

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE



“IMPLEMENTACIÓN Y ADAPTACIÓN DE UN
ALGORITMO DE RECONOCIMIENTO DE LENGUA DE
SEÑAS EN VIDEOS PARA DISPOSITIVOS ANDROID,
ENFOCADO EN LENGUA DE SEÑAS CHILENAS (LSCH):
CASO DIALOGAPP”

VICENTE GUSTAVO CUEVAS MORALES

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Carolina Saavedra
Profesor Correferente: Pedro Godoy

Julio - 2026



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: Implementación y adaptación de un algoritmo de reconocimiento de Lengua de Señas en videos para dispositivos Android, enfocado en Lengua de Señas Chilenas (LSCh): Caso DialogApp

Nombre del candidato(a): Vicente Gustavo Cuevas Morales

Carrera / Grado: Ingeniería civil en informática

Campus: Casa Central Valparaíso **Departamento:** Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Carolina Veronica Saavedra Ruiz, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☒ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por (**marcar una opción**):

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 10-7-2026 **Firma:** 

Estudiante o Candidato(a):

Fecha: 10-07-2026 **Firma:** 

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

DEDICATORIA

A mi familia, pareja, amigos y todos aquellos que me han formado como persona y profesional.

Dedicado también a todas las minorías que enfrentan la indiferencia de la sociedad, que esta memoria sea un aporte, por pequeña que sea, hacia un mundo más inclusivo.

AGRADECIMIENTOS

A mis padres, Carolina y Arturo, por su apoyo, amor, sacrificio y paciencia durante mis extendidos años de estudio, gracias por confiar en mí y motivarme a seguir adelante. A mi hermana Judith y mi sobrina Isidora, quienes probablemente sin saber, me han inspirado a seguir adelante a pesar de las dificultades de la vida. A mi familia les debo la persona que soy hoy, y espero poder retribuirles todo lo que me han dado.

Quiero agradecer a mis amigos y compañeros de carrera, con quienes compartí momentos de alegría, y que hicieron de esta etapa una experiencia inolvidable, gracias por hacer más llevadera la vida universitaria, obligarme a asistir a clases y recibirme en sus hogares. A José Miguel, Camilo, Matia, Tomás, Maximiliano, Cristian, Christopher, Sebastián, Ignacio, Paulo, Ariane, Javier, Felipe, Ismael y todos los demás, gracias por generar comunidad en un mundo individualista.

A mi pareja, Vivian, por su amor, cariño, paciencia y apoyo incondicional, gracias por estar a mi lado en los momentos difíciles y por celebrar conmigo los logros alcanzados. También por insistirme en que terminara esta memoria, y por ayudarme a mantener la motivación y la perspectiva en todo momento. Gracias por recordarme que la vida es un camino que se disfruta mejor acompañado.

Gracias a mis amigos de la infancia, Andrei, Nicolás y Sebastián, por su amistad incondicional a lo largo de estos años aun cuando los rechazara por estar ocupado con mis estudios, gracias por su apoyo y por recordarme la importancia de la amistad y la lealtad.

Mi gratitud también para mis amigos de la adolescencia, quienes me acompañaron en mis primeros pasos en la vida adulta y que siempre me han recibido con los brazos abiertos, a pesar del tiempo.

Mi reconocimiento también a los académicos, profesores y administrativos del Departamento de Informática, quienes día a día se esfuerzan por enseñar, educar y apoyar en un ambiente cercano y humano.

Al Estado de Chile y sus contribuyentes, por financiar mi educación y permitir que personas como yo puedan acceder a la educación superior y cumplir sus sueños.

RESUMEN

Resumen— Esta memoria presenta la implementación y optimización de un sistema de reconocimiento automático del alfabeto dactilológico de la Lengua de Señas Chilena (LSCh) para Android, superando el módulo de reconocimiento desarrollado previamente en el marco de la Feria de Software de la Universidad Técnica Federico Santa María. Ante la falta de recursos públicos, se construyó un dataset de 1.350 videos que abarca señas estáticas y dinámicas.

La solución emplea la extracción de puntos de referencia 3D mediante MediaPipe y una ingeniería de características basada en distancias inter-digitales sobre coordenadas métricas, reemplazando la representación de deltas de imagen del sistema original y permitiendo distinguir señas con morfologías similares. El motor predictivo utiliza una arquitectura híbrida Conv1D y GRU Bidireccional. Bajo un protocolo de validación cross-signer —generalización a usuarios no vistos en entrenamiento— el sistema propuesto alcanza un 85,3 % de accuracy, superando al modelo de referencia en 15,0 puntos porcentuales (test de Wilcoxon pareado, $p < 0,001$); en evaluación intra-signer el accuracy asciende a 98,9 %. La mejora se obtiene además con menor latencia computacional. En un Samsung Galaxy S23 (MediaPipe GPU), la propuesta alcanza un pipeline completo de 29,9 ms frente a 33,9 ms del original (−12 %), con inferencia TFLite un 24 % más rápida (10,2 ms frente a 13,5 ms), logrando un throughput de ≈ 33 fps suficiente para reconocimiento en tiempo real. En un Samsung Galaxy S20 FE 5G (MediaPipe CPU), la latencia de pipeline es de 76–81 ms, dominada en un 87–89 % por la detección de landmarks MediaPipe, con inferencia TFLite un 14 % más rápida que el original (8,6 ms frente a 10,0 ms).

Palabras Clave— Lengua de Señas Chilena; Reconocimiento de Señas; Deep Learning; Android Nativo; MediaPipe; GRU Bidireccional.

ABSTRACT

Abstract— This thesis presents the implementation and optimization of an automatic recognition system for the Chilean Sign Language (LSCh) finger-spelling alphabet on Android devices. Due to the lack of public resources, a novel dataset of 1,350 video clips was built, covering both static and dynamic signs.

The proposed methodology utilizes 3D landmark extraction via MediaPipe and a feature engineering stage based on inter-digital distances over metric coordinates, enabling the

discrimination of signs with similar hand shapes. The predictive engine is based on a hybrid architecture of Conv1D and Bidirectional GRU layers. Under a cross-signer validation protocol —generalization to users unseen during training— the system achieves 85.3 % accuracy, outperforming the reference model by 15.0 percentage points (paired Wilcoxon test, $p < 0,001$), while reaching 98.9 % under intra-signer evaluation. The native Android integration was evaluated on two devices. On a Samsung Galaxy S23 (MediaPipe GPU), the proposed model achieves a full-pipeline latency of 29.9 ms versus 33.9 ms for the baseline (−12 %), with TFLite inference 24 % faster (10.2 ms vs. 13.5 ms), reaching ≈ 33 fps throughput sufficient for real-time recognition. On a Samsung Galaxy S20 FE 5G (MediaPipe CPU), pipeline latency is 76–81 ms with TFLite inference 14 % faster than the baseline (8.6 ms vs. 10.0 ms), with MediaPipe landmark detection accounting for 87–89 % of total latency.

Keywords— Chilean Sign Language; Sign Language Recognition; Deep Learning; Native Android; MediaPipe; Bidirectional GRU.

GLOSARIO

ASOCH: Asociación de Sordos Chilenos.

LSCh: Lengua de Señas Chilena.

ML: *Machine Learning* o Aprendizaje de Máquinas.

RNN: *Recurrent Neural Network* o Redes Neuronales Recurrentes.

GRU: *Gated Recurrent Unit* o Unidad Recurrente con Compuertas.

CNN: *Convolutional Neural Network* o Redes Neuronales Convolucionales.

SLR: *Sign Language Recognition* o Reconocimiento de Lengua de Señas.

TFLite: TensorFlow Lite.

GPU: *Graphics Processing Unit* o Unidad de Procesamiento Gráfico.

FPS: *Frames Per Second* o Fotogramas por Segundo.

Roi: *Region of Interest* o Región de Interés.

ÍNDICE DE CONTENIDOS

RESUMEN	IV
ABSTRACT	IV
GLOSARIO	V
ÍNDICE DE FIGURAS	VIII
ÍNDICE DE TABLAS	VIII
INTRODUCCIÓN	1
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	4
1.1 Contexto	4
1.2 Motivación	5
1.3 Definición del problema	6
1.4 Objetivos	8
1.4.1 Objetivo General	8
1.4.2 Objetivos Específicos	9
CAPÍTULO 2: MARCO CONCEPTUAL	10
2.1 Aprendizaje Automático (Machine Learning)	10
2.1.1 Aprendizaje Supervisado	10
2.2 Redes Neuronales Artificiales (RNA)	11
2.2.1 La Neurona Artificial y el Perceptrón	11
2.2.2 Funciones de Activación	11
2.2.3 Regularización mediante Dropout	13
2.3 Redes Neuronales Convolucionales (CNN)	13
2.3.1 Convolución Unidimensional (Conv1D) para Series Temporales	13
2.4 Procesamiento de Series Temporales y Secuencias	14
2.4.1 Redes Neuronales Recurrentes (RNN)	14
2.4.2 Long Short-Term Memory (LSTM)	14
2.4.3 Gated Recurrent Units (GRU)	15
2.4.4 Redes Recurrentes Bidireccionales	15
2.4.5 Normalización por Capas (<i>Layer Normalization</i>)	15
2.5 Visión Computacional y Extracción de Landmarks con MediaPipe	16
2.6 Aumento de Datos (<i>Data Augmentation</i>)	16
2.7 Protocolos de Validación en Reconocimiento de Señas	17
2.7.1 Validación Intra-Signer con <i>StratifiedGroupKFold</i>	17
2.7.2 Validación Cross-Signer	17
2.8 Desarrollo Híbrido y Frameworks de Aplicaciones Móviles	18
2.8.1 React Native	18

2.8.2	Vision Camera y Frame Processors	18
2.9	Optimización y Despliegue en el Borde (Edge AI)	18
CAPÍTULO 3: PROPUESTA DE SOLUCIÓN		19
3.1	Creación del Dataset de LSCh en Video	19
3.1.1	Protocolo de Captura y Participantes	19
3.1.2	Validación y Curación de Datos	20
3.1.3	Conjunto Held-Out de Evaluación Cross-Signer	21
3.2	Diseño y Entrenamiento del Modelo Predictivo	22
3.2.1	Modelo de Referencia: Sistema Original de DialogApp	22
3.2.2	Ingeniería de Características	24
3.2.3	Estrategias de Augmentation para Robustez	25
3.2.4	Arquitectura de la Red Neuronal	26
3.3	Implementación y Optimización en Android	29
3.3.1	Infraestructura de Inferencia Nativa y Aceleración por Hardware	29
CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN		30
4.1	Metodología de Validación: Intra-Signer vs. Cross-Signer	30
4.2	Comparativa con el Modelo de Referencia	31
4.3	Comparativa Arquitectónica: Conv1D y Celda Recurrente	34
4.3.1	Ablación de la Capa Convolutiva	34
4.3.2	Ablación de la Celda Recurrente	35
4.4	Análisis de Ablación: Impacto de las Características y la Normalización	36
4.4.1	Impacto de las Características	36
4.4.2	Impacto de la Normalización	37
4.5	Eficacia del Data Augmentation	38
4.6	Análisis de Matriz de confusión	40
4.7	Precisión del Modelo en el Entorno Android	42
4.8	Latencia del Pipeline Completo en Dispositivo Android	43
CONCLUSIONES		44
ANEXOS		47
4.1	Anexo A: Alfabeto Dactilológico de la Lengua de Señas Chilena (LSCh)	47
4.2	Anexo B: Especificación del Modelo de Red Neuronal (Keras/Python)	47
4.3	Anexo C: Implementación del Motor de Inferencia (Kotlin/Android)	48
REFERENCIAS BIBLIOGRÁFICAS		50

ÍNDICE DE FIGURAS

1	Dificultad al utilizar servicios de salud para personas con discapacidad auditiva.	5
2	Letras dactilológicas para N y Ñ.	7
3	Diagrama de árbol del Problema	8
4	Funciones de activación ReLU, Sigmoide y Tanh.	12
5	Muestras representativas del dataset creado.	21
6	Arquitectura del modelo original de DialogApp.	23
7	Detalle del proceso optimizado de extracción de características para la entrada del modelo.	25
8	Arquitectura detallada del modelo V1 (Manos) para el reconocimiento de la dactilología LSCh.	28
9	Tradeoff exactitud vs. latencia en dispositivo real (Samsung Galaxy S23).	33
10	Curvas de convergencia del accuracy de entrenamiento y validación interna.	34
11	Comparativa de variantes convolucionales bajo validación cross-signer.	35
12	Comparativa de Accuracy y F1-Score entre distintas arquitecturas.	36
13	Impacto de la ingeniería de características en el Accuracy y F1-Score.	37
14	Comparativa de técnicas de normalización.	38
15	Desempeño del modelo V1 según estrategia de Data Augmentation (cross-signer).	39
16	Comparativa de estrategias de augmentation bajo validación cross-signer.	40
17	Matriz de confusión para las 27 clases de la LSCh.	41

ÍNDICE DE TABLAS

1	Comparativa de decisiones de diseño entre el modelo original de DialogApp y la propuesta V1.	29
2	Comparativa de desempeño intra-signer (split 80/20, seed=42).	31
3	Comparativa cross-signer (generalización a un signer no visto en entrenamiento), media $\pm$ desviación estándar sobre 20 semillas.	32
4	Comparativa de configuraciones recurrentes para V1.	35
5	Latencia del pipeline completo por dispositivo y modelo.	43

INTRODUCCIÓN

La comunicación es un pilar fundamental en la sociedad humana, permitiendo la transmisión de ideas, emociones y conocimientos. Sin embargo, para la comunidad sorda, las barreras comunicativas representan un obstáculo diario y significativo en su interacción con el resto de la sociedad, mayoritariamente oyente. En Chile, la Lengua de Señas Chilena (LSCh) es el medio natural y oficial de comunicación de las personas sordas. A pesar de los avances normativos, como la Ley 21.303 que reconoce y promueve el uso de la LSCh, la realidad cotidiana evidencia una profunda brecha de accesibilidad. La escasez de intérpretes calificados y el desconocimiento generalizado de la lengua de señas por parte de la población oyente limitan drásticamente la autonomía de las personas con discapacidad auditiva, afectando su acceso equitativo a servicios básicos como la salud, la educación y el mercado laboral.

En este contexto de necesidad urgente, la tecnología emerge como un puente potencial para mitigar esta problemática. Durante la última década, los avances en Inteligencia Artificial (IA) y, específicamente, en los campos de Visión Computacional (CV) y Aprendizaje Profundo (Deep Learning), han abierto nuevas posibilidades para el procesamiento y entendimiento automático del lenguaje visual. El reconocimiento automático de lengua de señas (Sign Language Recognition, SLR) se ha convertido en un área de investigación activa y prometedora, buscando desarrollar sistemas capaces de interpretar los gestos y movimientos característicos de las lenguas de señas y traducirlos a texto o voz en tiempo real.

No obstante, la gran mayoría de las investigaciones y desarrollos tecnológicos en esta área se han centrado en lenguas de señas con mayor cantidad de recursos de datos, como la Lengua de Señas Americana (ASL) o la Lengua de Señas China (CSL). Las lenguas de señas regionales, como la LSCh, presentan un desafío adicional debido a la escasez de corpus lingüísticos y datasets anotados de alta calidad y volumen, que son el insumo principal para el entrenamiento de modelos de Machine Learning. Además, muchas de las soluciones existentes están diseñadas para operar en entornos controlados, con hardware de alto rendimiento, lo que limita su aplicabilidad práctica en el día a día de un usuario común.

El presente trabajo de título surge como respuesta a estos desafíos técnicos y sociales. Su objetivo central es el diseño, implementación y validación comparativa de un algoritmo de reconocimiento de la Lengua de Señas Chilena en formato de video, optimizado para su ejecución en dispositivos móviles Android y evaluado rigurosamente frente al modelo de referencia integrado en DialogApp durante la 32ª Feria de Software. Este desarrollo se enmarca dentro del proyecto DialogApp, una aplicación concebida para la traducción bidireccional en tiempo real, cuyo propósito es facilitar la comunicación fluida y autónoma entre personas sordas y oyentes.

El alcance de este trabajo se centra en el reconocimiento automático del alfabeto dactilológico de la LSCh, considerando tanto las letras estáticas como dinámicas (como la 'G', 'J', 'Ñ',

'S', 'X', 'Z'). La inclusión de señas dinámicas representa una complejidad técnica sustancial, ya que requiere que el sistema no solo evalúe la configuración espacial de la mano en un instante dado, sino que también comprenda la trayectoria temporal del movimiento. Para abordar este desafío, se ha optado por un enfoque basado en el análisis de secuencias de puntos de referencia corporales (landmarks), extrayendo la estructura cinemática de las manos a través de la herramienta MediaPipe, en lugar de procesar los píxeles brutos de las imágenes. Esta estrategia metodológica reduce drásticamente la dimensionalidad de los datos, mejorando la eficiencia computacional y facilitando la invarianza frente a factores ambientales como la iluminación, el fondo o el color de piel del usuario.

El proceso metodológico llevado a cabo para la consecución de este objetivo se dividió en varias fases iterativas. En primer lugar, ante la inexistencia de conjuntos de datos de video públicos y adecuados para la LSCh, se diseñó y construyó un dataset propio. Este dataset fue meticulosamente recopilado, procesado y validado para asegurar que representara de manera fiel la diversidad de ejecuciones de las distintas letras del alfabeto dactilológico. Posteriormente, se diseñó la arquitectura del modelo de red neuronal. Se exploraron y evaluaron diferentes arquitecturas aptas para el procesamiento de series temporales, seleccionando una arquitectura híbrida Conv1D-GRU: dos capas convolucionales unidimensionales actúan como extractor jerárquico de rasgos locales sobre la secuencia temporal, seguidas de dos capas de unidades GRU bidireccionales que modelan la dinámica global del gesto. Esta combinación demostró empíricamente el mejor equilibrio entre precisión de generalización y eficiencia computacional en dispositivo móvil.

Una vez entrenado y validado el modelo en un entorno de escritorio, la fase final de la investigación se enfocó en el desafío del despliegue en el entorno móvil. La latencia es un factor crítico en una aplicación de traducción en tiempo real; una respuesta demorada degrada significativamente la experiencia de usuario y la fluidez de la conversación. Por tanto, se implementó una arquitectura de inferencia nativa en Android utilizando TensorFlow Lite y Frame Processors, eludiendo los cuellos de botella característicos de los frameworks de desarrollo multiplataforma. Este enfoque garantizó que el procesamiento de cada fotograma, la extracción de características y la ejecución del modelo predictivo se realizaran en milisegundos, permitiendo una experiencia de usuario natural e instantánea.

La estructura de esta memoria se organiza de la siguiente manera para guiar al lector a través de todo el proceso de investigación y desarrollo:

En el **Capítulo 1: Definición del Problema**, se detalla el contexto social y tecnológico que motiva esta investigación, se analiza la situación actual de la comunidad sorda en Chile y se formalizan los objetivos generales y específicos que guían el proyecto.

El **Capítulo 2: Marco Conceptual**, provee la fundamentación teórica necesaria para comprender las decisiones técnicas adoptadas. Se exploran conceptos de Machine Learning, Visión Computacional, redes neuronales recurrentes, la tecnología de extracción de landmarks y las consideraciones clave para el despliegue de modelos de Inteligencia Artificial en dispositivos

de borde (Edge AI).

En el **Capítulo 3: Propuesta de Solución**, se describe minuciosamente la metodología implementada. Este capítulo abarca desde el protocolo de creación del dataset propio de LSCh, el diseño de la etapa de ingeniería de características, la formulación de la arquitectura del modelo predictivo, hasta la estrategia de implementación nativa y optimización en la plataforma Android.

El **Capítulo 4: Validación de la Solución**, presenta los resultados experimentales bajo tres niveles de rigor creciente: validación intra-signer, validación cross-signer sobre un signer no visto en entrenamiento y medición en dispositivo Android real. Se incluyen estudios de ablación sobre la arquitectura (variantes Conv1D, elección de celda recurrente), la ingeniería de características y las estrategias de augmentation, así como la comparativa estadística frente al modelo original de DialogApp (test de Wilcoxon, 20 semillas) y la medición de latencia de pipeline completo en Samsung Galaxy S20 FE y S23.

Finalmente, el **Capítulo 5: Conclusiones**, sintetiza los hallazgos más relevantes del trabajo. Se reflexiona sobre el cumplimiento de los objetivos planteados, se discuten las limitaciones encontradas durante el desarrollo de la investigación y se proponen diversas líneas de trabajo futuro para continuar mejorando y expandiendo las capacidades del sistema de reconocimiento de la Lengua de Señas Chilena.

CAPÍTULO 1

DEFINICIÓN DEL PROBLEMA

1.1. Contexto

En la sociedad contemporánea, la inclusión y la accesibilidad de las personas con discapacidad se han consolidado como pilares fundamentales para el desarrollo de una comunidad equitativa. Es en este ámbito donde la ingeniería y la tecnología deben desempeñar un rol protagónico, actuando como herramientas facilitadoras para mejorar la calidad de vida de las personas y reducir las brechas históricas que limitan su plena participación social.

En el caso particular de Chile, la situación de la comunidad sorda presenta desafíos significativos. De acuerdo con el III Estudio Nacional de la Discapacidad (ENDISC 2022) [Servicio Nacional de la Discapacidad (SENADIS), 2022], existen 3.291.602 personas con discapacidad en el país, lo que representa el 17,6% de la población. Dentro de este grupo, se estima que un 10% posee dificultades de tipo auditivo, y un 9,4% de los niños, niñas y adolescentes con discapacidad presentan sordera o dificultades en el habla. Para abordar esta realidad desde una perspectiva legal, se promulgó la Ley 21.303 [MINISTERIO DE DESARROLLO SOCIAL Y FAMILIA, 2021], la cual reconoce oficialmente la Lengua de Señas Chilena (LSCh) como el medio de comunicación natural de la comunidad sorda. Esta normativa tiene como propósito fundamental promover el uso de la LSCh en ámbitos críticos como la educación, la salud y el mercado laboral, buscando garantizar el derecho a la información y a la comunicación de las personas sordas.

A pesar de la existencia de este marco legal, la implementación efectiva de la inclusión lingüística sigue siendo insuficiente en la práctica cotidiana. La principal barrera radica en el desconocimiento generalizado de la LSCh por parte de la población oyente, lo que genera una dependencia crítica de intermediarios. Tradicionalmente, la comunicación entre una persona sorda y una oyente se ha resuelto mediante la contratación de intérpretes. No obstante, esta solución enfrenta graves limitaciones logísticas y económicas. Según Gustavo Vergara, presidente de la Asociación de Sordos Chilenos (ASOCH), existen apenas unos 400 intérpretes certificados a nivel nacional [La Tercera, 2023], una cifra que resulta desproporcionadamente baja en comparación con la población que requiere sus servicios.

Esta escasez de profesionales obliga a que, en muchas ocasiones, sean los familiares de la persona sorda quienes asuman el rol de intérpretes, lo que conlleva problemas de privacidad y autonomía. Esta problemática es especialmente crítica en el sector salud, donde la comunicación precisa es vital. De acuerdo con la Encuesta Nacional de Discapacidad y Dependencia (ENDIDE) [Observatorio Social, 2022], solo un 10% de las personas con discapacidad auditiva reporta no haber tenido dificultades al utilizar servicios de salud, mientras que cerca del 30% manifiesta haber experimentado problemas severos o extremos. La falta de una comunica-

ción fluida no solo vulnera la privacidad del paciente, sino que también puede comprometer la calidad de la atención médica recibida, como refleja la Figura 1.

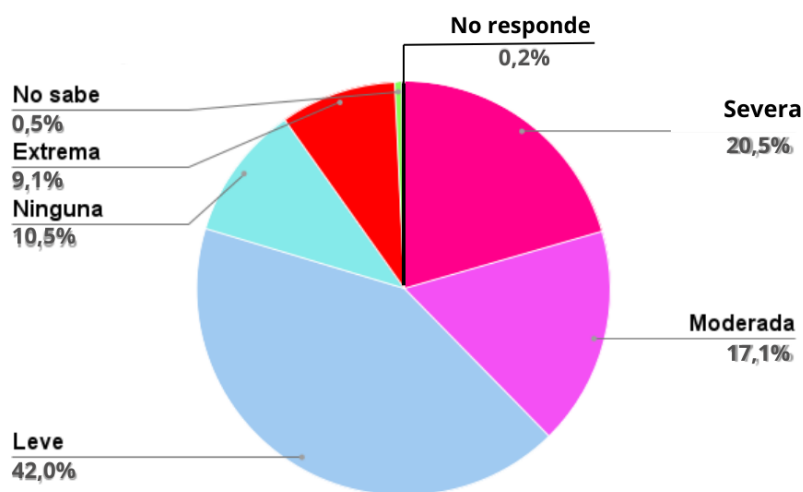


Figura 1: Dificultad al utilizar servicios de salud para personas con discapacidad auditiva.

Fuente: Base de datos ENDIDE 2022 adultos, cuidadores(as) y NNA

[Observatorio Social, 2022]. Gráfico de elaboración propia. Muestra: 20859.

1.2. Motivación

La motivación principal de esta memoria surge del desarrollo de DialogApp, un proyecto tecnológico concebido en el marco de la 32ª Feria de Software de la Universidad Técnica Federico Santa María. DialogApp es una aplicación móvil para el sistema operativo Android diseñada para facilitar la comunicación bidireccional en tiempo real entre personas sordas y oyentes a través de la traducción de la LSCh.

Un componente esencial y crítico para el éxito de DialogApp es el desarrollo de un módulo de reconocimiento automático de señas. En particular, este trabajo se centra en el diseño de un algoritmo capaz de interpretar el abecedario dactilológico de la LSCh a partir de secuencias de video, abarcando la totalidad de sus caracteres: [A, B, C, D, E, F, G, H, I, J, K, L, M, N, Ñ, O, P, Q, R, S, T, U, V, W, X, Y, Z]. La creación de un dataset propio y la adaptación de modelos de Aprendizaje Automático (Machine Learning) para su ejecución eficiente en dispositivos móviles representan los desafíos técnicos que impulsan este trabajo.

La posibilidad de integrar una herramienta de traducción directamente en un smartphone permite democratizar el acceso a la tecnología de asistencia, otorgando a las personas sordas una mayor independencia en sus gestiones diarias sin depender exclusivamente de la disponibilidad presencial de un intérprete humano.

1.3. Definición del problema

El reconocimiento automático de lenguas de señas (Sign Language Recognition - SLR) constituye uno de los problemas más complejos en el campo de la Visión Computacional y el Procesamiento de Lenguaje Natural. A diferencia de otros sistemas de reconocimiento biométrico o gestual simple, las lenguas de señas involucran una multiplicidad de canales de información simultáneos, incluyendo la posición y configuración de las manos, la orientación de las palmas, el movimiento de los brazos y las expresiones faciales.

La variabilidad es otro factor que añade complejidad al problema. Las señas pueden variar significativamente dependiendo de la fisonomía del usuario, la velocidad de ejecución y las condiciones ambientales como la iluminación o el fondo de la imagen. Por lo tanto, el sistema debe ser capaz de extraer características robustas que sean invariantes a estos factores externos, enfocándose estrictamente en la semántica del gesto.

En el contexto chileno, el desarrollo de soluciones tecnológicas para la LSCh se enfrenta a un obstáculo adicional: la falta de recursos de datos abiertos. Si bien existen investigaciones globales avanzadas para lenguas de señas como la Americana (ASL), estos modelos no son directamente aplicables a la LSCh debido a las diferencias léxicas y gramaticales propias de cada país. Investigaciones nacionales recientes han explorado el uso de arquitecturas basadas en Transformers para el reconocimiento aislado [Yáñez y Aguilera, 2025], pero el enfoque en dispositivos móviles sigue siendo una brecha tecnológica crítica. En el ámbito local, trabajos previos como el de Luis Benítez y Ricardo Catril [Benítez Andrade y Catril Carrillo, 2022] han logrado avances significativos en el reconocimiento del alfabeto LSCh utilizando imágenes estáticas, demostrando la viabilidad técnica del uso de dispositivos móviles para esta tarea.

Sin embargo, el abecedario de la LSCh presenta un desafío que el procesamiento de imágenes estáticas no puede resolver por sí solo: la existencia de señas dinámicas. Mientras que señas estáticas como la 'A' o la 'L' se definen por una postura fija de la mano, otras letras requieren de un movimiento específico para ser identificadas. Por ejemplo, las letras 'N' y 'Ñ' comparten la misma configuración manual, diferenciándose únicamente por el movimiento lateral característico de la 'Ñ' (Figura 2). Del mismo modo, letras como la 'G', 'J', 'S', 'X' y 'Z' poseen componentes dinámicos esenciales. Un sistema que solo analice fotogramas aislados sería inherentemente incapaz de distinguir entre estas clases, lo que hace imperativo el uso de algoritmos que procesen video y capturen la dimensión temporal del movimiento.

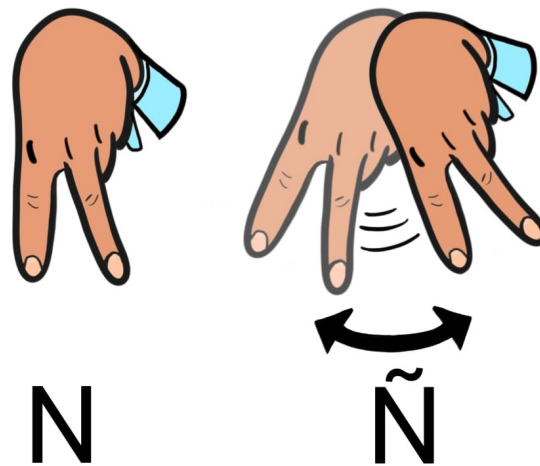


Figura 2: Letras dactilológicas para N y Ñ.
Fuente: Elaboración propia.

Dada esta necesidad, esta memoria aborda la implementación de un sistema que no solo reconozca las formas de las manos, sino también su trayectoria en el tiempo, cubriendo el alfabeto completo de la LSCh de la A a la Z. El enfoque en dispositivos Android responde a la necesidad de crear una solución portátil y accesible que pueda ser utilizada en cualquier entorno, desde una consulta médica hasta un trámite administrativo.

Finalmente, tras analizar la problemática, sus causas y su impacto en la comunidad sorda, se presenta el diagrama de árbol del problema (Figura 3).

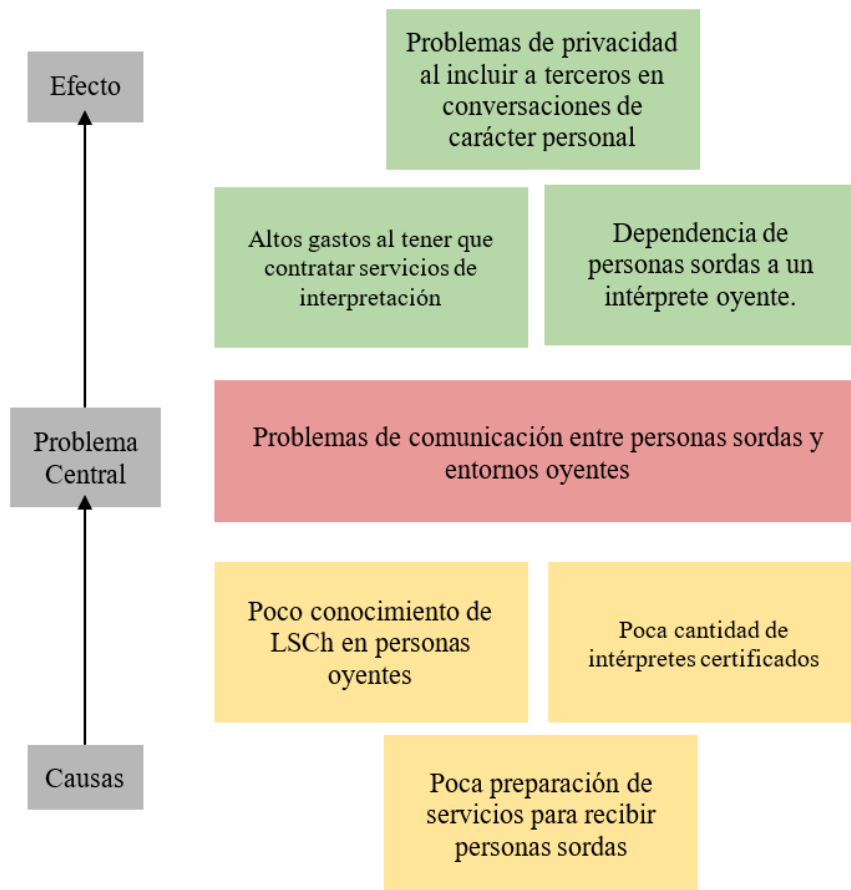


Figura 3: Diagrama de árbol del Problema
Fuente: Elaboración propia.

Así, la implementación de un sistema de reconocimiento de LSCh en dispositivos Android responde a la necesidad urgente de facilitar la comunicación entre personas sordas y oyentes en Chile. Este proyecto empleará un dataset propio y algoritmos adaptados a Android para el reconocimiento en tiempo real, maximizando la accesibilidad de la tecnología y promoviendo la inclusión.

1.4. Objetivos

1.4.1. Objetivo General

Proponer y validar rigurosamente una arquitectura mejorada de reconocimiento del alfabeto dactilológico de la LSCh para dispositivos Android, superando en precisión y eficiencia al

modelo original desarrollado en la Feria de Software.

1.4.2. Objetivos Específicos

- Diseñar una arquitectura de reconocimiento optimizada, mediante ingeniería de características y experimentación arquitectónica, que supere al modelo original bajo condiciones de evaluación exigentes.
- Aplicar una metodología de validación estadística robusta que permita comparar objetivamente el desempeño de ambos modelos y sustentar la significancia de las mejoras observadas.
- Optimizar y desplegar la arquitectura propuesta en dispositivos Android, evaluando su desempeño en precisión, latencia y capacidad de operar en tiempo real frente al modelo original.

CAPÍTULO 2

MARCO CONCEPTUAL

El presente capítulo expone las bases conceptuales, teóricas y tecnológicas que sustentan la presente investigación. Se realiza un recorrido desde los fundamentos del aprendizaje automático hasta las arquitecturas de aprendizaje profundo especializadas en el procesamiento de series temporales, concluyendo con las estrategias de optimización necesarias para el despliegue de modelos de inteligencia artificial en dispositivos de borde.

2.1. Aprendizaje Automático (Machine Learning)

El Aprendizaje Automático es una rama de la Inteligencia Artificial que se centra en el desarrollo de algoritmos y modelos estadísticos que permiten a los sistemas computacionales identificar patrones complejos y tomar decisiones basadas en datos, sin necesidad de ser programados explícitamente para cada tarea específica. Según la definición clásica de Tom Mitchell (1997) [Mitchell, 1997], un programa de computadora aprende de una experiencia E con respecto a alguna clase de tareas T y una medida de desempeño P , si su desempeño en las tareas de T , medido por P , mejora progresivamente con la experiencia E .

En el contexto de esta memoria, el aprendizaje automático permite que el sistema “entienda” la relación geométrica entre los puntos de la mano y el significado lingüístico que estos representan en la LSCh, generalizando este conocimiento a partir de ejemplos previos.

2.1.1. Aprendizaje Supervisado

Dentro del espectro del aprendizaje automático, el aprendizaje supervisado es la modalidad más utilizada para tareas de clasificación. En este enfoque, el modelo se entrena utilizando un conjunto de datos previamente etiquetados por un experto humano. Cada instancia del dataset de entrenamiento consiste en un par entrada-salida, donde la entrada son los datos brutos (o características procesadas) y la salida es la etiqueta o clase correcta.

El objetivo del proceso de entrenamiento es aprender una función de mapeo $\hat{y} = f(x)$ que minimice una función de pérdida $\mathcal{L}(y, f(x))$, es decir, la discrepancia entre las predicciones del modelo y las etiquetas reales. En el desarrollo de DialogApp, se emplea aprendizaje supervisado al alimentar el modelo con secuencias de video de señas donde se conoce de antemano la letra del alfabeto dactilológico correspondiente (de la A a la Z).

2.2. Redes Neuronales Artificiales (RNA)

Las Redes Neuronales Artificiales son modelos computacionales inspirados en la estructura y el funcionamiento biológico del cerebro humano. Estas redes están compuestas por una gran cantidad de unidades de procesamiento interconectadas y organizadas en capas: una capa de entrada, una o más capas ocultas y una capa de salida.

2.2.1. La Neurona Artificial y el Perceptrón

La unidad básica de procesamiento es la neurona artificial. Esta recibe múltiples señales de entrada (x_i), cada una ponderada por un peso sináptico (w_i). Los pesos representan la importancia o relevancia de cada entrada para la neurona. La suma ponderada de las entradas, junto con un término de sesgo o *bias* (b), se somete a una función de activación (f) para generar la salida de la neurona. Matemáticamente, este proceso se representa como:

$$y = f \left(\sum_{i=1}^n w_i x_i + b \right) \quad (1)$$

El aprendizaje en una red neuronal consiste en el ajuste iterativo de estos pesos y sesgos mediante algoritmos de optimización (como el Descenso del Gradiente) y el método de Retropropagación (*Backpropagation*), con el fin de reducir el error global de la red.

2.2.2. Funciones de Activación

Las funciones de activación son componentes críticos que introducen no linealidad en el modelo, permitiendo que la red neuronal aprenda y represente relaciones complejas que no podrían ser capturadas por modelos lineales simples.

- **ReLU (Rectified Linear Unit):** Definida matemáticamente como:

$$f(x) = \max(0, x) \quad (2)$$

Es la función más utilizada en las capas ocultas de las redes modernas debido a que ayuda a mitigar el problema del desvanecimiento del gradiente y es computacionalmente muy eficiente.

- **Sigmoide:** Definida como:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (3)$$

Comprime cualquier valor real al rango $(0, 1)$, lo que la hace útil para representar probabilidades o compuertas de activación binaria. En las celdas LSTM y GRU se emplea en las compuertas de olvido, entrada y salida, regulando qué proporción de la información fluye a través de la celda. Su principal limitación es la saturación: para valores de x grandes en magnitud, el gradiente se aproxima a cero, dificultando el aprendizaje en redes profundas.

- **Tanh (Tangente Hiperbólica):** Definida como:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (4)$$

Es una versión reescalada de la Sigmoide que comprime la entrada al rango $(-1, 1)$ y está centrada en cero, lo que favorece una convergencia más estable durante el entrenamiento. En las celdas LSTM y GRU se utiliza para generar el contenido candidato del estado interno de la celda, mientras que las compuertas (basadas en Sigmoide) determinan cuánto de ese contenido se conserva o descarta.

- **Softmax:** Se aplica exclusivamente en la capa de salida para problemas de clasificación multiclase. Transforma el vector de salida z en una distribución de probabilidad:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \quad (5)$$

donde cada valor representa la confianza del modelo para una clase específica i .

La Figura 4 ilustra comparativamente las funciones ReLU, Sigmoide y Tanh; las dos últimas actúan como compuertas internas de las celdas LSTM y GRU.

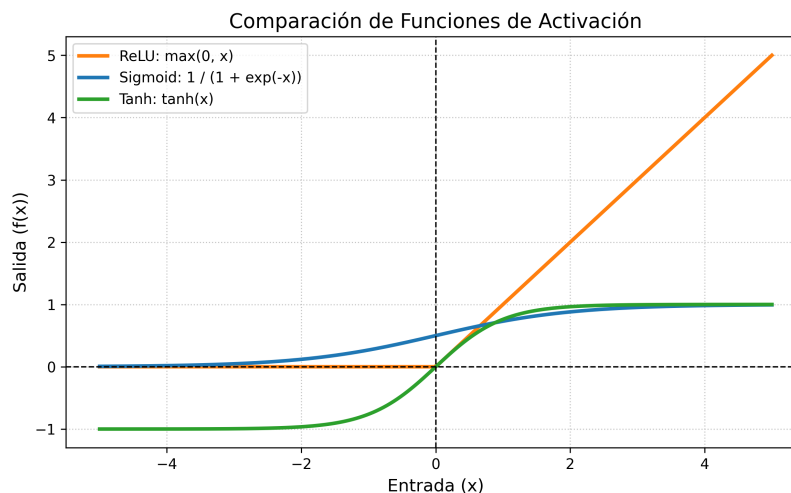


Figura 4: Funciones de activación ReLU, Sigmoide y Tanh. ReLU se utiliza en las capas ocultas del modelo; Sigmoide y Tanh actúan como compuertas internas de las celdas LSTM y GRU. Fuente: Elaboración propia.

2.2.3. Regularización mediante Dropout

El sobreajuste (*overfitting*) ocurre cuando un modelo memoriza los patrones específicos del conjunto de entrenamiento sin generalizar a datos nuevos. Para mitigarlo, Srivastava et al. [Srivastava et al., 2014] propusieron el *Dropout*: durante el entrenamiento, una fracción p de las neuronas de una capa se desactiva aleatoriamente en cada actualización de pesos, forzando a la red a aprender representaciones distribuidas y redundantes. En inferencia, todas las neuronas permanecen activas y sus salidas se escalan por $(1 - p)$ para mantener la magnitud esperada. En el modelo propuesto se aplica Dropout con $p = 0,2$ tras las capas convolucionales y con $p = 0,3$ entre las capas densas finales.

2.3. Redes Neuronales Convolucionales (CNN)

Las Redes Neuronales Convolucionales están diseñadas específicamente para procesar datos con una estructura de cuadrícula, como las imágenes. Su arquitectura se basa en la aplicación de filtros o núcleos (*kernels*) que realizan operaciones de convolución sobre la entrada para detectar patrones jerárquicos.

En las primeras capas, las CNN detectan características simples como bordes y orientaciones. A medida que se profundiza en la red, estas características se combinan para reconocer patrones más abstractos como formas, texturas y finalmente objetos complejos (como una mano o un rostro). En esta investigación, las CNN tienen presencia tanto directa como indirecta. De forma directa, el modelo propuesto emplea capas convolucionales unidimensionales (Conv1D) como extractor de rasgos temporales, tal como se detalla a continuación. De forma indirecta, el framework MediaPipe utiliza arquitecturas convolucionales altamente optimizadas (como BlazePalm) para realizar la detección inicial de la mano en el flujo de video con una latencia mínima.

2.3.1. Convolución Unidimensional (Conv1D) para Series Temporales

Mientras que las capas Conv2D operan sobre cuadrículas bidimensionales de píxeles, las capas Conv1D aplican filtros deslizantes a lo largo de un único eje temporal. Dado un tensor de entrada de dimensión (T, F) —donde T es el número de pasos temporales y F el número de características por paso—, un filtro de tamaño de kernel k opera sobre ventanas de k fotogramas consecutivos para detectar patrones locales en la secuencia. El apilamiento de dos capas Conv1D permite aprender una jerarquía de rasgos: la primera capa detecta micro-gestos (cambios de posición entre fotogramas adyacentes), mientras que la segunda combina estos rasgos para capturar transiciones de mayor duración. En el modelo propuesto se emplean dos capas Conv1D de 64 filtros con kernel de tamaño 5 y *padding same*, configuración que preserva la dimensión temporal para su posterior procesamiento por las capas

recurrentes.

2.4. Procesamiento de Series Temporales y Secuencias

A diferencia de la clasificación de imágenes estáticas, donde el análisis de un solo fotograma es suficiente, el reconocimiento de gestos dinámicos en lengua de señas requiere el modelado de la dimensión temporal. El significado de una letra como la 'Z' no reside en una sola posición de la mano, sino en la trayectoria que esta describe a lo largo de una secuencia de fotogramas.

2.4.1. Redes Neuronales Recurrentes (RNN)

Las RNN son una familia de redes diseñadas para trabajar con datos secuenciales. A diferencia de las redes *feed-forward* tradicionales, las RNN poseen conexiones cíclicas que les permiten mantener un “estado oculto” o memoria interna (h_t). Este estado se actualiza en cada paso de tiempo, permitiendo que la información de eventos pasados influya en el procesamiento de la entrada actual.

Sin embargo, las RNN estándar presentan una limitación teórica importante conocida como el problema del desvanecimiento del gradiente (*Vanishing Gradient*), documentado formalmente por Bengio et al. (1994) [Bengio et al., 1994]. Esto ocurre cuando las señales de error se vuelven extremadamente pequeñas al propagarse hacia atrás en el tiempo, impidiendo que la red aprenda dependencias a largo plazo entre elementos distantes de la secuencia.

2.4.2. Long Short-Term Memory (LSTM)

Las LSTM, introducidas por Hochreiter y Schmidhuber [Hochreiter y Schmidhuber, 1997], resuelven el desvanecimiento del gradiente mediante un estado de celda C_t protegido por tres compuertas aprendibles: la **compuerta de olvido** f_t , que decide qué información descartar del estado anterior; la **compuerta de entrada** i_t , que regula qué información nueva incorporar al estado de celda; y la **compuerta de salida** o_t , que controla qué parte del estado de celda se expone como salida h_t . Internamente, estas compuertas utilizan la función Sigmoide —que acota su salida a $(0, 1)$ para actuar como válvula de paso— y la función Tanh —que acota el estado de celda a $(-1, 1)$ para regular su magnitud—. Gracias a este mecanismo el gradiente puede fluir sin atenuarse a través de muchos pasos temporales, permitiendo aprender dependencias a largo plazo. El modelo original de DialogApp emplea cuatro capas LSTM apiladas (128, 64, 64 y 64 unidades) como codificador temporal.

2.4.3. Gated Recurrent Units (GRU)

Para solventar las limitaciones de las RNN básicas, se desarrollaron arquitecturas basadas en mecanismos de compuertas, siendo las más destacadas las Long Short-Term Memory (LSTM) y las Gated Recurrent Units (GRU).

Las GRU, introducidas por Cho et al. (2014) [Cho et al., 2014], simplifican la arquitectura de las LSTM al combinar las compuertas de olvido y entrada en una única “compuerta de actualización”. Esta reducción en la complejidad estructural se traduce en un menor número de parámetros (entre un 30 % y 40 % menos que una LSTM estándar), lo que resulta en un entrenamiento más rápido y, crucialmente para este proyecto, en una inferencia mucho más eficiente en dispositivos móviles con recursos limitados, tal como se demuestra en estudios comparativos de reconocimiento de señas [2023].

2.4.4. Redes Recurrentes Bidireccionales

Las arquitecturas recurrentes estándar procesan la secuencia en un único sentido temporal, por lo que el estado oculto h_t solo puede incorporar contexto pasado. Las redes bidireccionales [Schuster y Paliwal, 1997] extienden este esquema ejecutando dos instancias del modelo recurrente en paralelo: una en sentido directo (fotograma $1 \rightarrow T$) y otra en sentido inverso (fotograma $T \rightarrow 1$). Las representaciones de ambas pasadas se concatenan en cada paso temporal, permitiendo que la red acceda simultáneamente al contexto pasado y futuro. Esta capacidad es especialmente relevante para señas dinámicas cuyo inicio y fin comparten la misma configuración manual —como la ‘J’ y la ‘Z’—, ya que la red puede resolver la ambigüedad observando la trayectoria completa del movimiento antes de emitir su predicción. En el modelo propuesto se apilan dos capas GRU Bidireccionales de 64 unidades cada una.

2.4.5. Normalización por Capas (Layer Normalization)

Las técnicas de normalización estabilizan la distribución de las activaciones durante el entrenamiento, acelerando la convergencia. La Normalización por Lotes (*Batch Normalization*) [Ioffe y Szegedy, 2015] calcula estadísticas de normalización a lo largo de la dimensión de lote. Sin embargo, cuando las secuencias se rellenan con ceros (*zero-padding*) para homogeneizar el tamaño del lote, estos ceros contaminan las estadísticas de media y varianza, distorsionando la normalización. La *Layer Normalization* [Ba et al., 2016] resuelve esto calculando las estadísticas a lo largo de la dimensión de características de cada instancia de forma completamente independiente del lote. Esto la hace especialmente adecuada para secuencias de longitud variable con padding. En el modelo propuesto se aplica tras las capas Conv1D; los experimentos de ablación confirman su superioridad sobre Batch Normalization para esta tarea.

2.5. Visión Computacional y Extracción de Landmarks con MediaPipe

La visión computacional moderna ha pasado de procesar píxeles directamente a utilizar representaciones intermedias más robustas. Para el reconocimiento de señas, el uso de “esqueletos” o puntos de referencia (*landmarks*) es una técnica superior al procesamiento de imágenes en bruto.

MediaPipe Hands es un framework de código abierto desarrollado por Google que permite la percepción de manos y dedos en tiempo real [Zhang *et al.*, 2020]. Utiliza un pipeline que consta de dos modelos trabajando en conjunto:

1. **Detector de Palma (BlazePalm):** Es un modelo de detección de objetos optimizado para móviles que localiza la presencia de manos en la imagen y define una región de interés (RoI).
2. **Modelo de Landmarks:** Es una red neuronal que opera sobre la RoI para predecir las coordenadas 3D de 21 puntos clave de la mano (nudillos, puntas de los dedos, muñeca).

Este enfoque ofrece una gran ventaja: al trabajar con coordenadas geométricas en lugar de píxeles, el sistema se vuelve intrínsecamente más resistente a variaciones en la iluminación, el color de piel o fondos complejos, ya que el esqueleto de la mano mantiene su estructura independientemente de estos factores.

2.6. Aumento de Datos (*Data Augmentation*)

El aumento de datos amplía artificialmente el conjunto de entrenamiento mediante transformaciones aplicadas a las muestras existentes, exponiendo al modelo a mayor variabilidad de ejecución sin necesidad de recolectar nuevos datos. En este trabajo las transformaciones se aplican directamente sobre las coordenadas tridimensionales de los landmarks —en lugar de sobre píxeles—, lo que preserva las propiedades geométricas de los gestos. Se emplean cuatro estrategias:

- **Rotaciones espaciales ($\pm 15^\circ$):** Aplicación de una matriz de rotación aleatoria sobre los ejes X e Y a toda la secuencia de landmarks, simulando las inclinaciones naturales del smartphone durante el uso cotidiano.
- **Inversión lateral (*Flipping*):** Inversión del signo de la coordenada X de todos los landmarks, generando la ejecución especular de cada seña. Permite que el modelo reconozca señas ejecutadas con ambas manos de manera ambidiestra.

- **Ruido gaussiano:** Adición de perturbaciones aleatorias con desviación estándar $\sigma = 0,005$ sobre las coordenadas, simulando imprecisiones menores en la detección de MediaPipe causadas por desenfoque de movimiento o iluminación subóptima.
- **Mixup intra-clase:** Muestras sintéticas generadas interpolando linealmente pares de secuencias de la misma clase con factor $\lambda \sim \text{Beta}(0,2, 0,2)$: $\tilde{x} = \lambda x_i + (1 - \lambda)x_j$ [Zhang *et al.*, 2018]. Esta técnica suaviza las fronteras de decisión del clasificador, penalizando predicciones excesivamente confiadas. Los experimentos de ablación muestran que el beneficio del mixup emerge en sinergia con las transformaciones geométricas, no de forma aislada.

2.7. Protocolos de Validación en Reconocimiento de Señas

La elección del protocolo de validación determina si las métricas reportadas reflejan la capacidad real de generalización del sistema o una sobreestimación derivada de la configuración experimental.

2.7.1. Validación Intra-Signer con *StratifiedGroupKFold*

En el protocolo intra-signer, los conjuntos de entrenamiento y prueba provienen de los mismos participantes. La *StratifiedGroupKFold* es una variante de la validación cruzada que garantiza, adicionalmente, que ninguna variante aumentada de un mismo video original aparezca en particiones distintas, evitando la filtración de información (*data leakage*) entre entrenamiento y prueba cuando se aplican técnicas de augmentation. Las métricas bajo este protocolo miden la robustez del modelo ante nuevas ejecuciones de usuarios ya conocidos, no su generalización a personas nuevas.

2.7.2. Validación Cross-Signer

En el protocolo cross-signer, el conjunto de prueba está compuesto íntegramente por grabaciones de un signer no presente en el entrenamiento. Este es el escenario más exigente y el único que estima la verdadera capacidad de generalización inter-persona del sistema. La brecha entre el desempeño intra-signer y cross-signer es un fenómeno documentado en reconocimiento de señas [Pigou *et al.*, 2016]: el estilo individual de ejecución introduce variabilidad significativa entre personas, y el modelo debe aprender a ignorar estas particularidades. En este trabajo la métrica cross-signer se adopta como la estimación honesta del desempeño en producción.

2.8. Desarrollo Híbrido y Frameworks de Aplicaciones Móviles

El despliegue de soluciones de inteligencia artificial en dispositivos móviles requiere una integración profunda con el sistema operativo y el hardware de la cámara.

2.8.1. React Native

DialogApp se construye utilizando React Native [Meta Platforms, Inc., 2015], un framework de código abierto creado por Meta Platforms que permite desarrollar aplicaciones móviles nativas utilizando JavaScript y React. La principal ventaja de este enfoque es el desarrollo multiplataforma con un único código base. Sin embargo, para tareas de alto rendimiento como el procesamiento de video frame a frame, el puente de JavaScript (*JS Bridge*) puede introducir una latencia inaceptable.

2.8.2. Vision Camera y Frame Processors

Para superar las limitaciones del JS Bridge, se utiliza la librería *Vision Camera*. Esta herramienta introduce el concepto de *Frame Processors*, que son funciones escritas en lenguajes nativos (como Kotlin o C++) que se ejecutan directamente en el hilo de la cámara. Esto permite que el procesamiento de cada fotograma, incluyendo la extracción de landmarks y la inferencia de la red neuronal, ocurra de manera inmediata y sin bloqueos en la interfaz de usuario.

2.9. Optimización y Despliegue en el Borde (Edge AI)

El concepto de Edge AI se refiere a la ejecución de algoritmos de inteligencia artificial directamente en el dispositivo local (en este caso, un smartphone Android), en lugar de enviarlos a un servidor en la nube. Esto ofrece ventajas críticas en términos de privacidad de datos, reducción de costos de infraestructura y, sobre todo, una latencia mínima indispensable para una traducción fluida.

Para lograr esto, se utiliza TensorFlow Lite (TFLite) [2016], un framework de inferencia optimizado para hardware móvil. TFLite permite convertir modelos complejos a un formato binario ligero y emplear técnicas como la cuantización (reducir la precisión de los pesos de 32 bits a 16 u 8 bits) y el uso de delegados de hardware (GPU o DSP) para acelerar los cálculos matemáticos, permitiendo que modelos de Deep Learning se ejecuten en tiempo real sin agotar excesivamente la batería del dispositivo.

CAPÍTULO 3

PROPUESTA DE SOLUCIÓN

La solución propuesta para el reconocimiento de la Lengua de Señas Chilena (LSCh) se fundamenta en un paradigma de procesamiento híbrido que combina la extracción de características geométricas de alto nivel con el modelado temporal mediante redes neuronales recurrentes. El sistema evita el procesamiento directo de píxeles para centrarse en la cinemática de las manos, lo que garantiza eficiencia y robustez en dispositivos móviles.

El desarrollo de la solución integral para DialogApp se estructuró en tres fases metodológicas fundamentales, las cuales se detallan a continuación:

1. **Fase de Datos:** Creación, procesamiento y validación de un dataset de LSCh en formato de video.
2. **Fase de Modelado:** Diseño, entrenamiento y evaluación de una arquitectura de Deep Learning optimizada para secuencias temporales.
3. **Fase de Despliegue:** Implementación, optimización e integración del motor de inferencia en el entorno Android.

3.1. Creación del Dataset de LSCh en Video

La piedra angular de cualquier sistema de Aprendizaje Automático es la calidad y representatividad de sus datos. Al inicio de esta investigación, se constató la inexistencia de un corpus de video público y estandarizado para la LSCh. Si bien se contaba con el trabajo previo de Benítez y Catril [Benítez Andrade y Catril Carrillo, 2022], este consistía en un dataset de imágenes estáticas, insuficiente para el modelado de señas dinámicas.

3.1.1. Protocolo de Captura y Participantes

Para solventar esta carencia, se diseñó un protocolo de captura controlado pero representativo del uso real de un smartphone. El objetivo se fijó en la recolección de las 27 letras del alfabeto dactilológico de la LSCh: [A, B, C, D, E, F, G, H, I, J, K, L, M, N, Ñ, O, P, Q, R, S, T, U, V, W, X, Y, Z].

Se contó con la participación de siete estudiantes voluntarios de la Universidad Técnica Federico Santa María, quienes realizaron las señas siguiendo las definiciones del diccionario oficial de LSCh de Juego Educa [Juego Educa, 2025]. Las capturas se realizaron utilizando la

cámara frontal de dos dispositivos de gama media y alta (Samsung Galaxy A54 y Samsung Galaxy S20FE) para asegurar variabilidad en los sensores ópticos y resoluciones. Los videos se grabaron en formato vertical (9:16), a una resolución de 1080p y una tasa constante de 30 fotogramas por segundo (FPS).

Cada participante realizó múltiples repeticiones de cada letra, introduciendo variaciones naturales en la velocidad de ejecución, la inclinación de la mano y ligeros desplazamientos respecto al centro de la cámara. En total, se recolectaron 50 videos por cada seña, consolidando un dataset inicial de 1.350 clips de video, cada uno con una duración exacta de 1 segundo. Esta recolección se realizó durante el desarrollo del proyecto original de feria de software que dio origen a DialogApp, por lo que la distribución de videos entre los siete participantes no es uniforme.

3.1.2. Validación y Curación de Datos

No todos los videos grabados fueron aptos para el entrenamiento. Se implementó un proceso de curación semiautomatizado utilizando el modelo de extracción de puntos de referencia de MediaPipe. Cada video fue procesado para verificar que:

- La mano estuviera visible y fuera detectada correctamente en al menos el 90 % de los fotogramas.
- No existieran oclusiones severas o falsos positivos (detección de rostros u otros objetos como manos).
- La ejecución de la seña fuera fiel a la norma lingüística.

Tras una inspección manual de los resultados de la detección, se descartaron los videos deficientes, asegurando que el modelo se entrenara únicamente con datos de alta fidelidad geométrica. La Figura 5 muestra muestras representativas del dataset resultante.



Figura 5: Muestras representativas del dataset creado. (A) Fotograma de la ejecución de la seña 'D'. (B) Fotograma de la seña 'F'.

3.1.3. Conjunto Held-Out de Evaluación Cross-Signer

Para estimar el desempeño real del sistema frente a usuarios no vistos durante el entrenamiento, se construyó específicamente para esta memoria un conjunto de evaluación independiente, recolectado con posterioridad al dataset de entrenamiento, con una persona externa al grupo de participantes original. Este signer adicional no es hablante de LSCh; ejecutó cada seña siguiendo una referencia visual del diccionario oficial, replicando el escenario de un usuario oyente que aprende a utilizar la aplicación sin formación previa en lengua de señas.

Las condiciones de captura fueron deliberadamente distintas a las del dataset de entrenamiento: se utilizó un dispositivo diferente y un entorno físico distinto, introduciendo variabilidad en las características ópticas de la cámara y en las condiciones de iluminación. Esta elección refuerza la validez externa de la evaluación, al desacoplar el desempeño del modelo de las particularidades del hardware y el entorno utilizados durante la recolección de datos de entrenamiento.

El conjunto resultante comprende **135 clips** ($27 \text{ clases} \times 5 \text{ videos por clase}$), con distribución uniforme entre clases, grabados bajo el mismo protocolo temporal (1 segundo a 30 FPS). Ningún video de este conjunto ni ninguna de sus variantes aumentadas se utilizó en ninguna fase del entrenamiento. Este conjunto held-out es la única fuente de la métrica cross-signer reportada a lo largo de este trabajo, y constituye la estimación honesta del desempeño del sistema en producción.

3.2. Diseño y Entrenamiento del Modelo Predictivo

El núcleo del sistema de reconocimiento es un modelo de Deep Learning capaz de clasificar secuencias temporales de landmarks. Para justificar cada decisión de diseño, esta sección presenta primero el **modelo de referencia** (el sistema original de DialogApp) y sus limitaciones técnicas, y luego el **diseño propuesto (V1)**, que introduce cambios en tres ejes: la ingeniería de características (Sección 3.2.2), las estrategias de augmentation (Sección 3.2.3) y la arquitectura de la red neuronal (Sección 3.2.4). La Tabla 1 sintetiza estas diferencias.

Ambos modelos se entrenan sobre el mismo dataset base y se evalúan bajo el mismo protocolo held-out cross-signer (Sección 3.1.3), lo que aísla el efecto de las decisiones de diseño del efecto de los datos.

3.2.1. Modelo de Referencia: Sistema Original de DialogApp

El punto de partida de esta memoria es el módulo de reconocimiento implementado durante el desarrollo de DialogApp en la 32^a Feria de Software de la Universidad Técnica Federico Santa María. Su análisis permite identificar las limitaciones técnicas que motivan el diseño de la arquitectura propuesta.

El modelo original consiste en una pila de cuatro capas LSTM (*Long Short-Term Memory*) apiladas, seguidas de tres capas densas de reducción progresiva:

1. **Enmascaramiento:** Una capa *Masking* que ignora los fotogramas rellenos con ceros al homogeneizar las secuencias para el procesamiento en lotes.
2. **Capas LSTM:** Cuatro capas apiladas de 128, 64, 64 y 64 unidades respectivamente. Solo la última capa no retorna la secuencia completa, actuando como cuello de botella que condensa la información temporal en un único vector de estado.
3. **Capas Densas:** Tres capas de 64, 32 y 16 neuronas con activación ReLU, seguidas de una capa de salida Softmax de 27 neuronas correspondientes a las letras del alfabeto dactilológico.

La arquitectura completa se ilustra en la Figura 6.

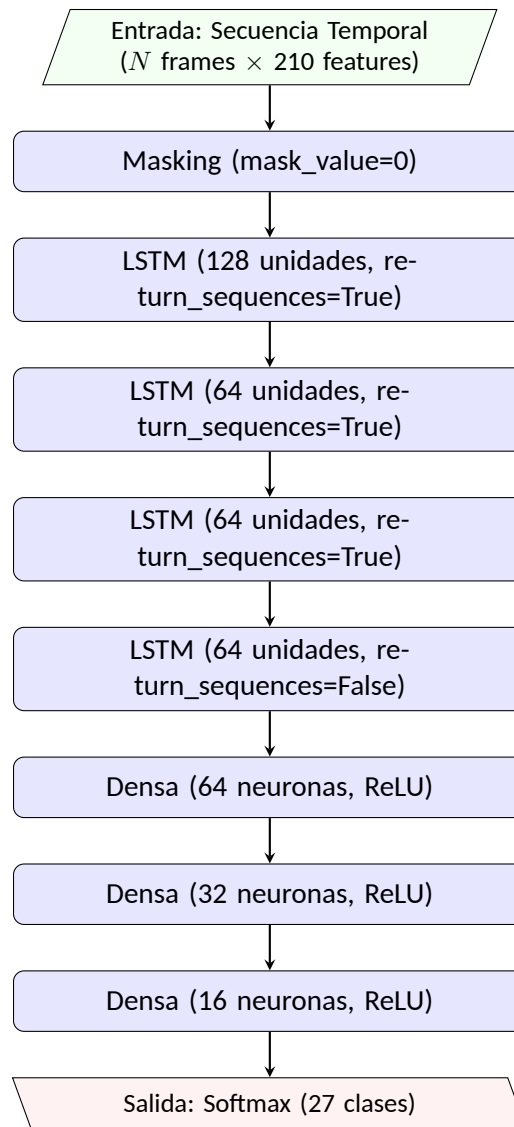


Figura 6: Arquitectura del modelo original de DialogApp. Cuatro capas LSTM apiladas actúan como codificador temporal, seguidas de tres capas densas de reducción progresiva.

El vector de entrada utiliza **210 características por fotograma**, construido para los 42 puntos de referencia de MediaPipe de la siguiente manera:

- **Coordenadas métricas (126 valores):** Los componentes (x, y, z) de los *world_landmarks* de MediaPipe, expresados en metros con origen en el centro geométrico de la mano.

- **Deltas de imagen (84 valores):** Las diferencias entre fotogramas consecutivos de las coordenadas de imagen normalizadas al encuadre, $(\Delta x_{img}, \Delta y_{img})$, para los mismos 42 puntos. Estas diferencias buscan codificar la velocidad de movimiento de cada punto de referencia.

Los deltas $(\Delta x_{img}, \Delta y_{img})$ introducen una inconsistencia en la representación: mientras las coordenadas métricas son invariantes a la distancia a la cámara, los deltas de imagen están normalizados al tamaño del encuadre en píxeles. El mismo gesto físico produce valores de delta distintos dependiendo de la distancia de la mano al lente, lo que obliga al modelo a aprender a compensar implícitamente esta variabilidad desde los datos. La propuesta descrita en los apartados siguientes reemplaza los deltas de imagen por distancias inter-digitales calculadas directamente sobre las coordenadas métricas, manteniendo una representación puramente métrica en las 156 dimensiones del vector de entrada.

3.2.2. Ingeniería de Características

A partir de este punto se describe el diseño de la **propuesta V1**. La ingeniería de características busca transformar los puntos de referencia entregados por MediaPipe en una representación matemática que facilite el aprendizaje del modelo, reemplazando el esquema de 210 features del modelo original (Sección 3.2.1) por una representación puramente métrica de 156 dimensiones.

A diferencia de los landmarks de imagen normalizados al tamaño del encuadre, los *world_landmarks* entregados por MediaPipe Hands son coordenadas tridimensionales métricas (en metros), intrínsecamente invariantes a la posición y distancia de la mano respecto a la cámara. Adicionalmente, el pipeline de extracción aplica una normalización de referencia restando el landmark de muñeca (P_0) a todos los puntos del fotograma, lo que fija la muñeca en el origen y elimina traslaciones residuales entre signers. Esta es la única normalización requerida: no se aplica escalado ni alineación adicional.

Ingeniería de Forma (Distancias Estructurales): Para reforzar la distinción entre señas con morfología similar, como la 'C' y la 'O', se incorporan características de distancia euclidiana entre puntos clave. Se seleccionan las puntas de los cinco dedos $T = \{P_4, P_8, P_{12}, P_{16}, P_{20}\}$ para calcular:

- **Distancias Inter-digitales:** Las 10 combinaciones posibles de distancias entre las puntas de los dedos ($\binom{5}{2} = 10$). Esto permite al modelo identificar el grado de apertura o cierre de la mano.
- **Distancias Punta-Muñeca:** La distancia de cada una de las 5 puntas respecto al punto de la muñeca (P_0), capturando la extensión longitudinal de los dedos.

Estas 15 distancias por mano (30 en total para el sistema bimanual) se computan directamente sobre las coordenadas métricas, sin normalización adicional. La Figura 7 ilustra el proceso completo de extracción.

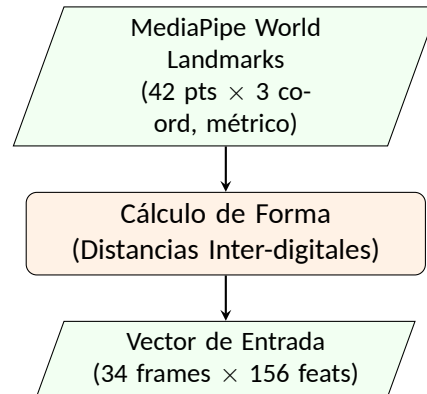


Figura 7: Detalle del proceso optimizado de extracción de características para la entrada del modelo.

El vector de entrada resultante para cada fotograma agrupa 156 dimensiones, representando un sistema bimanual estructurado en: coordenadas 3D métricas crudas (126, correspondientes a 21 puntos $\times$ 3 ejes $\times$ 2 manos) y distancias de estructura física (30). Los experimentos de ablación descritos en el Capítulo 4 confirmaron que esta combinación ofrece el mejor equilibrio entre precisión y estabilidad de entrenamiento.

3.2.3. Estrategias de Augmentation para Robustez

El modelo original de DialogApp (Sección 3.2.1) no incorporaba ninguna estrategia de augmentation. Para mejorar la capacidad de generalización del modelo frente a variaciones que no estaban presentes en el dataset original, se aplicaron técnicas de *Data Augmentation* durante el entrenamiento. Dado que el modelo opera exclusivamente sobre coordenadas tridimensionales, las transformaciones se aplicaron directamente en el espacio euclidiano:

- **Rotaciones Espaciales:** Aplicación de rotaciones aleatorias de $\pm 15^\circ$ sobre los ejes X e Y. Esto simula las inclinaciones naturales del smartphone cuando el usuario lo sostiene con una sola mano, forzando al modelo a aprender la estructura intrínseca de la señal independientemente de su orientación relativa a la cámara.
- **Ruido Gaussiano:** Inyección de perturbaciones aleatorias ($\sigma = 0,005$) en las coordenadas de los landmarks. Esta técnica permite que el modelo sea tolerante a pequeñas imprecisiones en la detección de MediaPipe causadas por desenfoque de movimiento o condiciones de iluminación subóptimas.

- **Inversión Lateral (Flipping):** Se duplicó el dataset mediante la inversión de los landmarks sobre el eje horizontal, permitiendo que el sistema reconozca señas ejecutadas tanto con la mano derecha como con la izquierda de manera ambidiestra.
- **Mixup Intra-clase:** Se generaron muestras sintéticas interpolando linealmente pares de secuencias de la misma clase con factor $\lambda \sim \text{Beta}(0,2, 0,2)$. Esta técnica suaviza las fronteras de decisión del modelo, penalizando predicciones excesivamente confías. Los experimentos de ablación (véase Figura 16 en el Capítulo 4) muestran que el mixup aplicado de forma aislada no mejora el desempeño; sin embargo, combinado con las rotaciones espaciales y el ruido gaussiano constituye la estrategia con mayor accuracy y menor varianza entre folds, lo que indica una mejora en la estabilidad del entrenamiento.

3.2.4. Arquitectura de la Red Neuronal

Se implementó una arquitectura de aprendizaje profundo diseñada específicamente para el procesamiento eficiente de secuencias temporales en dispositivos móviles, reemplazando la pila de cuatro capas LSTM del modelo original (Sección 3.2.1) por un bloque convolucional-recurrente. El modelo combina capas convolucionales para la extracción de rasgos locales y capas recurrentes para el modelado de la dinámica global del gesto.

La configuración final del modelo consiste en:

1. **Capa de Entrada:** Recibe tensores de dimensión (34, 156), correspondientes a los 34 fotogramas de la captura y las 156 características extraídas (coordenadas métricas y distancias estructurales).
2. **Capas Conv1D:** Dos capas convolucionales de 64 filtros con kernel de tamaño 5 y *padding* same. Actúan como extractor de rasgos espacio-temporales de bajo nivel: el kernel de tamaño 5 captura dependencias temporales locales más amplias que un kernel de tamaño 3, y el apilamiento de dos capas permite extraer jerarquías de patrones —micro-gestos en la primera capa, transiciones más largas en la segunda— antes de procesar la secuencia con las capas recurrentes. Esta configuración fue validada empíricamente, obteniendo una mejora de +2,8 pp respecto a la variante de una sola capa con kernel de tamaño 3.
3. **Layer Normalization:** Esta técnica se aplica después de la convolución para estabilizar la distribución de las activaciones. A diferencia de la Batch Normalization, normaliza a lo largo de la dimensión de características de cada instancia de forma independiente, lo que la hace especialmente adecuada para secuencias de longitud variable: las estadísticas de normalización no se ven afectadas por los ceros de relleno (*padding*)

introducidos al homogeneizar las secuencias para el procesamiento en lotes. Los experimentos de ablación confirman su superioridad sobre la Batch Normalization para esta tarea.

4. **Capas GRU Bidireccionales:** Se apilaron dos capas de unidades GRU (*Gated Recurrent Unit*) de 64 unidades cada una. Se seleccionaron unidades GRU en lugar de LSTM debido a su arquitectura interna más simplificada, lo que se traduce en un menor número de parámetros y una reducción significativa en la latencia de inferencia, factor crítico para la ejecución en tiempo real en dispositivos móviles. La naturaleza bidireccional permite que la red tenga acceso tanto al contexto pasado como al futuro de la secuencia, lo cual es fundamental para distinguir letras que comparten el inicio del movimiento pero difieren en su finalización.
5. **Capas Densas (Fully Connected):** Dos capas de 128 y 64 neuronas con activación ReLU, encargadas de realizar la combinación no lineal de las características temporales extraídas. Se aplicaron capas de *Dropout* [Srivastava *et al.*, 2014] de 0,2 y 0,3 para reducir el riesgo de sobreajuste, silenciando aleatoriamente una fracción de las neuronas durante el entrenamiento.
6. **Capa de Salida:** Una capa densa con activación Softmax de 27 neuronas, proporcionando la distribución de probabilidad sobre las 27 clases del alfabeto LSCh.

La arquitectura completa del modelo propuesto se detalla en la Figura 8.

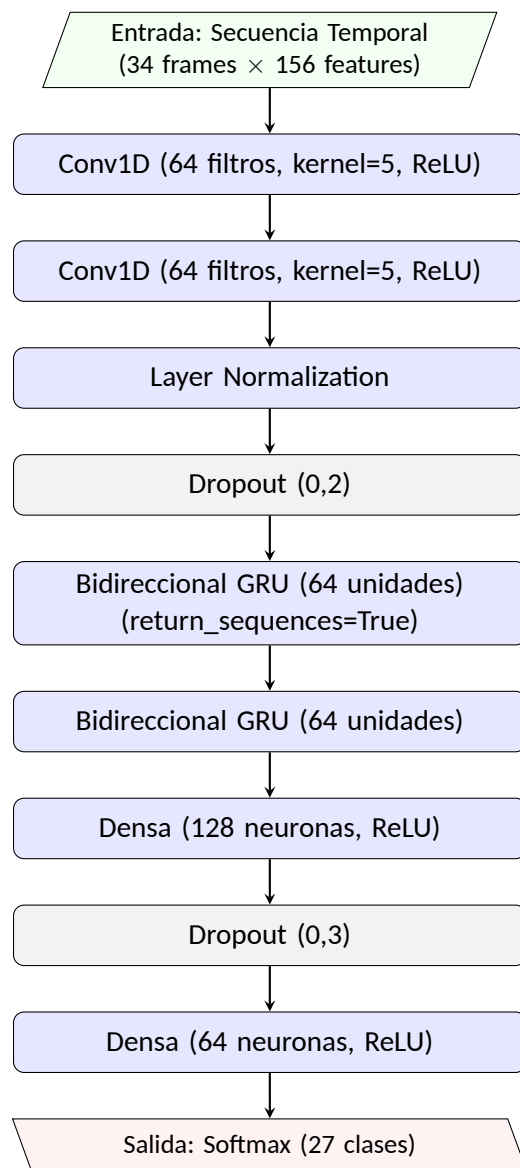


Figura 8: Arquitectura detallada del modelo V1 (Manos) para el reconocimiento de la dactilología LSCh.

La Tabla 1 sintetiza las diferencias de diseño entre el modelo original de DialogApp y la propuesta V1 descritas en esta sección. Los resultados de desempeño derivados de estas decisiones arquitectónicas se presentan y analizan en el Capítulo 4.

Aspecto	Original	V1 (Propuesta)
Capa frontal	—	2×Conv1D (64 filtros, kernel=5)
Celda recurrente	LSTM × 4 apiladas	BiGRU × 2 bidireccionales
Features por fotograma	210 (coords + deltas de imagen)	156 (coords + distancias inter-digitales)
Data Augmentation	No	Sí (espejado + rot. $\pm 15^\circ$ + mixup)

Tabla 1: Comparativa de decisiones de diseño entre el modelo original de DialogApp y la propuesta V1.

Estas decisiones de diseño se traducen en una ventaja de desempeño que se cuantifica en el Capítulo 4: la Propuesta V1 supera al modelo Original en generalización cross-signer (Tabla 3) y en latencia de inferencia TFLite (Tabla 5).

3.3. Implementación y Optimización en Android

El mayor desafío técnico de la memoria consistió en trasladar el modelo de un entorno de investigación (Python) a un entorno de producción móvil (Android) que garantizara una respuesta inmediata. Esta infraestructura de inferencia se utiliza para desplegar y medir **ambos modelos** (Original y V1) bajo idénticas condiciones de hardware, permitiendo una comparación de latencia justa en dispositivo (Tabla 5).

3.3.1. Infraestructura de Inferencia Nativa y Aceleración por Hardware

Para lograr una latencia mínima, se desarrolló un **Frame Processor Plugin** utilizando Kotlin y la librería *Vision Camera V4*. Para maximizar el rendimiento, se implementaron dos estrategias de optimización críticas:

- **Aceleración por GPU en detección de landmarks:** MediaPipe HandLandmarker opera con el delegate GPU (OpenGL ES) de forma automática, reduciendo la latencia de detección. El motor de inferencia TFLite del clasificador recurrente (GRU bidireccional) ejecuta en CPU, ya que las operaciones LSTM/GRU no están soportadas por el *GpuDelegate* de TFLite; la aceleración del clasificador queda como trabajo futuro mediante conversión a arquitecturas Conv1D o Transformer compatibles con GPU [Group, 2023].
- **Throttling del Bridge de comunicación:** Para evitar la saturación del puente (*bridge*) entre el código nativo y React Native, se implementó una lógica de filtrado que solo emite eventos de presencia de mano cuando ocurre un cambio de estado detectable, liberando recursos para el renderizado de la interfaz de usuario.

CAPÍTULO 4

VALIDACIÓN DE LA SOLUCIÓN

En el presente capítulo se exponen y analizan los resultados obtenidos tras la ejecución de la fase de entrenamiento del modelo V1. La validación se aborda mediante una evaluación estadística rigurosa, analizando el impacto de las decisiones de diseño arquitectónico y de ingeniería de características.

Las diferencias de diseño entre el modelo original de DialogApp y la propuesta V1 se resumen en la Tabla 1 del Capítulo 3. A continuación se presentan y analizan los resultados de desempeño derivados de dichas decisiones.

4.1. Metodología de Validación: Intra-Signer vs. Cross-Signer

Para garantizar la integridad científica de los resultados y evaluar la capacidad de generalización real del sistema, se implementaron dos esquemas de validación:

1. **Validación por Instancia de Video (Split Riguroso Intra-Signer):** Se implementó un protocolo donde la segregación se realiza estrictamente sobre los videos originales: si un video se asigna a testeo, ninguna de sus variantes (aumentadas, rotadas o espejadas) puede formar parte del entrenamiento. Bajo este esquema, el **modelo V1** alcanza un **98,89 %** de accuracy (detalle en la Tabla 2). Es importante precisar que este protocolo separa *instancias de video* pero no *signers*: las mismas personas que grabaron los videos de entrenamiento aparecen también en el conjunto de prueba, por lo que esta cifra mide la robustez del modelo V1 ante nuevas ejecuciones de usuarios *ya conocidos*.
2. **Validación Cross-Signer (Generalización a Usuarios Nuevos):** Para estimar el desempeño que un usuario nuevo experimentará realmente, se reservó un conjunto held-out grabado íntegramente por un signer que no participa en el entrenamiento. Este es el escenario más exigente y el único que mide verdadera generalización inter-persona. Bajo este protocolo el **modelo V1** alcanza un **85,3 % $\pm$ 2,6 %** de accuracy (media sobre 20 semillas).

La distinción entre ambos protocolos es metodológicamente relevante: como se detalla en la sección siguiente, el desempeño intra-signer (98,89 %, Tabla 2) es sustancialmente mayor que el cross-signer (85,3 %), un fenómeno esperado y bien documentado en reconocimiento de señas, donde el estilo individual de ejecución introduce variabilidad significativa entre personas. El valor **cross-signer del 85,3 %** es la estimación honesta del desempeño en producción y se adopta como cifra de referencia del sistema.

4.2. Comparativa con el Modelo de Referencia

Para cuantificar el avance respecto al sistema original de DialogApp, se evaluaron tres configuraciones bajo el mismo protocolo de validación intra-signer: split 80/20 estratificado por instancia de video (seed=42), que garantiza que ninguna variante aumentada de un video original se filtre entre los conjuntos de entrenamiento y prueba:

1. **Original:** El modelo de la feria, entrenado sin augmentation.
2. **Original+Aug:** El mismo modelo original entrenado con las técnicas de augmentation de V1 (espejado, rotación $\pm 15^\circ$ y mixup intra-clase), para aislar el efecto de la arquitectura del efecto de los datos.
3. **Propuesta Actual (V1):** La arquitectura Conv1D-GRU Bidireccional con ingeniería de características métrica, entrenada con las técnicas completas de *Data Augmentation* (espejado, rotación $\pm 15^\circ$ y mixup intra-clase).

Modelo	Accuracy	F1-Score	Latencia (ms)
Original	0.9407	0.9396	89.3
Original + Augmentation	0.9519	0.9522	62.5
Propuesta Actual (V1)	0.9889	0.9889	58.3

Tabla 2: Comparativa de accuracy, F1-Score y latencia de inferencia bajo protocolo de validación **intra-signer** (split 80/20 estratificado por instancia de video, seed=42; signers compartidos entre train y test). Las latencias son de inferencia: media sobre 50 inferencias consecutivas de una única instancia de video ($batch = 1$, secuencia completa de hasta 34 frames) tras 5 corridas de calentamiento descartadas, medida con `model.predict()` sobre el modelo Keras/TF (no TFLite) en CPU, entorno de entrenamiento Python — no equivale al pipeline Android reportado en la Sección 4.8.

Los resultados de la Tabla 2 revelan un avance simultáneo en precisión y eficiencia bajo evaluación intra-signer. La Propuesta Actual (V1) supera al modelo Original en **4,82 puntos porcentuales** de accuracy con una latencia de inferencia inferior en el entorno de ejecución Python.

Para cuantificar la generalización a usuarios nuevos se replicó la comparativa bajo validación cross-signer pareada: los tres modelos se entrenaron con el mismo split fijo (85 % entrenamiento / 15 % validación interna, estratificado) repitiendo el proceso con 20 semillas distintas que controlan la inicialización de pesos y el augmentation, obteniendo así 20 observaciones pareadas por modelo evaluadas sobre el signer held-out no visto en entrenamiento. Los resultados se sintetizan en la Tabla 3.

Modelo	Accuracy cross-signer	F1-Score
Original	0,703 $\pm$ 0,041	0,674
Original + Augmentation	0,732 $\pm$ 0,042	0,709
Propuesta Actual (V1)	0,853 $\pm$ 0,026	0,839

Tabla 3: Comparativa cross-signer (generalización a un signer no visto en entrenamiento), media $\pm$ desviación estándar sobre 20 semillas.

En generalización cross-signer la Propuesta V1 supera al modelo Original en **15,0 puntos porcentuales** y al Original con augmentation en **12,1 puntos porcentuales**. Esta ventaja es estadísticamente significativa: el test de Wilcoxon de rangos con signo sobre las 20 semillas arroja $p = 0,000043$, con un intervalo de confianza bootstrap del 95 % para la diferencia de **[13,0 pp; 17,0 pp]** —el intervalo no incluye el cero. La ventaja se mantiene en las 20 semillas evaluadas (Δ mínimo +4,4 pp, máximo +22,2 pp), y la Propuesta exhibe además menor varianza (0,026 frente a 0,042), indicando un entrenamiento más estable. Nótese que la brecha real cross-signer (+12,1 pp) es sustancialmente mayor que la brecha intra-signer (+4,8 pp): la superioridad de la arquitectura propuesta emerge con más fuerza precisamente en el escenario de generalización a usuarios nuevos.

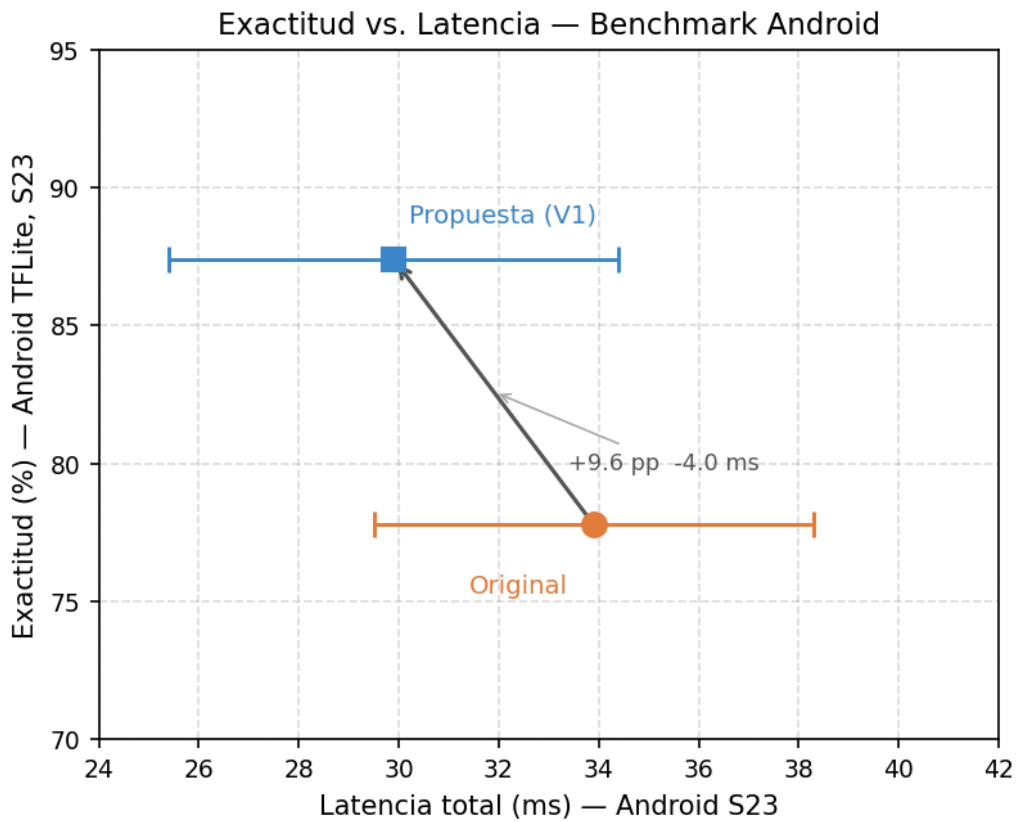


Figura 9: Tradeoff exactitud vs. latencia en dispositivo real (Samsung Galaxy S23). Exactitud medida como TFLite Android sobre conjunto held-out (135 clips, signer no visto, modelo seed=42); latencia corresponde al pipeline completo Android (MediaPipe + preproceso + inferencia TFLite). La Propuesta (V1) domina en ambas dimensiones: mayor exactitud (+9,6 pp) y menor latencia (-4,0 ms).

La Figura 9 muestra que la Propuesta (V1) domina al modelo Original en ambas dimensiones simultáneamente: logra una exactitud 9,6 puntos porcentuales superior (87,4 % vs. 77,8 %) con una latencia de pipeline 4,0 ms menor (29,9 ms vs. 33,9 ms). Esta doble mejora refleja que la arquitectura propuesta no incurre en el tradeoff típico donde mayor capacidad implica mayor costo computacional.

La Figura 10 muestra las curvas de convergencia del accuracy durante el entrenamiento. La V1 alcanza su desempeño máximo en aproximadamente 25 épocas, frente a las más de 50 requeridas por el modelo Original, con una brecha de generalización consistentemente menor, lo que evidencia mayor estabilidad del proceso de aprendizaje.

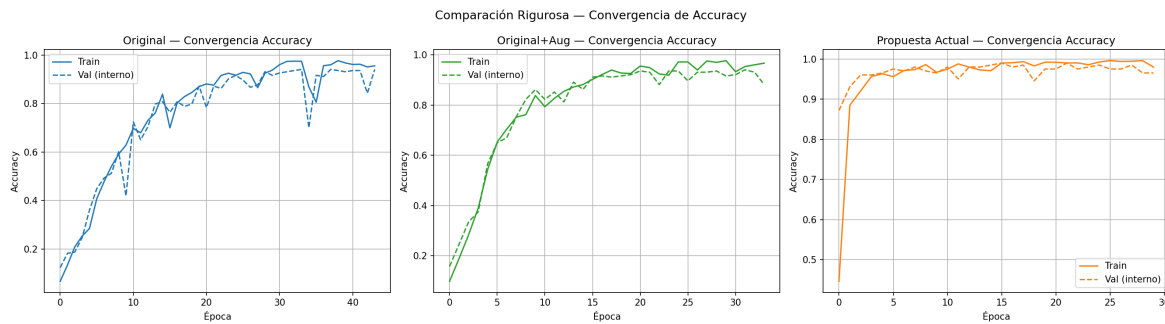


Figura 10: Curvas de convergencia del accuracy de entrenamiento y validación interna para los tres modelos. La Propuesta Actual (V1) converge en aproximadamente la mitad de épocas que el modelo Original, con menor varianza en la curva de validación.

4.3. Comparativa Arquitectónica: Conv1D y Celda Recurrente

Se realizaron dos experimentos de benchmark independientes para validar las decisiones de diseño arquitectónico: la elección de la capa convolucional frontal y la elección de la celda recurrente. La ablación de la capa convolucional emplea validación *StratifiedKFold* (5 folds), que garantiza que ninguna variante aumentada de un video original se filtre entre los pliegues de entrenamiento y validación interna; la ablación de la celda recurrente emplea un protocolo de 10 semillas con *StratifiedShuffleSplit* (sin K-Fold). Ambos experimentos evalúan el desempeño final sobre el mismo conjunto held-out cross-signer, independiente del particionado de entrenamiento.

4.3.1. Ablación de la Capa Convolucional

La Figura 11 compara tres variantes de la capa frontal Conv1D bajo el mismo protocolo de validación cross-signer. La configuración *Stacked* de $2 \times \text{Conv1D}(64,5)$ supera al *Baseline* de una sola capa con kernel de tamaño 3 en +2,8 pp de accuracy, con una varianza notablemente menor entre folds, lo que evidencia mayor estabilidad del entrenamiento. La variante dilatada (dilation rates 1+2) no supera a la apilada estándar, por lo que se descarta.

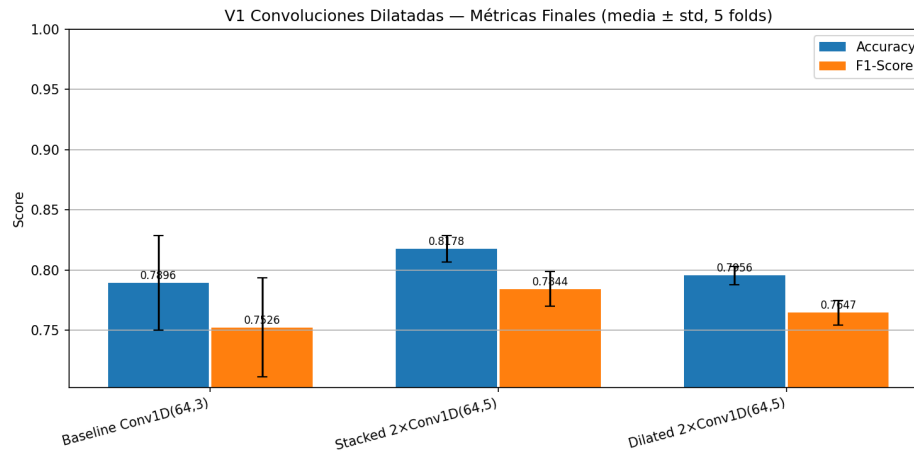


Figura 11: Comparativa de variantes convolucionales bajo validación cross-signer (StratifiedKFold 5 folds). *Baseline*: Conv1D(64,3). *Stacked*: 2×Conv1D(64,5) sin dilatación. *Dilated*: 2×Conv1D(64,5) con dilation rates 1 y 2.

4.3.2. Ablación de la Celda Recurrente

Se validó la elección de la celda recurrente y la direccionalidad de las capas. Los resultados, resumidos en la Tabla 4 y visualizados en la Figura 12, validan la selección de la GRU Bidireccional como celda recurrente de la arquitectura propuesta.

Arquitectura	Accuracy	F1-Score
GRU Bidireccional	0,806 ± 0,042	0,790 ± 0,045
LSTM Bidireccional	0,780 ± 0,045	0,757 ± 0,049
GRU Unidireccional	0,779 ± 0,031	0,758 ± 0,035
LSTM Unidireccional	0,742 ± 0,059	0,715 ± 0,071

Tabla 4: Comparativa de desempeño entre distintas configuraciones recurrentes para V1 (media ± desviación estándar, 10 seeds, evaluado sobre held-out cross-signer).

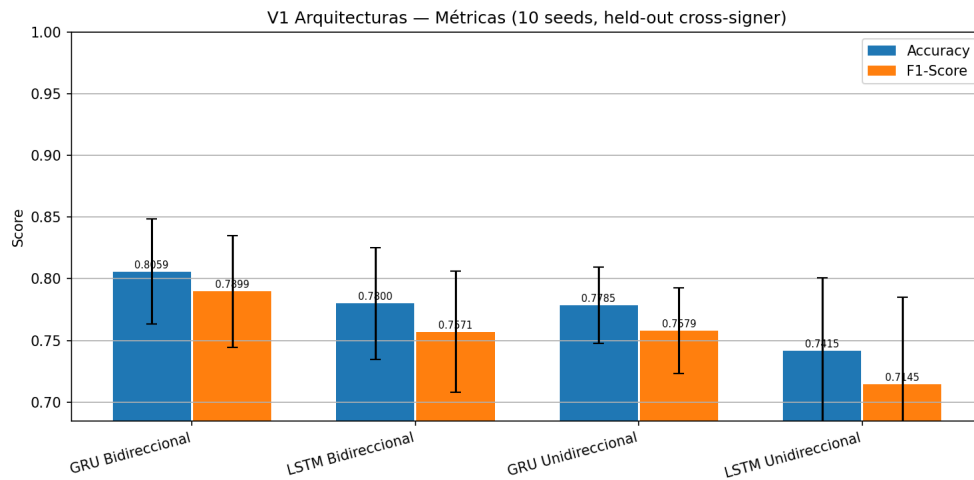


Figura 12: Comparativa de Accuracy y F1-Score entre distintas arquitecturas. La GRU Bidireccional obtiene el mayor desempeño promedio sobre el conjunto held-out cross-signer.

La **GRU Bidireccional** obtuvo el mayor desempeño promedio (Accuracy $0,806 \pm 0,042$), superando a la LSTM Bidireccional ($0,780 \pm 0,045$) con una diferencia dentro del margen de varianza. Su ventaja principal respecto a la LSTM radica en la menor cantidad de parámetros entrenables al prescindir de la compuerta de olvido, lo que favorece su exportación a TensorFlow Lite con menor huella de memoria.

4.4. Análisis de Ablación: Impacto de las Características y la Normalización

4.4.1. Impacto de las Características

La ingeniería de características (Feature Engineering) manual demostró ser más efectiva que el aprendizaje directo sobre coordenadas brutas. El impacto incremental de cada módulo, evaluado bajo el protocolo de validación cross-signer (StratifiedKFold, 5 folds, entrenado sobre el conjunto de entrenamiento y evaluado sobre el held-out de un signer no visto), se detalla en la Figura 13.

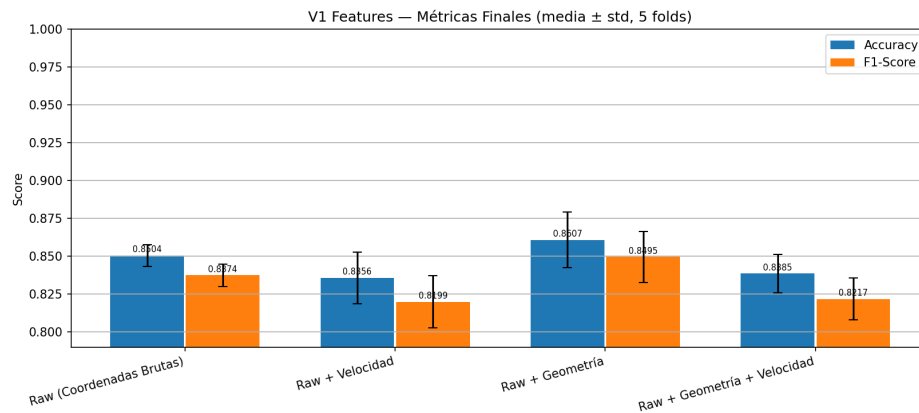


Figura 13: Impacto de la ingeniería de características en el Accuracy y F1-Score, evaluado sobre held-out cross-signer. La combinación Raw + Geometría obtiene el mayor desempeño promedio; agregar velocidad degrada el resultado en ambos casos.

La combinación **Raw + Geometría** obtuvo el mayor desempeño promedio (Accuracy $0,861 \pm 0,018$), superando a Raw solo ($0,850 \pm 0,007$) en +1,0 pp, y a las variantes que incorporan velocidad ($0,839$ y $0,836$). Agregar la velocidad (derivada temporal de la posición) degrada el desempeño en ambas variantes que la incluyen, lo que sugiere que introduce ruido de estilo personal (rapidez de ejecución) que no generaliza entre signers; por esta razón la arquitectura final de la propuesta prescinde de la velocidad.

- **Coordenadas *World* de MediaPipe:** Los *world_landmarks* proveen invarianza de escala y posición respecto a la cámara de forma intrínseca, al estar expresados en metros con origen aproximado en el centro de la palma. Adicionalmente, el pipeline de extracción aplica una normalización de referencia: se resta el landmark de muñeca (P_0) a todas las coordenadas, fijando la muñeca en el origen y haciendo la representación invariante a traslaciones residuales entre signers. Esta es la única normalización requerida.
- **Estructura Física (Distancias):** El cálculo de las 30 distancias inter-digitales y punta-muñeca fue el componente de mayor impacto. Permitió al modelo resolver ambigüedades en señas de morfología similar (como 'C' y 'O') mediante información geométrica explícita sobre el grado de apertura y extensión de los dedos, tal como refleja la mejora de accuracy sobre Raw solo reportada anteriormente.

4.4.2. Impacto de la Normalización

Se evaluó el impacto de la normalización por capas versus la normalización por lotes, bajo el mismo protocolo de validación cross-signer (Figura 14).

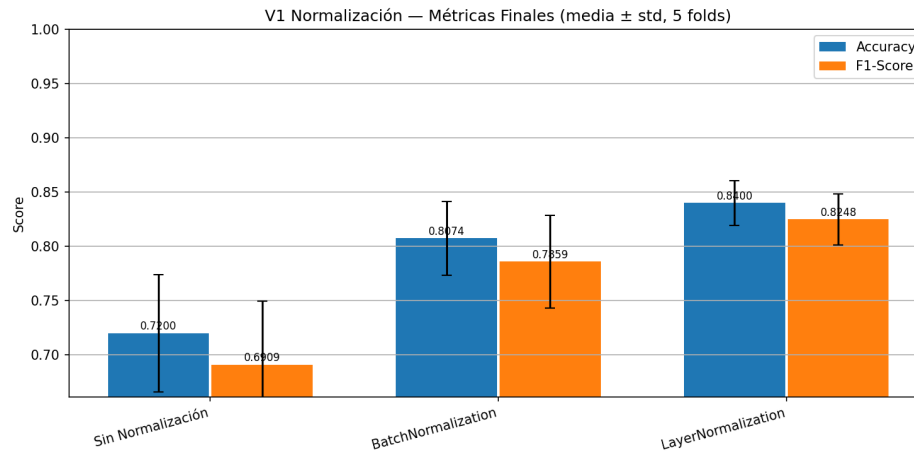


Figura 14: Comparativa de técnicas de normalización (media $\pm$ desviación estándar, 5 folds, evaluado sobre held-out cross-signer). LayerNormalization obtiene el mayor desempeño promedio con la menor varianza entre folds, consistente con normalizar por muestra en vez de por batch — ventaja teórica en secuencias de longitud variable.

La **LayerNormalization** obtuvo el mayor desempeño promedio (Accuracy $0,840 \pm 0,021$), superando a la BatchNormalization ($0,807 \pm 0,034$) y a la ausencia de normalización ($0,720 \pm 0,054$). Además de la mejora en accuracy, presenta la menor dispersión entre folds de las tres configuraciones, lo que respalda empíricamente el argumento teórico de que normalizar por muestra (en vez de por batch) favorece la estabilidad ante secuencias de longitud variable [Ba *et al.*, 2016].

4.5. Eficacia del Data Augmentation

El uso de técnicas de aumento de datos aporta una mejora moderada bajo el protocolo honesto de validación cross-signer, como se observa en la comparativa de la Figura 15. La ganancia máxima observada es de +1,04 pp de accuracy (Espejado + Rotación, 0,8326 frente a 0,8222 de Base), sensiblemente menor a la que sugeriría una evaluación intra-signer. Se observaron los siguientes impactos:

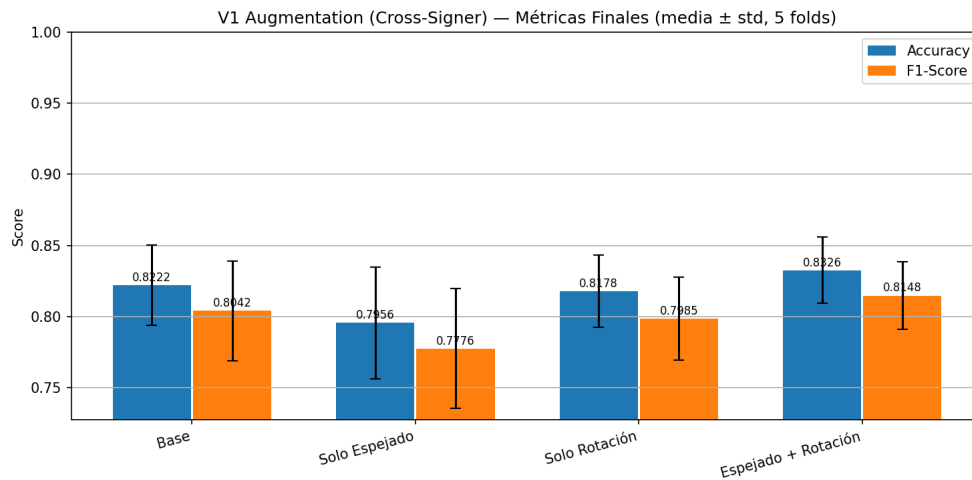


Figura 15: Desempeño del modelo V1 según la estrategia de Data Augmentation aplicada (media $\pm$ desviación estándar, StratifiedKFold 5 folds, evaluado sobre held-out cross-signer). La combinación de Espejado y Rotación obtiene el mayor accuracy promedio.

- **Espejado Horizontal:** Aplicado de forma aislada, reduce el desempeño respecto a Base (0,7956 frente a 0,8222) bajo validación cross-signer. Si bien permite que el modelo sea ambidiestro de forma nativa, no aporta diversidad de signers y, sin la rotación que lo acompañe, no se traduce en mejor generalización a un usuario nuevo.
- **Rotación ($\pm 15^\circ$):** Obtiene un desempeño similar a Base (0,8178), dentro del margen de varianza entre folds. Su aporte principal es mitigar la degradación cuando el smartphone no se sostiene en un ángulo perfectamente vertical, situación común en el uso cotidiano.
- **Combinación (Espejado + Rotación):** Solo al combinarse ambas transformaciones se obtiene una mejora consistente sobre Base, con la menor dispersión relativa entre folds de las cuatro condiciones evaluadas.

La Figura 16 extiende el análisis anterior incorporando el efecto del mixup intra-clase sobre la augmentación geométrica completa (rotación + ruido gaussiano). Los resultados muestran que el mixup aplicado de forma aislada no mejora el desempeño base; sin embargo, su combinación con las transformaciones espaciales (*v1_rot_mixup*) constituye la estrategia con mayor accuracy (+2,4 pp sobre *v1_sin_aug*) y la menor varianza entre folds (std=0,018), confirmando que el beneficio del mixup emerge únicamente en sinergia con la augmentación geométrica.

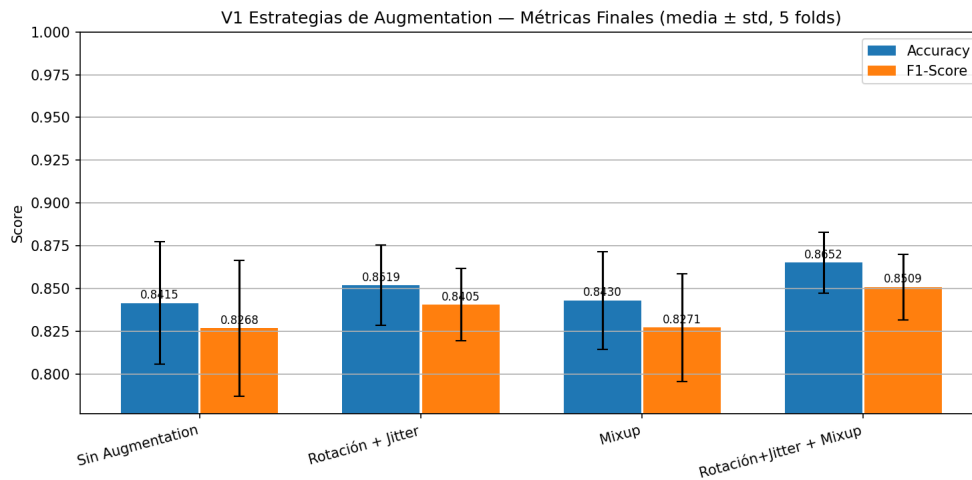


Figura 16: Comparativa de estrategias de augmentation bajo validación cross-signer (StratifiedKFold 5 folds). *Sin Augmentation*: sin augmentation. *Rotación + Jitter*: rotaciones $\pm 15^\circ$ y ruido gaussiano. *Mixup*: mixup intra-clase únicamente. *Rotación+Jitter + Mixup*: combinación de ambas estrategias (configuración final del modelo).

Para validar que este incremento en el desempeño no se debe a un sobreajuste de los datos aumentados, la evidencia más concluyente es el propio resultado cross-signer: el modelo alcanza en promedio un 85,3 % de accuracy sobre un signer completamente ajeno al entrenamiento. Un modelo que hubiera memorizado las particularidades de los datos aumentados no lograría generalizar con ese margen a un usuario nuevo. Las curvas de convergencia de la Figura 10 confirman además que la brecha entre accuracy de entrenamiento y validación interna se mantiene estable a lo largo de las épocas, sin señales de sobreajuste.

4.6. Análisis de Matriz de confusión

La matriz de confusión (Figura 17) presenta los valores absolutos de predicción sobre el conjunto held-out cross-signer ($n = 135$ clips, signer no visto en entrenamiento), reflejando el comportamiento real del modelo ante un usuario nuevo.

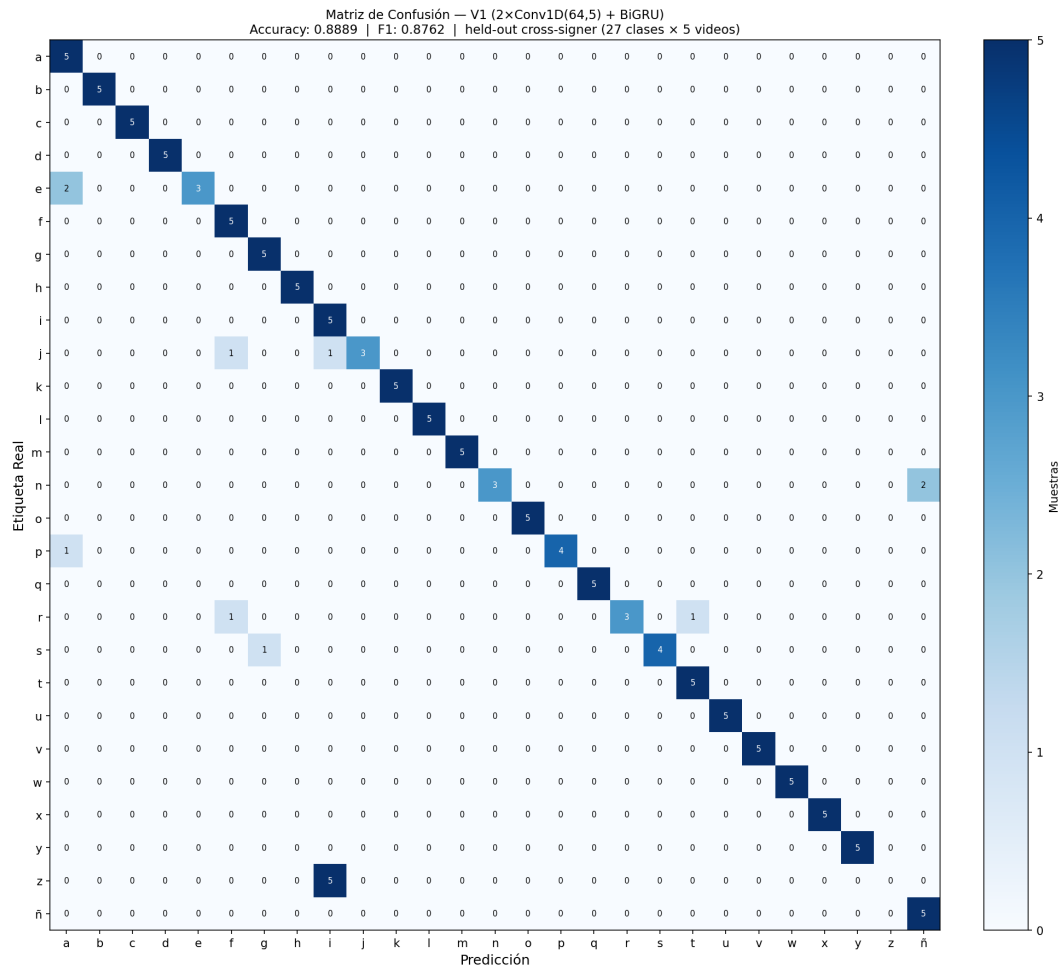


Figura 17: Matriz de confusión para las 27 clases de la LSCh. Valores absolutos sobre el conjunto held-out cross-signer ($n = 135$). Cada fila corresponde a la etiqueta real y cada columna a la clase predicha.

Las observaciones más relevantes son:

- **Clases perfectas:** 20 de las 27 clases del alfabeto (A, B, C, D, F, G, H, I, K, L, M, O, Q, T, U, V, W, X, Y, Ñ) no presentan ningún error de clasificación, evidenciando robustez del modelo ante un signer no visto para la gran mayoría de configuraciones dactilológicas.
- **Confusión N/Ñ:** La confusión más frecuente es 'N' clasificada como 'Ñ' (2 instancias). Ambas letras comparten idéntica configuración manual; su distinción radica únicamente en el desplazamiento lateral de la 'Ñ', un rasgo dinámico sutil que varía significativamente entre signers.
- **Confusión E/A:** La letra 'E' es confundida con 'A' en 2 instancias. Ambas comparten

una postura de dedos cerrados con diferencias geométricas menores en la posición del pulgar, lo que dificulta la generalización a estilos de ejecución no vistos.

- **Confusión J/I:** La letra 'J' (dinámica, traza un arco) es confundida con 'I' en 1 instancia. La postura inicial del gesto de la 'J' es estructuralmente similar a la 'I', y la trayectoria completa no siempre es capturada dentro de la ventana de 34 fotogramas ante variaciones de velocidad inter-signer.
- **Confusión R/T:** La 'R' es confundida con 'T' en 1 instancia, atribuible a la cercanía del pulgar respecto a los dedos índice y medio en ambas configuraciones, sensible al ángulo de captura.
- **Letra Z:** La 'Z' presenta 0 aciertos en el conjunto cross-signer (5 instancias predichas como 'I'). La configuración inicial y final de la 'Z' es estructuralmente idéntica a la 'I' (meñique extendido), y la trayectoria en zigzag depende fuertemente del estilo de ejecución del signer. Esta confusión constituye el principal límite del modelo en generalización cross-signer y representa la oportunidad de mejora más relevante para trabajo futuro.
- **Confusiones aisladas sin similitud estructural clara:** 'J' hacia 'F', 'R' hacia 'F', 'P' hacia 'A' y 'S' hacia 'G' presentan 1 instancia de error cada una, sin una configuración manual compartida evidente entre las señas involucradas. Son consistentes con variación individual del signer o ruido puntual de tracking más que con una limitación estructural sistemática del modelo. Es notable que 'F' aparece como predicción errónea en dos pares distintos (J y R) pese a que su propia clase no presenta ningún error (5/5 correcto), lo que sugiere que 'F' podría actuar como un atractor espurio en el espacio de características del modelo, sin que esto sea concluyente con el tamaño de muestra disponible.

4.7. Precisión del Modelo en el Entorno Android

La precisión reportada hasta aquí corresponde al modelo en formato Keras ejecutado en PC. Dado que el tercer objetivo específico exige evaluar el rendimiento **en el dispositivo objetivo**, se midió la precisión del artefacto `.tflite` —el mismo bytecode que ejecuta la aplicación Android— corriéndolo con el intérprete TensorFlow Lite directamente en el dispositivo Android sobre el conjunto held-out cross-signer. El modelo `.tflite` (seed=42) alcanzó un accuracy de **87,4 %** sobre el conjunto held-out en el dispositivo Android, con una diferencia de **0,00 puntos porcentuales** respecto al modelo Keras equivalente. Esta diferencia nula confirma que la conversión a TensorFlow Lite —sin cuantización, preservando la aritmética de punto flotante— no degrada en absoluto la capacidad predictiva del sistema. El valor 87,4 % corresponde al modelo de semilla 42; la media sobre 20 semillas es 85,3 % $\pm$ 2,6 %.

4.8. Latencia del Pipeline Completo en Dispositivo Android

Para medir la velocidad de respuesta del sistema en condiciones controladas, se instrumentó la aplicación Android con temporización por fases (`SystemClock.uptimeMillis()`) y se procesaron los 135 clips del conjunto de validación cross-signer sobre dos dispositivos: Samsung Galaxy S20 FE 5G (Exynos 990) y Samsung Galaxy S23 (Snapdragon 8 Gen 2). La medición cubre tres componentes: detección de landmarks con MediaPipe HandLandmarker (`mp_ms`), extracción de características (`prep_ms`) e inferencia TensorFlow Lite (`tf_ms`). El benchmark del S20 FE utilizó MediaPipe en modo CPU; el del S23 utilizó MediaPipe en modo GPU (OpenGL ES), replicando las condiciones del modo de uso en vivo. El clasificador TFLite ejecuta en CPU en ambos casos, ya que los delegates GPU y NPU de TFLite no son compatibles con las operaciones LSTM y BiGRU recurrentes del modelo.

Modelo	Dispositivo	MediaPipe (ms)	Preproc (ms)	TFLite (ms)	Total (ms)
Original	S20 FE 5G	66,3 ± 18,7	0,1	10,0 ± 4,6	76,4 ± 20,2
Original	S23	20,3 ± 2,3	0,1	13,5 ± 3,4	33,9 ± 4,4
V1 (Propuesta)	S20 FE 5G	72,3 ± 21,4	0,1	8,6 ± 4,6	81,0 ± 22,6
V1 (Propuesta)	S23	19,5 ± 2,6	0,2	10,2 ± 2,4	29,9 ± 4,5

Tabla 5: Latencia media $\pm$ desviación estándar del pipeline completo ($n = 135$ clips por fila). S20 FE: MediaPipe en CPU. S23: MediaPipe en GPU (OpenGL ES), TFLite en CPU. La V1 reduce la inferencia TFLite un 14 % en el S20 FE y un 24 % en el S23 respecto al modelo original.

El comportamiento del pipeline difiere entre dispositivos según el delegate de MediaPipe. En el S20 FE (MediaPipe CPU), HandLandmarker constituye el cuello de botella, representando el 87–89 % de la latencia total. En el S23 (MediaPipe GPU), la detección cae a ~20 ms, reduciendo el total a 29,9–33,9 ms y elevando la tasa de inferencia a **~30–33 fps**. En ambos dispositivos, la V1 reduce la inferencia TFLite respecto al original: –14 % en el S20 FE (8,6 ms frente a 10,0 ms) y –24 % en el S23 (10,2 ms frente a 13,5 ms), coherente con la reducción de características de 210 a 156.

CONCLUSIONES

El presente trabajo propuso, implementó y evaluó una arquitectura de reconocimiento del alfabeto dactilológico de la LSCh en tiempo real para dispositivos Android, con el objetivo de superar al modelo original de la aplicación DialogApp en precisión y eficiencia computacional.

Hallazgos Principales

Los resultados obtenidos demuestran que la propuesta (V1) supera consistentemente al modelo original en todos los escenarios de evaluación considerados.

Bajo evaluación **cross-signer** —el escenario más exigente y representativo, en el que el modelo debe generalizar a un signer completamente ajeno al entrenamiento— la propuesta alcanzó una exactitud media de 85,3 % sobre 20 semillas, frente al 70,3 % del modelo original, una mejora de **+15,0 puntos porcentuales**. Esta diferencia es estadísticamente significativa (Wilcoxon signed-rank, $W = 210,0$, $p = 4,4 \times 10^{-5}$; IC 95 % bootstrap [+13,0 pp; +17,0 pp]) y se mantiene en la totalidad de las 20 semillas evaluadas, con un delta mínimo de +4,4 pp y máximo de +22,2 pp.

Bajo evaluación **intra-signer**, la propuesta alcanzó un 98,9 % de exactitud frente al 94,1 % del modelo original (+4,8 pp), con menor latencia de inferencia en el entorno de desarrollo Python.

En el **dispositivo objetivo** (Samsung Galaxy S23), la exactitud TFLite de la propuesta fue de 87,4 % frente al 77,8 % del modelo original (+9,6 pp), con una latencia total de pipeline de 29,9 ms frente a 33,9 ms (−4,0 ms, −12 %). La inferencia TFLite de la propuesta es un 24 % más rápida que la del original (10,2 ms frente a 13,5 ms), pese a ser una arquitectura más expresiva. El throughput resultante de ≈ 33 fps es suficiente para reconocimiento en tiempo real.

La doble mejora simultánea en exactitud y latencia demuestra que la arquitectura propuesta no incurre en el tradeoff típico donde mayor capacidad implica mayor costo computacional. Esto se explica principalmente por la sustitución de cuatro capas LSTM apiladas (arquitectura secuencial profunda) por capas Conv1D que extraen patrones locales en paralelo antes de las capas BiGRU, reduciendo la longitud efectiva de la secuencia que deben procesar las capas recurrentes.

Decisiones de Diseño Validadas

Los experimentos de ablación permiten atribuir la mejora a decisiones de diseño específicas:

- **Arquitectura Conv1D + BiGRU:** La combinación de dos capas Conv1D(64, kernel=5) seguidas de dos capas BiGRU(64) superó a todas las variantes alternativas evaluadas (LSTM bidireccional, GRU unidireccional, LSTM unidireccional), con menor varianza entre semillas.
- **Ingeniería de características:** Las 30 distancias estructurales inter-digitales y punta-muñeca calculadas sobre coordenadas métricas fueron el componente de mayor impacto en la ablación, permitiendo al modelo resolver ambigüedades entre señas de morfología similar.
- **Data augmentation:** La combinación de espejado horizontal, rotación y mixup intra-clase (*v1_rot_mixup*) mejoró el accuracy en +2,4 pp respecto a la variante sin augmentation, con la menor varianza entre folds (std=0,018) de las estrategias evaluadas, bajo ablation dedicado de 5 folds sobre la arquitectura V1 (Figura 16).

Limitaciones

La principal limitación del trabajo es que el conjunto de evaluación cross-signer proviene de un **único signer** (5 videos por clase, 135 clips totales). Si bien este diseño garantiza que el signer de test no aparece en entrenamiento, la generalización del modelo a la población de signers de la LSCh no puede afirmarse con certeza a partir de un solo individuo. Los resultados deben interpretarse como evidencia de que la arquitectura es capaz de generalizar más allá de los signers de entrenamiento, pero no como una estimación representativa de la precisión esperable en usuarios arbitrarios.

Una segunda limitación es la **representación de entrada**: el modelo utiliza exclusivamente los 42 landmarks de mano (21 por mano) entregados por MediaPipe Hands. Esta elección es suficiente para la mayoría de las configuraciones dactilológicas estáticas, pero excluye información relevante para un subconjunto de señas. Las confusiones residuales observadas en el análisis de errores —Z clasificada como I, N confundida con Ñ, J confundida con I— involucran precisamente señas cuya distinción depende de la trayectoria del movimiento o de la orientación del brazo, aspectos que landmarks de muñeca y cuerpo (pose landmarks) podrían capturar. La incorporación de landmarks de pose o de la muñeca extendida queda como línea de mejora directa.

Una tercera limitación es la cobertura del dataset: 27 clases del alfabeto dactilológico con 50 videos de entrenamiento por clase, grabados en condiciones controladas. El sistema no

fue evaluado sobre señas dinámicas complejas (palabras o frases), ni sobre variaciones de iluminación, fondo o velocidad de ejecución.

Trabajo Futuro

Las siguientes líneas de trabajo se identifican como prioritarias para extender los resultados de esta memoria:

- **Diversificación de signers:** Incorporar al menos 5–10 signers distintos en el conjunto de evaluación para obtener una estimación estadísticamente representativa de la generalización. Idealmente, segregar signers entre train y test desde la recolección de datos.
- **Arquitectura compatible con delegate GPU:** El clasificador TFLite ejecuta en CPU porque las operaciones BiGRU no son soportadas por el delegate GPU ni por NNAPI. Sustituir las capas recurrentes por una arquitectura puramente convolucional (Conv1D profundo o Transformer) desbloquearía la aceleración por GPU en TFLite, reduciendo la dependencia del cuello de botella de MediaPipe en dispositivos de gama media.
- **Señas dinámicas completas:** Extender el sistema al reconocimiento de palabras y frases en LSCh, que involucran movimientos bimanuales complejos y expresiones faciales como componentes gramaticales.
- **Dataset aumentado:** Escalar el corpus de entrenamiento en número de clases y volumen por clase, incorporando variaciones controladas de iluminación, fondo y ángulo de captura para mejorar la robustez en condiciones de uso real.
- **Incorporación de landmarks de pose:** El modelo actual utiliza exclusivamente los 42 landmarks de mano de MediaPipe Hands. Agregar los landmarks de cuerpo (*pose landmarks*) permitiría capturar la orientación del brazo y la trayectoria de la muñeca, información discriminativa para las señas con mayor tasa de confusión residual (Z clasificada como I, N confundida con Ñ, J confundida con I).

ANEXOS

4.1. Anexo A: Alfabeto Dactilológico de la Lengua de Señas Chilena (LSCh)

A continuación se detallan las señas que componen el alfabeto dactilológico de la LSCh utilizado para el entrenamiento del modelo. Se destaca la inclusión de las 27 letras del abecedario español, adaptadas a la expresión viso-gestual chilena.

- **Letras Estáticas:** A, B, C, D, E, F, H, I, K, L, M, N, O, P, Q, R, T, U, V, W, Y.
- **Letras Dinámicas:** G, J, Ñ, S, X, Z.

4.2. Anexo B: Especificación del Modelo de Red Neuronal (Keras/Python)

Se presenta un extracto del código fuente utilizado para definir la arquitectura del modelo en Python utilizando la API secuencial de Keras. Este modelo fue posteriormente convertido a formato TFLite para su despliegue en Android.

```
def get_v1_model(input_shape, num_classes):
    model = Sequential([
        layers.Input(shape=input_shape),

        # Extracción de patrones temporales locales (2 capas, kernel=5)
        layers.Conv1D(64, kernel_size=5, padding='same', activation='relu'),
        layers.Conv1D(64, kernel_size=5, padding='same', activation='relu'),
        layers.LayerNormalization(),
        layers.Dropout(0.2),

        # Modelado de dependencias temporales bidireccionales
        layers.Bidirectional(layers.GRU(64, return_sequences=True)),
        layers.Bidirectional(layers.GRU(64)),

        # Bloque de clasificación densa
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.3),
        layers.Dense(64, activation='relu'),
        layers.Dense(num_classes, activation='softmax')
    ])

```

```
model.compile(loss='categorical_crossentropy', optimizer='adam')
return model
```

4.3. Anexo C: Implementación del Motor de Inferencia (Kotlin/Android)

Se presenta la implementación optimizada del motor de inferencia en Android, destacando el cálculo de distancias estructurales y la aceleración por hardware.

```
class InferenceProcessor(context: Context) {
    private val tflite: Interpreter
    // 42 landmarks × 3 coords + 15 distancias/mano × 2 manos = 156
    private val featuresSize = 156

    // Índices MediaPipe de puntas de dedos y muñeca
    private val TIP_INDICES = intArrayOf(4, 8, 12, 16, 20)
    private val WRIST_INDEX = 0

    init {
        // BiGRU no es compatible con GPU/NPU delegates de TFLite → CPU exclusivo
        val options = Interpreter.Options().apply { numThreads = 4 }
        tflite = Interpreter(loadModelFile(context, "model_v1.tflite"), options)
        tflite.allocateTensors()
    }

    fun process(worldBuffer: ArrayDeque<Array<FloatArray>>): InferenceResult? {
        // Tensor de entrada [1][34][156]
        val inputTensor = Array(1) { Array(34) { FloatArray(featuresSize) } }

        for (i in 0 until 34) {
            val frame = worldBuffer[i] // 42 landmarks × 3 (x,y,z en metros)
            var idx = 0

            // 1. World landmarks aplanados: 42 × 3 = 126 valores
            for (lm in 0 until 42) {
                inputTensor[0][i][idx++] = frame[lm][0]
                inputTensor[0][i][idx++] = frame[lm][1]
                inputTensor[0][i][idx++] = frame[lm][2]
            }

            // 2. Distancias inter-digitales: 15 por mano × 2 = 30 valores
```

```
41      // Mano derecha: landmarks 0-20 | Mano izquierda: landmarks 21-  
  
      for (handOffset in intArrayOf(0, 21)) {  
          // 10 distancias punta-punta C(5,2)  
          for (a in TIP_INDICES.indices) {  
              for (b in a + 1 until TIP_INDICES.size) {  
                  inputTensor[0][i][idx++] = euclidean(  
                      frame[handOffset + TIP_INDICES[a]],  
                      frame[handOffset + TIP_INDICES[b]]  
                  )  
              }  
          }  
          // 5 distancias punta-muñeca  
          for (tip in TIP_INDICES) {  
              inputTensor[0][i][idx++] = euclidean(  
                  frame[handOffset + tip],  
                  frame[handOffset + WRIST_INDEX]  
              )  
          }  
      }  
  }  
  
  val outputBuffer = Array(1) { FloatArray(27) }  
  tflite.run(inputTensor, outputBuffer)  
  return decodePredictions(outputBuffer[0])  
}  
  
private fun euclidean(a: FloatArray, b: FloatArray): Float {  
    val dx = a[0] - b[0]; val dy = a[1] - b[1]; val dz = a[2] - b[2]  
    return Math.sqrt((dx*dx + dy*dy + dz*dz).toDouble()).toFloat()  
}  
}
```

REFERENCIAS BIBLIOGRÁFICAS

- [2016] Abadi, Martín and others (2016). Tensorflow: A system for large-scale machine learning. En *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283.
- [2023] Abid, Ahmed and others (2023). An integrated mediapipe-optimized gru model for sign language recognition. *Engineering Applications of Artificial Intelligence*, 117.
- [Ba et al., 2016] Ba, J. L., Kiros, J. R., y Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.
- [Bengio et al., 1994] Bengio, Y., Simard, P., y Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.
- [Benítez Andrade y Catril Carrillo, 2022] Benítez Andrade, L. I. y Catril Carrillo, R. O. (2022). Reconocimiento en tiempo real del alfabeto de lengua de señas chilena empleando aprendizaje automático. Memoria (Ingeniero Civil en Informática).
- [Cho et al., 2014] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., y Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.
- [Group, 2023] Group, E. A. R. (2023). Optimizing tflite models for on-edge machine learning for efficiency. *Journal of Mobile Computing*.
- [Hochreiter y Schmidhuber, 1997] Hochreiter, S. y Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.
- [Ioffe y Szegedy, 2015] Ioffe, S. y Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. En *Proceedings of the 32nd International Conference on Machine Learning*, pp. 448–456. PMLR.
- [Juego Educa, 2025] Juego Educa (2025). Palabras en lengua de señas chilena (Isch). Consultado: 2025-01-10.
- [La Tercera, 2023] La Tercera (2023). Falta de inclusión, estigma social y discriminación: la compleja realidad diaria que vive una persona sorda en Chile.
- [Meta Platforms, Inc., 2015] Meta Platforms, Inc. (2015). React native: A framework for building native apps using react.
- [MINISTERIO DE DESARROLLO SOCIAL Y FAMILIA, 2021] MINISTERIO DE DESARROLLO SOCIAL Y FAMILIA (2021). Ley 21303. Ley que promueve el uso de lengua de señas.

- [Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, New York.
- [Observatorio Social, 2022] Observatorio Social (2022). Encuesta de discapacidad y dependencia 2022.
- [Pigou et al., 2016] Pigou, L., Van den Oord, A., Dieleman, S., Van Herreweghe, M., y Dambre, J. (2016). Beyond temporal pooling: Recurrence and temporal convolutions for gesture recognition in video. En *International Journal of Computer Vision*, volumen 119, pp. 32–47. Springer.
- [Schuster y Paliwal, 1997] Schuster, M. y Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.
- [Servicio Nacional de la Discapacidad (SENADIS), 2022] Servicio Nacional de la Discapacidad (SENADIS) (2022). Iii estudio nacional de la discapacidad (endisc 2022). Technical report, Ministerio de Desarrollo Social y Familia, Chile.
- [Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., y Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.
- [Yáñez y Aguilera, 2025] Yáñez, M. y Aguilera, A. (2025). Reconocimiento aislado de la lengua de señas chilena en tiempo real usando deep learning. *Ingeniare. Revista chilena de ingeniería*, 33.
- [Zhang et al., 2020] Zhang, F., Bazarevsky, V., Vakunov, A., Tkachenka, A., Sung, G., Chang, C.-L., y Grundmann, M. (2020). Mediapipe hands: On-device real-time hand tracking. En *arXiv preprint arXiv:2006.10214*.
- [Zhang et al., 2018] Zhang, H., Cisse, M., Dauphin, Y. N., y Lopez-Paz, D. (2018). mixup: Beyond empirical risk minimization. En *International Conference on Learning Representations*.