[2] y F. Sáenz[3], *Desarrollo de una Interfaz para el Reconocimiento Automático del Lenguaje de Signos **.

[9] W. C. Stokoe, *Sign Language Structure*, 1980.

[10] A. Lara-Cázares, M. A. Moreno-Armendáriz y H. Calvo, «Advanced Hybrid Neural Networks for Accurate Recognition of the Extended Alphabet and Dynamic Signs in Mexican Sign Language (MSL),» *Applied Sciences (Switzerland)*, vol. 14, 22 nov. de 2024, ISSN: 20763417. DOI: 10.3390/app142210186.

[11] J. Maccormick, *How does the Kinect work?* Vol 6. Dickinson College, 2011.

[12] M. R. Sadik, R. I. Sony, N. N. I. Prova et al., *Computer Vision Based Bangla Sign Language Recognition Using Transfer Learning*. Institute of Electrical y Electronics Engineers (IEEE), jul. de 2024, págs. 1-7, ISBN: 9798350381665. DOI: 10.1109/icdsis61070.2024.10594269.

[13] B. Sonare, A. Padgal, Y. Gaikwad y A. Patil, «Video-based sign language translation system using machine learning,» en *2021 2nd International Conference for Emerging Technology, INCET 2021*, Institute of Electrical y Electronics Engineers Inc., mayo de 2021, ISBN: 9781728170299. DOI: 10.1109/INCET51464.2021.9456176.

[14] N. Alrobah y A. Selmi, *Digits Recognition for Sign Language Using Transfer Learning Models*. Institute of Electrical y Electronics Engineers Inc., 2023, ISBN: 9798350312546. DOI: 10.1109/ICOA58279.2023.10308819.

[15] R. G. Rajan y M. J. Leo, *American Sign Language Alphabets Recognition using Hand Crafted and Deep Learning Features*. Institute of Electrical y Electronics Engineers Inc., feb. de 2020, págs. 430-434, ISBN: 9781728146850. DOI: 10.1109/ICICT48043.2020.9112481.

[16] A. P. Engelbrecht, *Computational Intelligence: An Introduction*, 2nd. Wiley Publishing, 2007, ISBN: 0470035617.

[17] F. A. Salazar Vasquez, «Implementacion Aprendizaje para la Arquitectura Deep Learning,» pág. 69, 2017.

[18] L. Alzubaidi, J. Zhang, A. J. Humaidi et al., «Review of deep learning: concepts, CNN architectures, challenges, applications, future directions,» vol. 8, n.º 1, 2021, ISSN: 2196-1115. DOI: 10.1186/s40537-021-00444-8.

[19] F. Chollet, *Deep Learning with Python*, 1st. USA: Manning Publications Co., 2017, ISBN: 1617294438.

[20] E. F. C. Bravo y J. A. L. Sotelo, *UNA APROXIMACIÓN PRÁCTICA A LAS REDES NEURONALES ARTIFICIALES*. Universidad del Valle, 2009, ISBN: 9789586707671. DOI: 10.25100/peu.64.

[21] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow : concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Inc., 2019, pág. 819, ISBN: 9781492032649.