PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

CONVERSIÓN Y EVALUACIÓN DE UN MODELO DE APRENDIZAJE PROFUNDO TRADUCTOR DE LENGUA DE SEÑAS PARA EDGE COMPUTING

Tesis para obtener el título profesional de Ingeniero Electrónico

AUTOR:

Mauricio Dario Salazar Espinosa

ASESOR:

Dra. Gissella María Bejarano Nicho

Lima, noviembre 2025

Informe de Similitud

Yo, **Gissella María Bejarano Nicho**, docente de la Facultad de **Ciencias e Ingeniería** de la Pontificia

Universidad Católica del Perú, asesor(a) de la tesis/el trabajo de investigación titulado **"CONVERSIÓN Y EVALUACIÓN DE UN MODELO DE APRENDIZAJE PROFUNDO TRADUCTOR DE LENGUA DE SEÑAS PARA EDGE COMPUTING"** del autor: **Mauricio Dario Salazar Espinosa**.

dejo constancia de lo siguiente:

- El mencionado documento tiene un índice de puntuación de similitud de **12%**. Así lo consigna el reporte de similitud emitido por el software *Turnitin* el 17/11/2025.
- He revisado con detalle dicho reporte y la Tesis o Trabajo de Suficiencia Profesional, y no se advierte indicios de plagio.
- Las citas a otros autores y sus respectivas referencias cumplen con las pautas académicas.

Lugar y fecha: **Lima, 17 de Noviembre del 2025**

Apellidos y nombres del asesor / de la asesora: Bejarano Nicho, Gissella María	
DNI: 43653710	Firma
ORCID: https://orcid.org/0000-0002-5624-8553	

A mis padres, Dario y Martha, por ser mis guías
con sus principios y por su amor incondicional.
Por el esfuerzo que realizaron para darme la
oportunidad de completar mi carrera.

A mi hermanita, Valentina, quién me llena de
orgullo y con quién siempre he podido contar
con cariño y risas cuando lo necesito.

A mi novia, Brenda, cuyo apoyo lleno de amor
a lo largo de este proyecto y toda la carrera
ha sido invaluable para superar todas
las dificultades en el camino.

Resumen

Actualmente, los diccionarios en línea para lenguas de señas que utilizan aprendizaje profundo dependen generalmente de servidores externos para realizar las operaciones de reconocimiento de señas. El diccionario de lengua de señas peruana no es la excepción, lo que implica limitaciones en términos de eficiencia y autonomía. Se propone eliminar esta dependencia a un servidor externo llevando el procesamiento al navegador del usuario. En este sentido, la presente tesis tiene como objetivo central adaptar un modelo de aprendizaje profundo para reconocimiento de lengua de señas a un enfoque de*edge computing*para inferencia desde navegador.

El trabajo inicia con la definición de los fundamentos necesarios, incluyendo los modelos de estimación de pose y la arquitectura encoder-decoder utilizada para el reconocimiento de señas. También se detallan las técnicas de compresión evaluadas, la cuantización y la destilación de conocimiento, aplicadas para optimizar el modelo para su despliegue en el navegador. Con esta base, se desarrolla la metodología que abarca el uso de la base de datos AEC-DGI305, el entrenamiento del modelo y su integración en un entorno de navegador mediante el*framework*React.

Los resultados obtenidos muestran que el modelo alcanza una precisión máxima Top-1 de 72.3 % y una precisión Top-5 de 89.3 %. Además, se encuentra que aplicar cuantización de 8 bits es más efectivo en reducir el tamaño de los modelos sin afectar su precisión significativamente. Asimismo, se valida que al eliminar la dependencia a un servidor externo el tiempo de inferencia se reduce hasta 4.28 veces.

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

MARCO PROBLEMÁTICO

1.1. La discapacidad auditiva en el Perú

En el Perú, según el último censo nacional, se estima que más de 232 mil personas enfrentan dificultades para la audición [4]. Dentro de este grupo, la ciudad de Lima alberga la mayor cantidad de habitantes con esta dificultad, con aproximadamente 74.5 mil personas, seguida por La Libertad con 13.4 mil, Arequipa con 13 mil y Puno con 12.5 mil. Estas cifras revelan una realidad significativa que demanda atención: la necesidad de mejorar la accesibilidad y brindar facilidades a este segmento de la población.

Como lo señalan E. Parks y J. Parks en su encuesta sociolingüística a la comunidad sorda, existe una percepción generalizada de que el gobierno proporciona una ayuda mínima para garantizar su bienestar [5]. Esta percepción se refleja en diversas métricas recopiladas en la encuesta. Donde, por ejemplo, el 41.1 % indicó que no recibía la educación necesaria para encontrar empleo y el resto indico que recibía una mínima preparación pero no suficiente. Más aun, la encuesta también encontró que no existe un entrenamiento formal para ser interprete dado que en el Perú no es considerado una profesión. Debido a ello, no existe una disponibilidad accesible a este apoyo con interpretes. Considerando que más de la mitad de estos interpretes operan en Lima, la mayoría de habitantes con discapacidad auditiva no pueden solicitar la ayuda de un interprete con facilidad. La solución a la que se tiende a recurrir es pedirle ayuda a amigos que los puedan ayudar durante actividades cotidianas tales como clases escolares, reuniones, revisiones médicas, entre otros [5]. Si bien esta asistencia informal les permite enfrentar estas situaciones, este tipo de apoyo es crítico y debería ser proporcionado de manera más sistemática y estructurada por parte de las instituciones pertinentes. La dependencia de la ayuda de amigos o conocidos para actividades básicas de la vida diaria pone de relieve la falta de accesibilidad y

apoyo adecuados para la comunidad sorda en la sociedad peruana actual.

La carencia de consideración y respaldo en diversos aspectos de la vida cotidiana de esta comunidad subraya la urgencia de desarrollar herramientas y recursos que faciliten sus actividades del día a día. Esta falta de apoyo no solo representa una barrera para su participación activa en la comunidad, sino que también limita su acceso a oportunidades educativas, laborales y sociales. Al desarrollar herramientas que aborden estas necesidades, podemos eliminar las barreras existentes en la actualidad para formar una sociedad más inclusiva y equitativa.

1.2. Diccionarios para Lengua de Señas

Los diccionarios son herramientas fundamentales para la difusión y el aprendizaje de cualquier lengua, ya que facilitan la búsqueda de equivalencias y simplifican su comprensión, especialmente durante el proceso de aprendizaje. Sin embargo, la implementación de diccionarios para la lengua de señas presenta desafíos únicos debido a la naturaleza visual y gestual de esta forma de comunicación. A diferencia de los diccionarios de lenguaje escrito, donde las equivalencias se establecen entre palabras y definiciones escritas, en el caso de la lengua de señas, las equivalencias se presentan entre vídeos y texto, lo que complica el proceso de búsqueda y consulta.

Ante esta dificultad, surgen alternativas que emplean modelos de aprendizaje profundo para clasificar y buscar equivalencias entre señas automáticamente y de manerafluida. Estos modelos se basan en modelos enfocados en reconocimiento de patrones para identificar y asociar las señas con sus correspondientes traducciones, permitiendo una búsqueda más intuitiva y eficiente en los diccionarios de lengua de señas. Esto se ha visto implementado entre el*American Sign Language* (ASL) e inglés[1] [6] y entre la lengua de señas francesa/belga y francés[2][7]. Ambas implementaciones están disponibles en línea sin costo alguno con su uso.

Un ejemplo particularmente relevante resulta el despliegue de un diccionario en línea de uso libre para la traducción de lengua de señas peruana (LSP) hacia castellano y vice versa[3][8]. Este diccionario en línea ofrece un servicio de traducción español-LSP con 100 palabras y reconocimiento de 38 señas para traducción LSP-español. Permite a los usuarios grabarse para consultar la traducción de la seña que están realizando. El servicio es gratuito y no requiere registro. Además, permite traducir palabras escritas a su representación en LSP, facilitando el

[1] https://research.sign.mt/
[2] https://dico.corpus-lsfb.be/
[3] https://diccionariolsp.pucp.edu.pe/search-by-sign

aprendizaje de la lengua para los usuarios interesados. La base de datos incluye 517 palabras para búsqueda por texto.

Figura 1.1: Interfaz de uso para diccionario LSP a español.

Un detallefinal importante respecto a los 3 diccionarios mencionados es que todos usan el mismo modelo base de arquitectura SPOTER [3]. Esta arquitectura esta basada en la arquitectura de transformers [2] con un*encoder*y*decoder*que reciben vectores de puntos relevante de manos/rostro y retornan la predicción de la palabra señalizada. Se explicará más a detalle en el capítulo 2 abordando el fundamento técnico, no obstante, resulta significativo mencionarlo como el modelo base más popular de estas variaciones para el despliegue de diccionarios en línea.

1.3. Estado actual y problemática de despliegue de modelos

Los últimos avances en el desarrollo de traducción de lengua de señas se han basado en la utilización de PyTorch debido a la versatilidad que esta librería ofrece en términos de configuraciones y diseño de modelos [9, 10, 11, 3]. Esta librería permite guardar los pesos (parámetros) de modelos entrenados junto con su arquitectura en un archivo .pth que se puede compartir para inicializar y correr el modelo en otro computador. Haciendo uso de este archivo, el flujo usual para el despliegue web de modelos suele emplear un servidor de procesamiento donde se carga el archivo .pth y se realiza la inferencia del modelo ejecutándolo con Python. Este servidor se encargará del cálculo de todas las operaciones que requiera el modelo para realizar sus predicciones. En este proceso, la inferencia hace referencia a los cálculos de predicciones que el modelo ejecuta a partir de los datos de entrada que reciba. En la actualidad, el requisito de un servidor externo puede satisfacerse mediante servicios como AWS SageMaker o Azure ML, que proporcionan servidores de procesamiento de inferencia listos para usar. En este contexto, los datos capturados y preprocesados por la interfaz web del navegador del usuario viajan desde su

navegador hasta este servidor externo, donde se utilizan como entrada para el modelo, y el servidor responde con el resultado de la inferencia del modelo. En 1.3 se observa la arquitectura de despliegue que se uso en el diccionario de lengua de señas francesa/belga a francés para cargar el modelo [7]. Para comenzar, Mediapipe es usado para hacer un procesamiento inicial del video cargado por el usuario desde la web. Posteriormente, los datos se envían a una API ubicada en un servidor externo, donde se realizan los cálculos correspondientes para la inferencia del modelo, y se devuelve únicamente la palabra traducida como respuesta. Por el momento, solo se presenta como una etapa de procesamiento inicial para el vídeo. En el capítulo 2 se explicará en mayor detalle el rol de modelo de estimación de pose que desempeña Mediapipe.

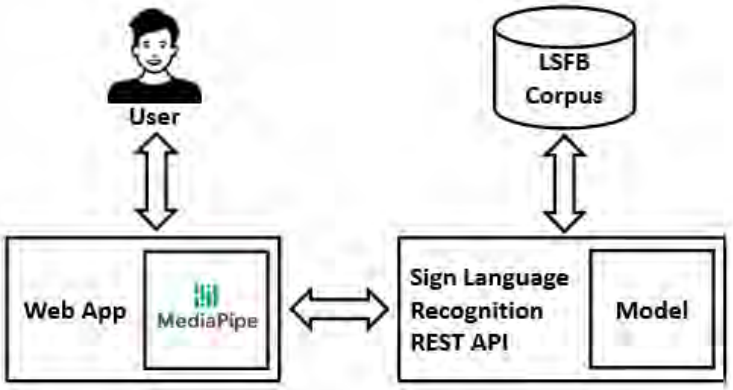

Figura 1.2: Arquitectura de despliegue para diccionario francés de Fink et al.

Aunque esta metodología de implementación es funcional, conlleva un gasto recurrente en el mantenimiento del servidor de procesamiento para la inferencia. Más aún, este gasto aumentaría significativamente a medida que aumenta el número de usuarios que utilizan el servicio de traducción. Los servicios de procesamiento a través de servidores externo cobran por la capacidad computacional que se ocupa, por lo que un incremento en las solicitudes de inferencia debido a más usuarios será equivalente a un aumento proporcional en costo. Este aumento en los costos puede resultar un obstáculo para la sostenibilidad a largo plazo del proyecto considerando que se busca mantener su uso gratuito. Incluso en casos de organizaciones dedicadas a brindar estos servicios a gran escala, este factor resulta un gasto significativo. Por ejemplo, para un proyecto desarrollado en base al modelo Yolov8 para reconocimiento de objetos se alcanzaron costos oscilando entre los $12.87 hasta los $46.92 diarios considerando un rango de hasta 118,000 imágenes solicitadas [12]. Para reconocimiento de lengua de señas, estas solicitudes no estarán enviando imágenes aisladas sino que secuencias de imágenes en videos, por lo que los costos serían aún más elevados.

Por otro lado, otro desafío de esteflujo de operaciones es la necesidad de transferir los datos a un servidor externo. Este proceso requiere subir los datos de video preprocesados por Mediapipe a internet para cada seña que se desee traducir. Dada la cantidad de información que esto puede representar debido a la longitud variable de los vídeos, se necesita una conexión estable a internet para garantizar una transferencia exitosa. No obstante, en el Perú, solo el 41.7 % de los hogares tienen acceso a internetfijo, mientras que el 89.5 % utilizan internet móvil [13]. Por este factor, su acceso a internet se ve limitado por la cantidad de datos móviles con los que cuenten y la capacidad de conexión de sus celulares.

1.4. Edge Computing

Considerando entonces los gastos de mantenimiento por procesamiento en la nube, surgen las alternativas enfocadas en el*Edge Computing*, un área que se enfoca en llevar el procesamiento de datos y la ejecución de aplicaciones directamente al lugar donde se encuentran los usuarios. En este enfoque, el dispositivo del usuario se convierte en la plataforma para realizar los cálculos necesarios para las inferencias del modelo de aprendizaje profundo. Este enfoque tiene el beneficio de eliminar la necesidad de depender de un servidor externo para procesar todas las operaciones, lo que resulta en una reducción de costos y una simplificación significativa de la arquitectura de despliegue. El modelo puede ser entrenado en un computador con determinadas características y luego ser desplegado con ajustes para que pueda realizar sus predicciones desde el dispositivo del usuario.

Además, esta estrategia soluciona el problema anteriormente descrito de una conexión estable a internet. No requiere de consumir banda de internet subiendo vídeos del usuario preprocesados localmente, sino que los utiliza localmente sin que salgan del dispositivo. Del mismo modo, proporciona una ventaja adicional de mejorar la privacidad del usuario, ya que sus datos no necesitan ser transferidos a un servidor externo para su procesamiento. En cambio, toda la computación se lleva a cabo en el propio dispositivo del usuario, lo que reduce el riesgo de exposición de datos sensibles. Estofinalmente resulta en una mejora de protección de la privacidad y la seguridad de los datos del usuario.

El dispositivo del usuario puede ser cualquier dispositivo que utilice para acceder al modelo: su navegador web, una app en su teléfono móvil o incluso un dispositivo especializado diseñado específicamente para esta tarea. En el contexto de esta investigación, se evaluará la viabilidad de utilizar el navegador del usuario para realizar estas tareas, prescindiendo así de la necesidad de un

servidor externo para el procesamiento de datos.

Sin embargo, en este punto es donde los modelos más modernos presentan una deficiencia para su implementación en entornos de*Edge Computing*. Esto se debe a que el formato .pth de PyTorch no es directamente compatible con los métodos de implementación actuales de este método. Para abordar esta limitación, es necesario realizar una conversión del modelo a otros formatos más adecuados que se explicarán en las siguientes secciones.

1.4.1. Inferencia desde el navegador

La integración de la inferencia de modelos de aprendizaje profundo con*edge computing* directamente a través de los navegadores de los usuarios ha sido un área de interés y exploración en los últimos años. Varias librerías y tecnologías han surgido para abordar este desafío, entre las que se destacan TensorFlow.js [14] y ONNX Runtime [15]. Estas bibliotecas se basan en WebAssembly [16], el cual es un método de compilar eficientemente código en el mismo navegador. También es posible utilizar WebGL, otro método de compilación, para ejecutar la inferencia apoyándose del gpu del equipo. En el caso de ONNX Runtime, su repositorio indica que ambos métodos están disponibles para Chrome/Edge de Windows, Android, iOS y MacOS, Safari de iOS y MacOS, y Firefox de Windows. Los detalles técnicos de estas implementaciones serán explicados más a detalle en el capítulo 2, sin embargo, por el momento se introduce su alcance y aplicaciones. En este sentido, representan una forma viable de trasladar modelos de aprendizaje profundo al navegador mismo del usuario. Al analizar el efect

Un ejemplo destacado es la implementación de Segment Anything de Meta a través de ONNX Runtime [17, 15]. Esta tecnología utiliza una arquitectura*encoder-decoder*variación de la arquitectura de transformers [2] para la segmentación de imágenes, lo que permite identificar y separar diferentes objetos en una imagen con precisión y eficiencia. En la instancia en cuestión, se comenzó comprimiendo el modelo original para bajar su demanda computacional a través de cuantización y luego se pasó a convertirlo a un formato .onnx para su inferencia en el navegador. Con esta configuración, se lograron tiempos de ejecución de aproximadamente 45 segundos, donde el*encoder*toma 45 segundos y el*decoder*200 ms al ejecutarse con WebAssembly.

1.4.2. Limitaciones computacionales

No obstante, el*edge computing*conlleva una desventaja inherente debido a dónde se realiza el procesamiento: el dispositivo del usuario debe llevar a cabo la carga de procesamiento de inferencia. En casos de modelos complejos, esta carga puede ser muy intensa para ser realizada

Figura 1.3: Inferencia de modelo Segment Anything en navegador.

con facilidad. Como se puede observar en el caso de la implementación del modelo de segmentación mencionado anteriormente, esto puede tardar tiempos de hasta 45 segundos. Asimismo, si bien este enfoque elimina la necesidad de subir los datos a un servidor externo consumiendo internet, aún es necesario realizar una descarga inicial del modelo al navegador del usuario. Cada vez que un nuevo usuario ingrese a la página, su navegador tendrá que descargar el modelo. Por lo tanto, es esencial comprimir al máximo el peso de los modelos a utilizar para reducir esta demanda sobre la conectividad de los usuarios.

Para ello se consideran varias técnicas de compresión para modelos. Entre estas se encuentra cuantizar los pesos del modelo y realizar destilación de conocimiento para conseguir modelos más pequeños a partir de uno ya entrenado. Se explicará más a profundidad acerca de los principios de cada técnica en el siguiente capítulo, por lo que por el momento solo se explica como han sido aplicadas para ciertos ejemplos.

En el caso de cuantización, existen también estudios previos analizando su efecto sobre la arquitectura de transformers. Como ejemplo de ello, un estudio del 2020 verificó su efectividad en aplicar cuantización sobre variaciones de un modelo transformers con solo 2 capas de *encoders*para medir el impacto posible. En base al análisis realizado durante el estudio, se encontró que es posible pasar un modelo de peso original de 243 Mb a uno de hasta 31.57 Mb sin pérdida de precisión significativa con cuantización de 4 bits [18]. En otro caso, se aplicó la técnica de destilación de conocimiento sobre el modelo BERT para comprimirlo [19]. De esta forma, se logro entrenar un modelo más pequeño DistilBERT con capacidades similares de inferencia partiendo del modelo BERT original compuesto de un*encoder*sin*decoder*. En

específico, estos ajustes lograron una reducción de tamaño del 40 % y aumento de velocidad de inferencia de 60 % manteniendo el 97 % de su precisión original.

1.5. Justi□cación para desarrollar la tesis

Según encontraron E. Parks y J. Parks, existen variaciones entre la lengua de señas manejada por la comunidad sorda peruana dependiendo de la afiliación religiosa, ubicación, edad, entre otros factores [5]. Sin embargo, luego de realizar una prueba de texto grabado (PTG) para evaluar estas variaciones, se encontró que esta variación era mínima regionalmente, sugiriendo que LSP era suficientemente estandarizado acorde al estudio en cuestión. Considerando entonces que un modelo especializado en LSP sería útil a través del Perú, se eligió esta lengua para el trabajo de tesis.

Asimismo, como también se mencionó previamente, el estado del arte de los diccionarios en línea para lenguas de señas indica que la arquitectura SPOTER [3] es un buen camino como punto de partida. Su diseño con etapa de encoder y decoder es útil para la traducción de lengua de señas, así como para para una variedad de aplicaciones adicionales en visión por computadora así como para procesamiento de lenguaje natural al ajustarse ligeramente. Si bien la arquitectura de SPOTER especifica es ligera, está igualmente puede resultar demandante para inferencia en dispositivos no especializados tal como teléfonos móviles o navegadores. Como se vio en casos anteriormente mencionados, ciertos modelos pueden tardar hasta 45 segundos en ejecutar una sola inferencia al usar el navegador web. Por este motivo, investigar como comprimir e implementar adecuadamente la arquitectura de un modelo encoder/decoder similar al SPOTER [3] ayudará a entender cómo aplicar la misma secuencia de pasos para otros modelos con arquitecturas similares que puedan tener distintas aplicaciones para agilizar su despliegue accesible a través de*edge computing*.

Finalmente y como punto central, la inferencia desde navegador para este modelo en particular ayudará a volver más viable la accesibilidad al diccionario entre LSP-castellano a un mayor número de usuarios. Esta ventaja abordará la actual falta de interpretes de LSP identificada [5].

1.6. Objetivos

En síntesis, se facilitará la accesibilidad a un modelo con capacidad de interpretación aislada de lengua de señas peruana a nivel de señas individuales a través de la inferencia en el navegador mismo. Los objetivos para alcanzar esta propuesta se plantean a continuación

1.6.1. Objetivo General

Adaptar y evaluar para*edge computing*un modelo de aprendizaje profundo encoder-decoder para reconocimiento de lengua de señas midiendo su desempeño en navegadores web con capacidad mínima de identificar 38 palabras señalizadas.

1.6.2. Objetivos Específicos

Los objetivos específicos delimitaran el progreso del trabajo como una guia para el desarrollo.

- Entrenar modelos de aprendizaje profundo con arquitectura encoder-decoder para la clasificación de señas aisladas en lengua de señas peruana.
- Comprimir modelos evaluando las técnicas de cuantización y destilación de conocimiento.
- Comparar variaciones en peso, precisión y tiempos de inferencia para distintos métodos de conversión de modelos a formato para*edge computing*.
- Desarrollar un sistema de validación que permita evaluar la inferencia del modelo en navegadores web de escritorio.

Capítulo 2

FUNDAMENTOS TEÓRICOS

Previo al desarrollo de los objetivos, resulta importante definir los fundamentos de las herramientas que se utilizarán. Por ello, el presente capítulo abarca las explicaciones de la arquitectura del modelo, las técnicas de compresión que se utilizarán, y el entorno de ejecución para inferencia desde navegador.

2.1. Modelos de estimación de pose

La arquitectura del modelo de aprendizaje profundo seleccionado requiere de un preprocesamiento inicial de los datos de vídeo grabados. En lugar de ingresar directamente la secuencia de *frame* RGB de cada vídeo, el modelo toma como entrada a vectores de posición de partes del cuerpo específicas extraídos de cada *frame*. Para este primer paso de procesamiento, es necesaria la aplicación de un modelo de estimación de pose que brinde las coordenadas de dichos puntos, llamados generalmente *keypoints*.

La estimación de pose es una técnica utilizada para localizar las coordenadas de puntos clave en el cuerpo humano dentro de una imagen o un vídeo. Estos puntos clave tienden a corresponder a articulaciones, secciones de manos, o secciones del rostro.

En Fig. 2.1, se presenta una visualización de los *keypoints* extraídos por el modelo de estimación de pose Blazepose, el cual es usado por Mediapipe. Mediapipe es un *framework* desarrollado que permite la detección y seguimiento en tiempo real de puntos clave del cuerpo humano, incluyendo las manos y la cara [20]. En partícular, Mediapipe Hollistic combina tres modelos de estimación de pose para conseguir un total de 543 *keypoints* para cada *frame* procesado. Los modelos que combina son:

- BlazePose: Este modelo se encarga de detectar puntos clave del cuerpo humano. Es capaz

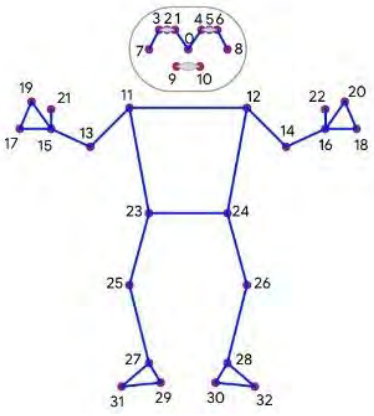

Figura 2.1: Keypoints de pose detectados por Blazepose [1].

de identificar y rastrea 33 puntos de referencia del cuerpo, tales como las posiciones de las articulaciones y los segmentos del cuerpo.

- BlazeFace: Este modelo se especializa en detectar y mapear puntos clave del rostro. El modelo proporciona una malla detallada de 468 puntos de referencia faciales, capturando una gran cantidad de información sobre la estructura y la expresión facial.
- MediaPipe Hands: Este modelo se centra en las manos, detectando y rastreando 21 puntos de referencia por mano. Es capaz de capturar movimientos y gestos detallados de las manos, lo cual es crucial para aplicaciones que requieren un análisis preciso de estos segmentos.

Si bien el total de*keypoints*extraidosfinalmente resulta en 543, no se usan todos para el reconocimiento de señas. Debido a la complejidad y al volumen de datos generados por Mediapipe Holistic, el paper original de SPOTER detalla que se realizo un paso defiltrado inicial para obtener los puntos más relevantes para la tarea específica de reconocimiento de lenguaje de señas. En este caso, se seleccionan así 54 puntos de referencia clave y eliminando el resto. Esta selección se debe al modelo actual desplegado en`https://diccionariolsp.pucp.edu.pe/`, donde se utilizan 54*keypoints*de los 543 totales disponibles a través de Mediapipe Holistic [8].

2.2. Arquitectura del modelo

A continuación, se explican los componentes de la arquitectura del modelo de aprendizaje profundo que se utilizará para el reconocimiento de señas individuales.

2.2.1. Transformers

La arquitectura de transformers fue propuesta inicialmente en 2016, donde se planteó un modelo basado puramente en el concepto de autoatención para traducción de lenguajes [2]. El principal valor de esta propuesta recae en el uso de autoatención para capturar las dependencias que existen en secuencias de datos, dado que previamente se usaban herramientas tales como RNNs o convoluciones para estefin. Si bien esta arquitectura fue diseñada originalmente para procesamiento de lenguaje natural, iteraciones posteriores demostraron su efectividad para el campo de visión por computadora. Suflujo de operación constituye principalmente de tres etapas principales:*embedding*,*encoder*y*decoder*.

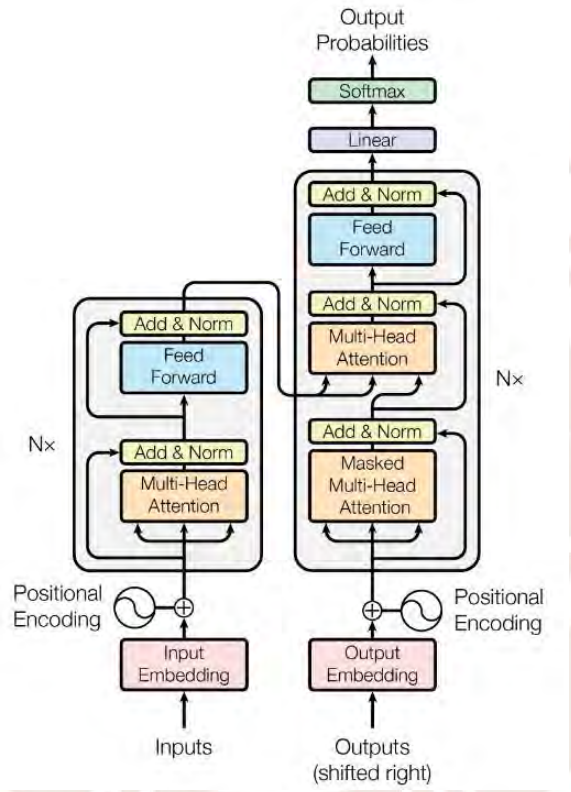

Figura 2.2: Arquitectura de transformers [2].

En un primer lugar, el paso de*embedding*define un método a través del cual convertir el dato inicial de entrada en un vector que pueda ser procesado (referido como*embedding*). En el planteamiento inicial de la arquitectura, este fue aplicado para transformar palabras en oraciones en vectores. Sin embargo, en el caso de la arquitectura SPOTER [3] que se utilizará para el trabajo, diseñada específicamente para lenguas de señas, esta etapa se reemplaza por un preprocesamiento basado en un modelo de estimación de pose. Este modelo transforma los videos en coordenadas de puntos clave, capturando información valiosa sobre posiciones corporales necesaria para la

comprensión del lenguaje de señas.

Seguidamente, se pasan estos *embeddings* a través del *encoder*, donde se codifican en una representación vectorial significativa que va aprendiendo el modelo para la tarea asignada. Esta etapa está compuesta por un codificador (*encoder*) que puede contener una cantidad N de capas que procesen secuencialmente los datos aprendiendo distintas características. Inicialmente se propusieron 6 capas del *encoder*, cada una con 2 sub-capas: la primera de autoatención seguida por una de *Feed Forward*[2]. Estas sub-capas cuentan con conexiones residuales, las cuales suman el resultado de la sub-capa con su vector de entrada y normalizan la suma total.

Finalmente se envía la salida de la última capa del *encoder* hacia el *decoder*, donde se encarga de predecir la inferencia deseada. En el *decoder* se incluyen 3 sub-capas principales: 2 de autoatención y 1 de *Feed Forward*. La primera capa de autoatención se encarga de relacionar las predicciones de los elementos anteriores de la secuencia con el elemento actual. Es decir, se apoya en las predicciones anteriores para inferir la predicción actual. Mientras tanto, la otra capa de autoatención relaciona la representación codificada por el *encoder* del dato actual con el resultado de la sub-capa anterior. A continuación se pasará el resultado de esta sub-capa por una *Feed Forward* final que luego de pasar por una capa lineal que nos brinda la predicción final al utilizar una función de activación *softmax*. Esta última función resulta importante de detallar dado que también forma parte del cálculo para el componente de autoatención de la arquitectura. En síntesis, *softmax* o σ normaliza la salida para generar una distribución de probabilidad sobre a qué clase corresponde el dato ingresado a través de 2.1.

$$Softmax(z_i) = \sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \tag{2.1}$$

El vector resultante $z = [z_1, z_2, ..., z_N]$ de un modelo de aprendizaje profundo es transformado por la función *softmax* a $\sigma(z) = [\sigma(z_1), \sigma(z_2), ..., \sigma(z_N)]$. En este nuevo vector, cada elemento $\sigma(z_i)$ representa la probabilidad de que el dato de entrada pertenezca a la clase i. Para ello, z_i se toma como exponente de e y se lo normaliza dividiéndolo entre la sumatoria de todos los elementos de z tomados como exponentes para poder interpretarlo como una probabilidad entre 0 y 1.

2.2.2. Autoatención

El componente central de la arquitecutra de *transformers* explicada es el mecanismo de autoatención, que permite al modelo centrarse en diferentes partes de la entrada al procesar cada elemento. De esta forma, puede relacionar entre sí a distintas partes del vector de datos ingresado

para evaluar cada uno con un mayor contexto. Como se mencionó previamente, este fue el principal aporte de la arquitectura, resultando más eficiente y eficaz que las opciones anteriores de RNNs y convoluciones que usaban los modelos*encoder-decoder*previos. Con esto, se reemplazaron arquitectuas previas que empleaban otras herramientas como redes neuronales convolucionales [21] o LSTMs en el*encoder*y*decoder*con una capa de atención entre ellos calculada con una capa*Feed Forward*de 1 dimensión [22]. A diferencia de estas implementaciones, la autoatención es capaz de conectar los datos en todas las posiciones entre si en una cantidadfija de operaciones$O(1)$con una complejidad por capa de$O(n^2 \cdot d)$, donde$n$es la longitud de secuencias y$d$la cantidad de dimensiones. Comparando con RNNs, donde las operaciones entre datos en una secuencia pueden tener una distancia de$O(n)$y posee una complejidad computacional de$O(n \cdot d^2)$, resulta más efectiva. La cantidad dimensiones$d$tiende a ser mayor a la longitud de las secuencias evaluadas$n$, por lo quefinalmente la complejidad de la autoatención resulta menor ($O(n^2 \cdot d) < O(n \cdot d^2)$). Todas estas métricas fueron evaluadas y presentadas en la publicación que proponía la arquitectura en cuestión [2].

En referencia a las operaciones en sí que realiza estas capas de autoatención, se presentan tres componentes principales: el*query*(Q), la llave (K) y el valor (V). El proceso determina cómo el *query*se relaciona con la llave y escala el valor proporcionalmente a esa relación. Se comienza así tomando el producto punto de los*queries*Qy llavesK^Tpara calcular la similitud entre estos, el resultado escalado pord_k^{-1}se pasa por una función*softmax*que normaliza los valores en una distribución de probabilidad entre qué llaves corresponden más con qué*queries*. De esta forma, puede encontrar que*queries*corresponden con mayor seguridad a una determinada llave. Esta distribución de probabilidad se multiplica con los valores$V$para asignarle un peso correspondiente con su relevancia con el$Q$ingresado.

$$Autoatencion(Q, K, V) = \sigma\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V \quad (2.2)$$

Esta operación es calculada en paralelo sobre el mismo conjunto de$Q$,KyVcon sus respectivas matrices de pesos para cada capa paralela. El resultado de estas capas paralelas (conocidas como cabezas de atención) es concatenado y multiplicado por otra matriz de pesos final, lo que resulta en que cada cabeza pueda aprender diversas relaciones entre el dato inicial aprovechables por la salidafinal.

$$MultiHead(Q, K, V) = Concat(h_1, \ldots, h_n) W^O \quad (2.3)$$

$$h_i = \sigma\left(\frac{QW_i^Q \cdot (KW_i^K)^T}{\sqrt{d_k}}\right) \cdot VW_i^V \quad (2.4)$$

Para el trabajo a realizar, se utiliza la implementación de autoatención proporcionada por el paquete de Pytorch. Sin embargo, se evaluará la viabilidad de este procesamiento y ajustará acorde a las pruebas.

En el caso del modelo seleccionado, este componente de autoatención resulta útil para capturar la relación temporal que existe entre las coordeanadas de *keypoints* al hacer el gesto de alguna seña. A partir de ello, el cambio de ubicación de los *keypoints* durante distintos *frames* del vídeo puede considerarse para la inferencia. Se evalúa las relaciones entre estos puntos clave extraídos por el modelo de estimación de pose en distintos tiempos, permitiendo que el modelo identifique patrones importantes que representan movimientos y gestos específicos. Este proceso es relevante debido a que los gestos involucran un movimiento secuencial para llevarse a cabo.

2.2.3. SPOTER

Para esta investigación, se seleccionó la arquitectura SPOTER propuesta por Matyáš Boháček y Marek Hrúz [3]. Esta arquitectura consiste en una variación del marco de trabajo transformer estándar. Sus entradas se extraen a través de un modelo estimador de postura y se pasan como puntos de referencia a través de 6 capas de *encoder* con 9 cabezas de atención. Después de eso, el decodificador procesa la consulta a través de otras 6 capas de *decoder*. Sin embargo, dado que el objetivo consultado es un solo elemento decodificado de la secuencia codificada anterior, las capas del *decoder* no utilizan autoatención para sus cálculos con los otros elementos de la secuencia, solo la utilizan para la conexión *encoder-decoder*. Este resultado se pasa finalmente a una capa lineal que clasifica el signo en el vídeo correspondientemente.

Este modelo fue elegido porque su arquitectura prioriza el bajo costo computacional, con menos parámetros y muchas menos operaciones de puntoflotante (FLOPs) para la inferencia que otros modelos de reconocimiento de lenguas de señas. Asimismo, es capaz de apoyarse del conocimiento de modelos de estimación de pose pre-entrenados con una cantidad de datos mayor a la disponible con las bases de datos de lenguas de señas. La arquitectura SPOTER está diseñada específicamente para tareas de reconocimiento de señas, donde la precisión y la eficiencia son cruciales. Al utilizar un modelo estimador de postura para extraer *keypoints* del cuerpo y luego procesarlos a través de múltiples capas de atención, SPOTER puede capturar de manera efectiva las características espaciales y temporales necesarias para identificar correctamente los gestos de

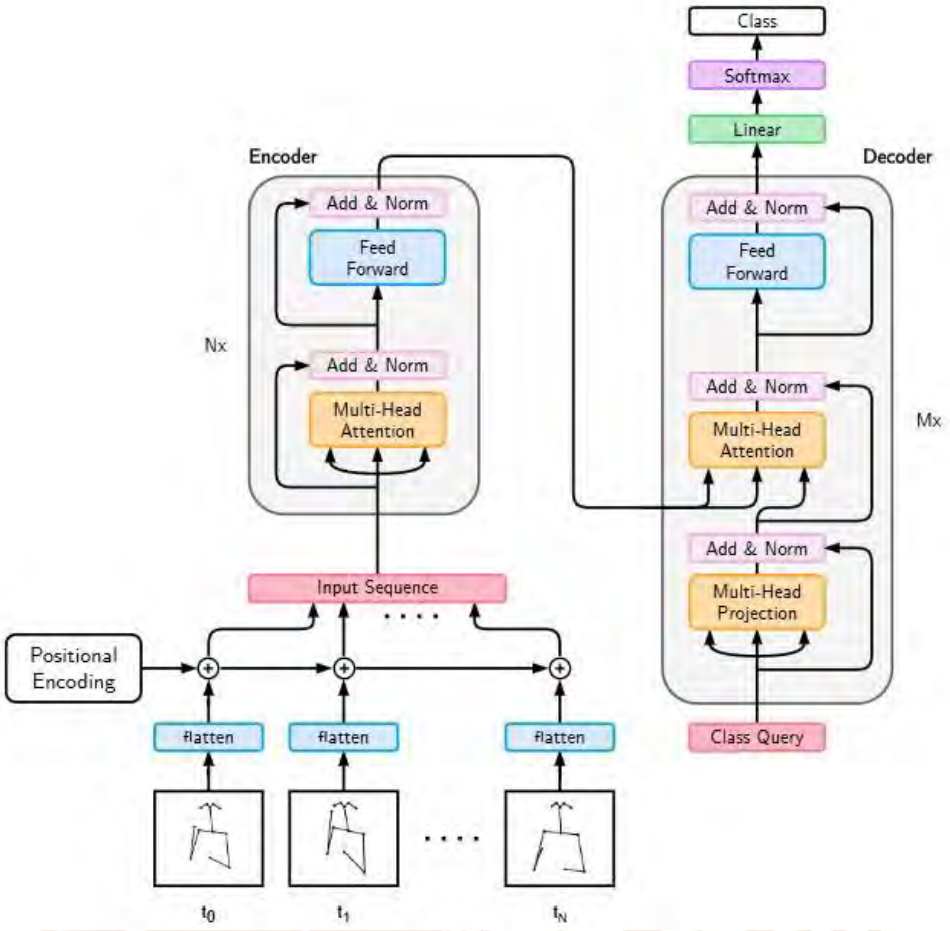

Figura 2.3: Arquitectura de modelo SPOTER [3].

la lengua de señas.

2.3. Técnicas de Compresión

Como se explico previamente en el capítulo 1, la inferencia a través de*edge computing*trae limitaciones sobre los recursos a los que el modelo tiene acceso. En este sentido, a continuación se presentan las principales herramientas que se consideran en el presente trabajo para combatir dichas limitaciones.

2.3.1. Cuantización de modelos

La cuantización es una técnica que reduce la precisión de los parámetros del modelo, comprimiendo los pesos originales de puntoflotante de 32 bits en pesos más pequeños de 8 bits, 4 bits o incluso 2 bits. Esta reducción disminuye significativamente el tamaño del modelo y los requisitos de cálculo, lo que lo hace ideal para la inferencia en dispositivos. Sin embargo, esto

puede conllevar a una reducción de precisión de inferencia en comparación con el modelo original. Esta técnica surgió de la noción de que los modelos tienden a estar sobreparametrizados, con una redundancia considerable entre sus pesos que resulta en un tamaño innecesario [23]. Ha sido exitosa en la reducción del tamaño de los modelos*transformers*smientras se mantienen sus puntuaciones de rendimiento al transformar sus pesos de puntoflotante de precisión completa de 32 bits en una forma reducida de enteros de 8 bits [24, 18].

Para el caso del presente trabajo, los métodos de cuantización disponibles en el entorno de *ONNX Runtime*que se utilizará resultan los más relevantes de explicar, dado que serán los utilizados para las pruebas. Este entorno proporciona únicamente la opción de realizar una cuantización de 8-bits (0 a 255) para los pesos del modelo [25]. En general, el mapeo de los modelos a su representación comprimida viene dado por la ecuación 2.5, donde el punto cero resulta el valor equivalente de 0 en fp32 para la escala cuantizada. El valor de la escala depende si se asume que los rangos de valores máximos de fp32 son simétricos o asimétricos alrededor de 0. Para el simétrico se asume que el valor absoluto de su valor máximo positivo y negativo es el mismo, se toma el máximo de los dos y la escala se calcula según la fórmula 2.6. Mientras tanto, en el caso de que se asuma son asimétricos con un rango positivo distinto al rango negativo, la escala se calcula según la fórmula 2.7.

$$val\_fp32 = escala \cdot (val\_cuantizado - punto\_cero) \quad (2.5)$$

$$escala_{simetrica} = \frac{max(abs(rango\_max),\ abs(rango\_min)) \cdot 2}{rango\_cuantizado\_max - rango\_cuantizado\_min} \quad (2.6)$$

$$escala_{asimetrica} = \frac{rango\_max - rango\_min}{rango\_cuantizado\_max - rango\_cuantizado\_min} \quad (2.7)$$

Estos parámetros de escala y punto cero pueden ser calculados una sola vez o cada vez que se realice una inferencia dependiendo del método de cuantización: estática o dinámica. Para la cuantización estática, se calibra una sola vez estos parámetros luego de pasar un conjunto de datos que le permitan encontrar al modelo los parámetros más efectivos. Mientras tanto, la cuantización dinámica calculará los parametros en cuestión para cada inferencia realizada a partir de los rangos de pesos que genere el dato ingresado. Esta segunda implementación proporciona una menor perdida de precisión al coste de un mayor tiempo por cada inferencia al tener que calcular los parámetros cada vez y es la usada durante las pruebas.

2.3.2. Destilación de modelos

La destilación de conocimiento es un enfoque para entrenar modelos más pequeños aprovechando la distribución de probabilidad de salida de un modelo más grande y complejo (referido como el modelo maestro). El modelo grande enseña al modelo más pequeño (referido como el modelo estudiante), permitiendo que el modelo estudiante logre un alto rendimiento mientras mantiene una arquitectura más eficiente y compacta.

El proceso de entrenamiento implica el uso de una función de pérdida que combina dos componentes: la*cross-entroppy loss*convencional entre las predicciones del modelo estudiante ($\sigma(z_e)$) y las etiquetas verdaderas (y), y la divergencia de Kullback-Leibler entre las salidas del estudiante ($\sigma(z_e)$) y del maestro ($\sigma(z_m)$) [26]. La*cross-entroppy loss*L_{CE} en (2.9) asegura que las predicciones del estudiante se alineen con las etiquetas verdaderas, mientras que la divergencia KLL_{KL} en (2.10) ayuda al estudiante a replicar los patrones del modelo maestro. Al equilibrar estos dos componentes de pérdida en la pérdida total de destilación de conocimiento L_{KD} en (2.8) con un factor de ponderaciónα, el modelo estudiante aprende a reproducir el modelo maestro, reduciendo su tamaño mientras mantiene su rendimiento.

$$L_{KD} = \alpha L_{CE}(y, \sigma(z_e)) + (1-\alpha) L_{KL}(\sigma(z_m / T), \sigma(z_e / T)) \quad (2.8)$$

$$L_{CE}(y, \sigma(z_e)) = -\sum_i y_i \log(\sigma(z_{ei})) \quad (2.9)$$

$$L_{KL}(\sigma(z_m / T), \sigma(z_e / T)) = \sum_i \sigma(z_{t_i} / T) \log \frac{\sigma(z_{t_i} / T)}{\sigma(z_{s_i} / T)} \quad (2.10)$$

Para esta función de pérdida, hay dos parámetros importantes: la temperatura (T) y un factor (α). La temperatura determina el suavizado de las distribuciones de probabilidad del modelo maestro. Cuando es alta, las probabilidades están más distribuidas, lo que ayuda al estudiante a aprender la diferencia entre todas las clases, mientras que se enfocará más en la clase más probable cuando T es baja. Mientras tanto,αdefine el equilibrio entre la divergencia KL y la pérdida de entropía cruzada. Siαes demasiado baja, el modelo podría sobre-ajustarse a las salidas del maestro y no aprender lo suficiente de las etiquetas verdaderas. Sin embargo, siαes demasiado alta, el estudiante podría no capturar suficiente del conocimiento destilado del maestro.

2.4. Onnx Runtime

Si bien el entrenamiento inicial de las variaciones del modelo se realiza a través de Pytorch debido a su versatilidad, el formato de despliegue final para la inferencia desde navegador será ONNX. Es decir, se deberá convertir el formato .pth original de Pytorch a formato .onnx de ONNX. Esto se debe a que ONNX cuenta con el entorno de *ONNX Runtime*, un framework enfocado en la interoperabilidad de modelos entre diferentes plataformas. El entorno de *ONNX Runtime* cuenta así con un modulo para Javascript que le permite compilar y ejecutar los modelos de manera eficiente en un entorno de navegador. Para estefin, puede apoyarse de varios métodos de compilación: WebAssembly, WebGPU, WebNN o WebGL. No obstante, para el presente trabajo se evaluarán únicamente WebAssembly dado que es el método que más disponibilidad tiene independientemente del equipo donde se acceda al navegador [25].

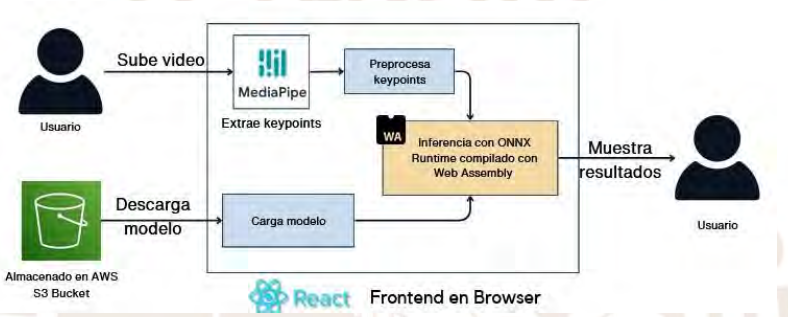

Figura 2.4: Proceso de inferencia en el navegador.

Acerca de ellos, WebAssembly es un formato binario de instrucciones que permite la ejecución eficiente de operaciones de aplicaciones web. Este método accede al CPU del equipo del usuario a través del navegador para realizar los cálculos. La Fig. 2.4 ilustra el proceso de despliegue e inferencia del modelo dentro de un navegador web utilizando WebAssembly y *ONNX Runtime* al utilizar el *framework* de React para el diseño del *frontend*.

Asimismo, *ONNX Runtime* también proporciona la opción de delimitar entornos reducidos que contengan únicamente los operadores usados por el modelo a procesar. De esta manera, es posible reducir la cantidad de elementos que requiera cargar la página cuando cargue para el usuario. Para que este entorno personalizado incluya únicamente aquellos operadores necesarios, se requiere convertir el modelo desde el formato .onnx a formato .ort.

Capítulo 3

METODOLOGÍA

Se determina llevar a cabo un análisis cuantitativo para evaluar el desempeño del modelo bajo múltiples condiciones. En primer lugar, variando la configuración del modelo al probar las distintas técnicas de compresión descritas anteriormente. En segundo lugar, evaluando su desempeño en distintos entornos de ejecución para su inferencia en navegador. De este modo, los resultados de estas pruebas proporcionarán evidencia sobre la utilidad y viabilidad de los métodos evaluados.

3.1. Base de datos

La base de datos utilizada para el entrenamiento del modelo se compone de videos señalizando distintas glosas en la Lengua de Señas Peruana (LSP). Esta base de datos, denominada AEC-DGI305, incluye 931 videos en total y permite entrenar el modelo para la identificación de 38 señas distintas. Estos videos fueron grabados como parte de un proyecto previo en la Pontificia Universidad Católica del Perú, por lo que están disponibles públicamente en su portal de datos abiertos En la base de datos se incluyen videos individuales de cada seña con su respectiva anotación. Esta anotación de la palabra representada es denominada la glosa, correspondiente a la clase que predice el modelo. Para el presente trabajo, se divide la base de datos en un conjunto de entrenamiento y validación con una proporción de 80/20 entre el tamaño de ambos conjuntos. De este modo, se obtiene un conjunto de 744 videos de entrenamiento y 187 videos de validación para el desarrollo de los modelos.

[1] https://datos.pucp.edu.pe/dataset.xhtml?persistentId=hdl:20.500.12534/JU4OLG

3.1.1. Preprocesamiento

El primer paso del procedimiento consiste en utilizar el modelo de estimación de pose Mediapipe Holistic para convertir los videos en un arreglo de vectores que representan los puntos clave del cuerpo y manos. Este arreglo de vectores constituye la entrada del siguiente modelo, el cual está diseñado para predecir la seña realizada a partir de la secuencia de puntos clave extraídos en cada*frame*del video. Sin embargo, antes de que estos vectores puedan ser utilizados como entrada, es necesario realizar un preprocesamiento adecuado de los datos.

En este estudio, el preprocesamiento sigue los lineamientos establecidos en el desarrollo del diccionario de lengua de señas peruana, presentado en [27]. En particular, se aborda la imputación de valores faltantes debido a que, en varios videos, solo se detecta una de las dos manos durante ciertos periodos. Para resolverlo, se imputan los puntos clave no identificados de una mano utilizando las coordenadas de las muñecas correspondientes durante el*frame*, asegurando así la continuidad de la información gestual en ambos lados.

Posteriormente, cada entrada de datos se estructuró en la forma [*frame*,*keypointID*, coordenadas]. Más específicamente, la forma de los vectores preprocesados se convierte en [eje dinámico, 54, 2]. Donde eje dinámico hace referencia a que tiene dimensiones variables, se busca seaflexible de modo que el modelo puede manejar videos de diferentes longitudes. La segunda dimensión corresponde a los 54*keypoints*seleccionados de los 543 disponibles, como se describió anteriormente. Finalmente, la última dimensión representa las coordenadas x e y de cada punto clave, indicando su posición en cada fotograma.

3.2. Entrenamiento de modelos SPOTER

Una vez preparados los datos, se llevó a cabo el entrenamiento de múltiples variantes del modelo SPOTER con el objetivo de evaluar su rendimiento y realizar pruebas de desempeño. Este procedimiento buscó explorar cómo las reducciones en la arquitectura del modelo afectan tanto la precisión como la eficiencia, permitiendo identificar un equilibrio óptimo entre rendimiento y costo computacional. En este contexto, se entrenaron modelos con diferentes configuraciones de hiperparámetros para determinar la combinación que maximice el desempeño en versiones comprimidas.

Para mantener la consistencia en las comparaciones, todos los modelos se entrenaron durante 900 épocas, con paciencia de 100 épocas, una tasa de aprendizaje de 0.001 y utilizando la función de pérdida*cross-entropy loss*. Inicialmente, se entrenaron tres configuraciones base del modelo

SPOTER, que sirvieron como referencia para evaluar el número óptimo de capas en el encoder y el decoder. La primera variante, denominada SPOTER 6C-32FF, representó la versión completa del modelo con 6 capas en el encoder y el decoder, y un tamaño de 32 unidades en las capas *feedforward*. La segunda variante, SPOTER 3C-32FF, redujo el número de capas a 3 en ambas partes, permitiendo observar cómo una simplificación moderada afecta el rendimiento. Por último, se entrenó la variante SPOTER 1C-32FF con una sola capa en el encoder y el decoder, explorando una reducción máxima en términos de arquitectura.

Tras identificar el modelo base con mejor desempeño, se realizaron experimentos adicionales ajustando el tamaño de las capas*feedforward*úbicadas dentro del encoder y decoder, entrenandolos por 1500 épocas esta vez. En esta fase, se entrenaron modelos con configuraciones de 2, 4, 8, 128, 512 y 2048 capas*feedforward*, manteniendo 1 capafija de encoder y de decoder acorde al mejor modelo identificado (SPOTER 1C). Estas variantes permitieron analizar cómo la complejidad de las capas internas influye en la precisión y velocidad de inferencia. La Tabla 3.1 resume las configuraciones evaluadas durante este proceso.

Tabla 3.1: Parámetros de los modelos SPOTER entrenados.

Nombre	Capas Encoder-Decoder	Capas Feedforward
6C-32FF	6	32
3C-32FF	3	32
1C-32FF	1	32
1C-2FF	1	2
1C-4FF	1	4
1C-8FF	1	8
1C-128FF	1	128
1C-512FF	1	512
1C-2048FF	1	2048

3.2.1. Compresión del modelo

Como siguiente paso, se implementó la técnica de destilación de conocimiento para entrenar versiones reducidas del modelo SPOTER. Para seleccionar al modelo maestro que guiaría el aprendizaje de los modelos comprimidos, se eligió el modelo con mejor desempeño en precisión top-1 y top-5 obtenido durante la fase previa de entrenamiento con*cross-entropy loss*. Una vez elegido el modelo maestro, se tomaron como estudiantes los dos modelos más compactos: SPOTER con 1 capa en el encoder y el decoder, equipado con 2 capas*feedforward*, y SPOTER con 1 capa en el encoder y el decoder, pero con 4 capas*feedforward*. Para analizar el efecto del

componente de divergencia*Kullback-Leibler*, se probó la variación del parámetroαcon los valores 0.25, 0.5 y 0.75. Un menor valor deαreduce la influencia del modelo maestro en el entrenamiento del estudiante, mientras que un mayor valor incrementa la dependencia en las predicciones del modelo maestro como guía.

Esta etapa produjo un nuevo conjunto de modelos, adicional al grupo anterior entrenado exclusivamente con*cross-entropy loss*. Los modelos de este segundo conjunto aprovecharon la técnica de destilación para entrenarse con la base de datos (verdad base) en conjunto con las predicciones del modelo maestro, que posee una arquitectura de 1 capa en el encoder y el decoder pero con 128 capas*feedforward*. La configuración de la función de pérdida empleada se describe en el listado 3.1, que reemplaza la*cross-entropy loss*estándar utilizada previamente en la ecuación (2.9). En esta función, se equilibra la contribución de dos componentes: la pérdida basada en la distancia entre las etiquetas reales y las predicciones del modelo estudiante (línea 11), y la pérdida basada en la distancia entre las predicciones del modelo maestro y las del estudiante referida como divergencia de*Kullback-Leibler*(línea 12). Este balance permite integrar efectivamente el conocimiento del modelo maestro durante el proceso de entrenamiento, maximizando el rendimiento de los modelos comprimidos.

Listing 3.1: Cálculo de pérdida por destilación

```
1  class DistillationLoss(nn.Module):
2      def   init  (self, temperature=3.0, alpha=0.5):
3          super(DistillationLoss, self).  init  ()
4          self.temperature = temperature
5          self.alpha = alpha
6          self.criterion = nn.KLDivLoss(reduction='batchmean')
7
8      def forward(self, student_outputs, teacher_outputs, targets):
9          soft_targets = F.softmax(teacher_outputs / self.temperature, dim=1)
10         soft_outputs = F.log_softmax(student_outputs / self.temperature,
11                                      dim=1)
12         hard_loss = nn.CrossEntropyLoss()(student_outputs, targets)
13         distillation_loss = self.criterion(soft_outputs, soft_targets) *
14                                  (self.alpha * self.temperature ** 2)
15         return distillation_loss + (1. - self.alpha) * hard_loss
```

Finalmente, para probar la técnica de cuantización, se aplicó cuantización dinámica a 8 bits a todas las variantes anteriormente entrenadas. Esto permitió generar versiones más ligeras tanto de

Tabla 3.2: Parámetros de los modelos entrenados con destilación de conocimiento.

Modelo Estudiante	α	Formato de pesos	Modelo maestro	
1C-2FF-Alpha25-M1		0.25	Float32	1C-128FF
1C-2FF-Alpha50-M1		0.50	Float32	1C-128FF
1C-2FF-Alpha75-M1		0.75	Float32	1C-128FF
1C-4FF-Alpha25-M1		0.25	Float32	1C-128FF
1C-4FF-Alpha50-M1		0.50	Float32	1C-128FF
1C-4FF-Alpha75-M1		0.75	Float32	1C-128FF
1C-2FF-Alpha25-M1	Cuantizado	0.25	Int8	1C-128FF
1C-2FF-Alpha50-M1	Cuantizado	0.50	Int8	1C-128FF
1C-2FF-Alpha75-M1	Cuantizado	0.75	Int8	1C-128FF
1C-4FF-Alpha25-M1	Cuantizado	0.25	Int8	1C-128FF
1C-4FF-Alpha50-M1	Cuantizado	0.50	Int8	1C-128FF
1C-4FF-Alpha75-M1	Cuantizado	0.75	Int8	1C-128FF

los modelos originales como de las variantes reducidas, preservando su capacidad de inferencia con un menor costo computacional. La combinación de destilación de conocimiento y cuantización resultó en un conjunto amplio de modelos comprimidos. En la Tabla 3.2 se presenta un resumen de todas las variantes generadas, detallando el modelo maestro, las configuraciones de capas *feedforward*, el valor de α en los modelos entrenados con destilación y si se aplicó cuantización a cada variante.

3.3. Métricas evaluadas

Si bien las técnicas de compresión pueden resultar prometedoras para reducir la demanda computacional de la inferencia. Es importante recordar que traen sus propias desventajas que pueden limitar su utilidad aplicada en escenarios de uso real. Por ello, es importante combinar la evaluación de su capacidad de compresión con evaluaciones sobre sus efectos en el desempeño de los modelos integrados en el navegador. Para medir el desempeño de los modelos (precisión y velocidad de inferencia) junto con otros factores relevantes, se plantearon entonces las métricas detalladas en la Tabla 3.3.

3.4. Integración en navegador

Para habilitar el despliegue de cada modelo a través de inferencia en el navegador, convertimos todos los modelos al formato ONNX. Una vez convertidos se evaluó su desempeño en un entorno especializado de inferencia explicado en 3.4.2 y en un entorno equivalente al de uso real dentro de

Tabla 3.3: Descripción de métricas evaluadas.

Métrica	Definición
Precisión (Top-1 y Top-5)	Predicciones correctas/Todas las predicciones
Velocidad de predicción	Velocidad total de procesamiento e inferencia del modelos
Tiempo de época	Tiempo que demora en entrenar 1 época del modelo
Tamaño	Tamaño en Mb del modelo
Velocidad de carga	Tiempo que demora la página en dar una respuesta

una página general descrito en 3.4.3.

3.4.1. Conversión a ONNX

El proceso de conversión implica exportar los modelos entrenados desde sus*frameworks* originales (.pth en Pytorch) al formato .onnx. Este formato proporciona la compatibilidad necesaria para ejecutar los modelos directamente en navegadores mediante ONNX Runtime, que facilita la inferencia en dispositivos sin depender de frameworks pesados. Es requerido dado que no existe una alternativa de Pytorch que permita ejecutar las operaciones requeridas para la inferencia de modelos con los recursos del dispositivo local desde el navegador.

Al momento de migrar el modelo, el entorno de ONNX opera con su propio conjunto de operadores. Por ello, la conversión realizada no es idéntica a la original, sino que un equivalente. Sin embargo, se observó que en casos donde el modelo deba aceptar secuencias de longitudes variables esta conversión resulta inexacta. El modelo en formato ONNXfija la entrada a una sola longitud de secuencias en base al vector de ejemplo utilizado durante la conversión.

De forma más precisa, se utilizó como punto de partida la función*export*de la librería*onnx*. Esta función relaciona aquellos operadores equivalentes entre ambas librerías para generar un modelo convertido en el formato ONNX. Sin embargo, esta función no permitió una conversión directa debido a la presencia de un eje dinámico en el número de frames de los vectores de entrada*(frames, 54, 2)*. Al utilizar la función sin modificaciones, se generó un error de tipo *RUNTIME_EXCEPTION*en la inferencia en formato ONNX, relacionado con la operación *Reshape*en el nodo de multi-atención del*encoder*y el nodo de atención encoder-decoder. En el modelo convertido en formato ONNX, no se permitía realizar inferencia de videos con longitud variable.

Al analizar el método de operación para estos nodos, se identificó que el origen del error estaba en la implementación de la función*_in_projection_packed*dentro de la librería Pytorch. Esta función se encarga de calcular de manera optimizada los pesos para el mecanismo de atención.

Es utilizada en operaciones donde dos o tres tensores del mecanismo de atención son idénticos, lo que permite realizar las proyecciones de manera conjunta. Sin embargo, en la versión más reciente de Pytorch, esta operación fue actualizada para optimizar el cálculo en GPU. Donde N representa el número de tensores idénticos (por ejemplo, query = key = value) y E es la dimensión del embedding.

$$proj = proj \square unflatten(-1, (N, E)) \square unsqueeze(0) \square transpose(0, -2) \square squeeze(-2) \square contiguous() \tag{3.1}$$

Estas operaciones no permitían la conversión correcta dado que en uno de sus operadores equivalentes en ONNX asignaban como constante el número de *frames*. Por ello, para solucionar el problema se optó por simplificar los operadores.

$$proj = proj \square chunk(N, dim = -1) \tag{3.2}$$

Esta modificación permitió resolver el conflicto con la operación *Reshape*, habilitando el uso de un número variable de *frames* en los vectores de entrada. Cabe destacar que ambas operaciones proporcionan el mismo resultado exacto, con la única diferencia de que la versión corregida con *chunk* permite aceptar ejes dinámicos al convertirse en formato onnx.

3.4.2. Pruebas en entorno especializado

Los modelos convertidos fueron desplegados inicialmente en un entorno de pruebas utilizando el benchmark web de ONNX proporcionado por Microsoft [28]. Este benchmark ofrece una estimación precisa de las velocidades de inferencia que los usuariosfinales experimentarían en sus navegadores. Los modelos se integran compilando sus ejecuciones a través del formato Web Assembly disponible mediante ONNX Runtime. En consecuencia, esta implementación accede a la CPU del dispositivo al realizar inferencias.

La efectividad de los modelos destilados cuantizadosfinales se evalua implementándolos en un entorno basado en la web. Los modelos se integraron compilando sus ejecuciones utilizando el formato Web Assembly disponible a través de ONNX Runtime. Primero, el modelo se carga mediante una solicitud inicial para un calentamiento de 50 muestras de forma que el entorno asigne la memoria requerida para la carga del modelo [29]. Una vez cargado, se calcularon otras 1000 inferencias mientras se medía el tiempo requerido para cada una para calcular el tiempo medio de inferencia promedio.

Dado que el objetivo de este *benchmark* era medir únicamente el tiempo de inferencia, la entrada para el modelo es un conjunto de matrices generadas aleatoriamente con forma fija de [70,54,2]. La duración de cada inferencia se registra en una lista, de la cual se calcularon posteriormente el tiempo promedio y su desviación estándar.

3.4.3. Pruebas en entorno de producción

Además de las pruebas realizadas en entornos especializados de *benchmarks*, se replicó el modelo en un entorno de producción realista para evaluar su desempeño. Este proceso consistió en integrar el modelo en una copia del sitio web del diccionario de lengua de señas peruana desarrollado por la PUCP, utilizando React.js[2]. En este entorno, el sistema previo del diccionario fue adaptado para realizar la inferencia de manera completamente local, eliminando la dependencia de un servidor externo. A diferencia del entorno especializado, en el entorno de producción se recibe videos de usuarios como datos de entrada en lugar de los vectores de *keypoints* ya extraídos. La interfaz utilizada, mostrada en la Figura 3.1, representa el entorno de pruebas en producción, donde se realizaron las pruebas en cuestión.

Figura 3.1: Interfaz web de pruebas en producción.

La extracción de puntos clave se integró al sitio mediante la API de Mediapipe Holistic, la cual permite ejecutar la inferencia del modelo de pose de forma local. Esta API procesa los videos proporcionados por los usuarios y genera vectores de puntos clave para pose y manos en cada *frame*. Dado que este componente ya estaba implementado previamente en la página mediante la

[2] `https://diccionariolsp.pucp.edu.pe`

importación de la librería de Mediapipe, el principal desafío consistió en trasladar las operaciones adicionales, como el preprocesamiento y la inferencia del modelo SPOTER, al entorno local.

El primer paso fue integrar el preprocesamiento directamente en la página, lo cual consistió principalmente en la imputación de valores faltantes para garantizar la consistencia de los datos antes de la inferencia. Posteriormente, se desarrolló un componente que permite a los usuarios cargar videos en lotes y ejecutar la inferencia de los modelos de predicción de señas entrenados para cada uno de los videos. Esteflujo de trabajo habilita que la página pueda procesar múltiples videos de forma autónoma.

Para evaluar el desempeño del modelo en este entorno de producción, se utilizó el conjunto de validación de la base de datos de 187 videos. Esto permitió medir tanto la velocidad de inferencia bajo condiciones reales como la precisión del modelo al predecir las señas a partir de los videos. Además, la implementación incluye una interfaz interactiva, que permite cargar los videos en bloques, ejecutar las inferencias, y descargar los resultados principales en formato CSV para un análisis posterior. Esta integración no solo permitió evaluar la precisión y velocidad del modelo en un entorno realista, sino que también permitió poner a prueba la viabilidad de ejecutar inferencias de modelos de aprendizaje profundo directamente en el navegador.

Asimismo, se desplegó una versión de este entorno con inferencia en la nube con elfin de realizar una análisis comparativo entre ambas metodologías de despliegue. En este sentido, se utilizó como base la arquitectura de AWS descrita en [8] para el despliegue del modelo a través del servicio de AWS Sagemaker. No obstante, a diferencia de la arquitectura web original del diccionario bilingüe LSP existente, en esta implementación el preprocesamiento de los*keypoints* se traslada al navegador. El proposito de este cambio es aislar la inferencia del modelo como único componente ejecutado en la nube. De este modo, el entorno de producción en la web envía la entrada ya preprocesada a un*API Gateway*de AWS, el cual invoca una función AWS Lambda encargada de llamar al servicio SageMaker para ejecutar la inferencia del modelo. La Figura 4.7 describe el diagrama de arquitectura correspondiente al sistema descrito con elflujo de operaciones completo que realiza.

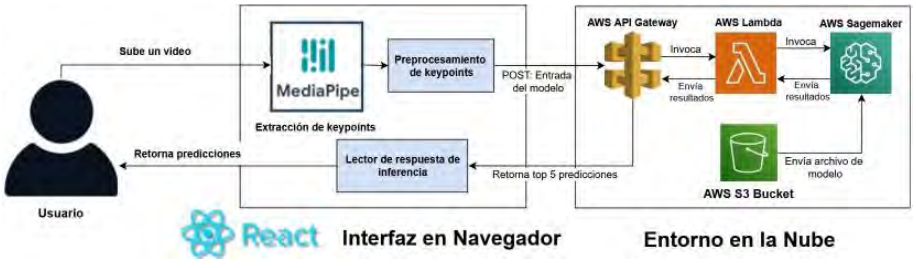

Figura 3.2: Arquitectura del sistema para pruebas con inferencia en la nube.

Capítulo 4

RESULTADOS

Siguiendo la metodología descrita en el capítulo anterior, se obtuvieron resultados relacionados con el entrenamiento de modelos, su compresión y la posterior adaptación para inferencia en navegadores. Estos resultados incluyen tanto pruebas en entornos especializados como evaluaciones en un entorno que simula condiciones reales. Para el entrenamiento de los modelos se utilizó un conjunto de seis tarjetas gráficas NVIDIA A100 de 80 GB, mientras que las pruebas de análisis en navegador se llevaron a cabo en un dispositivo único equipado con un procesador AMD Ryzen 7 de 3200 MHz y 8 núcleos.

4.1. Precisión de modelos entrenados

Se entrenaron tres variantes del modelo original SPOTER para determinar el número óptimo de capas en el encoder y el decoder. Estas configuraciones fueron modelos con 6 capas, 3 capas y 1 capa, respectivamente. La Figura 4.1 muestra la evolución de la precisión en el conjunto de validación durante el entrenamiento de las tres configuraciones evaluadas.

De acuerdo con los resultados presentados en lafigura, la configuración con 1 capa en el encoder y el decoder (1C-32FF) demostró ser la más efectiva, alcanzando una precisión del 70.1 % después de completar las 900 épocas asignadas. En contraste, los modelos con mayor número de capas alcanzaron menores niveles de precisión antes de detenerse debido al criterio de paciencia configurado en 100 épocas. Específicamente, el modelo con 3 capas (3C-32FF) alcanzó una precisión del 63.6 %, mientras que el modelo con 6 capas (6C-32FF) obtuvo un máximo del 59.4 %. Estos resultados indican que un diseño más compacto de 1 capa en encoder y decoder no solo logra una mayor precisión, sino que también obtiene mejores resultados durante el entrenamiento. Más aún, el modelo con 1 capa en el encoder y el decoder (1C-32FF) también se

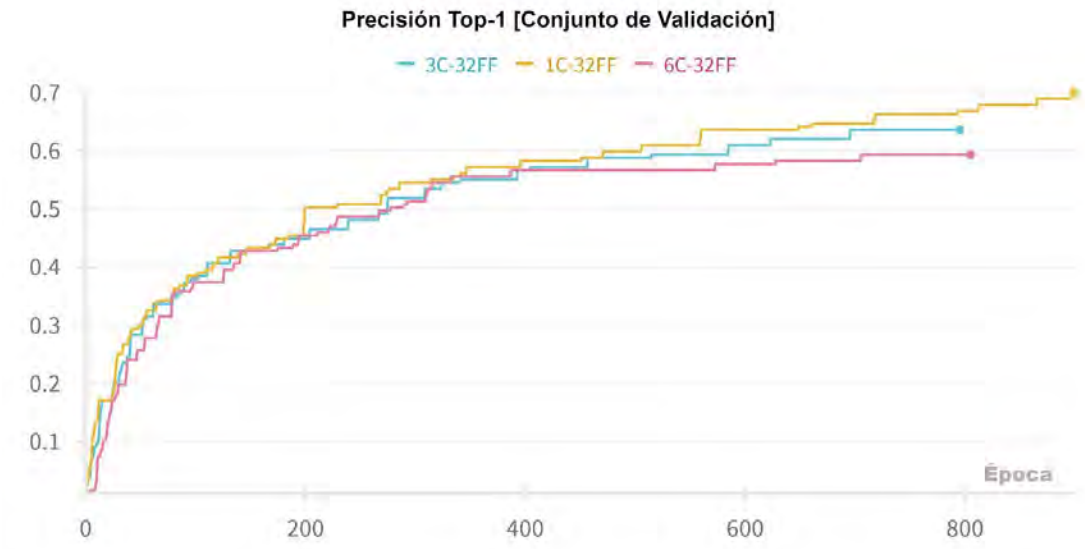

Figura 4.1: Evolución de la precisión en validación para los modelos iniciales entrenados con *cross-entropy loss*.

destacó por ser el más rápido durante el entrenamiento. Este modelo requirió un promedio de 25.5 segundos por época, mientras que el modelo con 3 capas (3C-32FF) necesitó 61 segundos por época, y el modelo con 6 capas (6C-32FF) alcanzó un promedio de 114.5 segundos por época.

Con base en estos resultados, se seleccionó el modelo con 1 capa en el encoder y el decoder como la configuración inicial para entrenar un nuevo conjunto de modelos, variando el número de capas*feedforward*en el encoder y el decoder. Los resultados de estas pruebas se presentan en las Figuras 4.2 y 4.3.

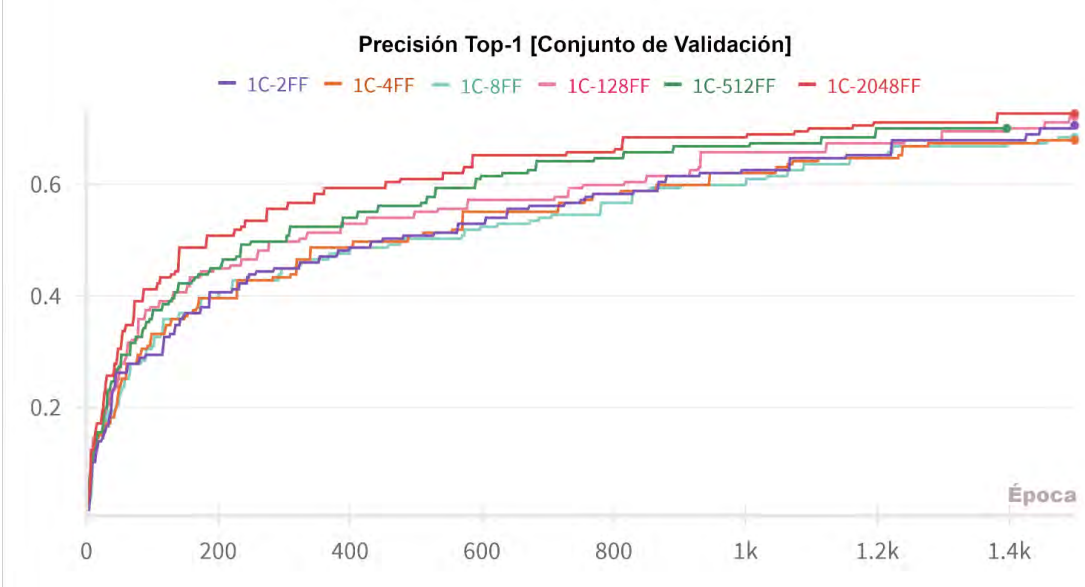

Figura 4.2: Precisión Top-1 de modelos de 1 capa encoder-decoder.

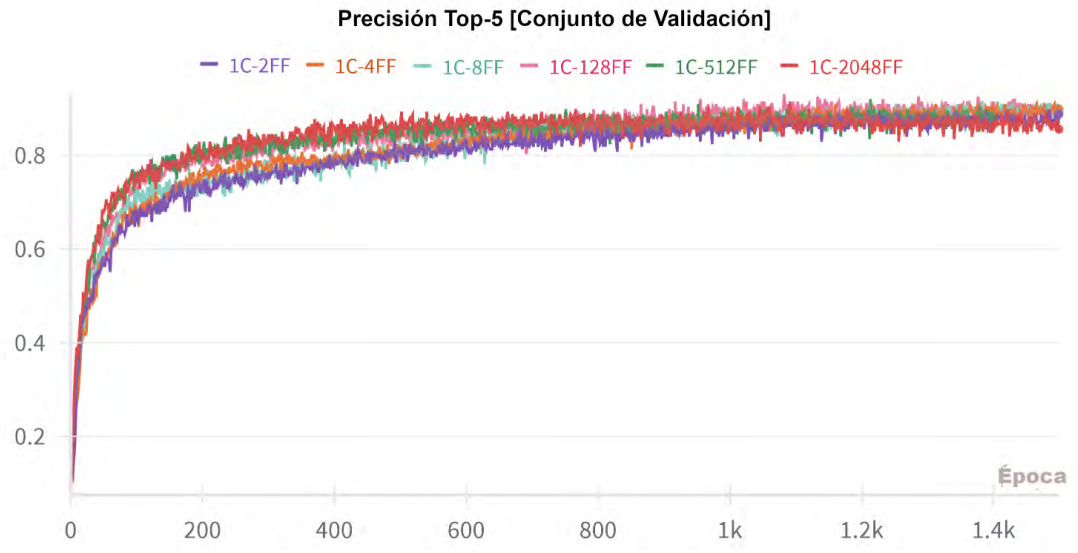

Figura 4.3: Precisión Top-5 de modelos de 1 capa encoder-decoder.

Los datos muestran que los modelos con 2048 y 128 capas*feedforward*alcanzaron los mejores desempeños en términos de precisión top-1, logrando valores de 72.3 % y 72.2 %, respectivamente. No obstante, al analizar la precisión top-5, se observa una diferencia más pronunciada: el modelo con 128 capas*feedforward*superó al de 2048 capas, logrando una precisión de 89.3 % frente a 86.1 %

En cuanto al tiempo de entrenamiento, se observó que los modelos con diferentes configuraciones de capas*feedforward*presentaron tiempos muy similares, oscilando entre 26 y 27 segundos por época. Esto indica que el número de capas*feedforward*no tiene un impacto significativo en el tiempo de entrenamiento, manteniendo una eficiencia computacional consistente. En comparación con los tiempos de entrenamiento de los modelos iniciales con 3 capas (61 segundos por época) y 6 capas (114.5 segundos por época), esta variación resulta notablemente menor.

Se presenta entonces en la Tabla 4.1 un resumen de los resultados de entrenamiento, donde se destaca el desempeño del modelo 1C-128FF, que obtuvo una precisión Top-1 de 72.2 % y una precisión Top-5 de 89.3 %. Para el siguiente paso entonces, este modelo es elegido como modelo maestro para guiar a los modelos reducidos que serán asignados como estudiantes.

Tabla 4.1: Resumen de precisión de modelos entrenados con *cross-entropy loss*.

Modelo	Top-1 (%)	Top-5 (%)
6C-32FF	59.4	86.1
3C-32FF	63.6	86.6
1C-32FF	70.1	87.7
1C-2FF	70.1	86.7
1C-4FF	67.4	89.8
1C-8FF	68.4	88.7
1C-128FF	72.289.3	
1C-512FF	70.1	88.7
1C-2048FF	72.386.1	

4.2. Compresión de modelo

Tomando como referencia al modelo 1C-128FF como maestro, se entrenaron modelos reducidos con arquitecturas más compactas, específicamente 1C-2FF y 1C-4FF, utilizando la técnica de destilación de conocimiento. En las Figuras 4.4 y 4.5, se ilustran los valores de precisión Top-1 y Top-5 de estos modelos, mostrando cómo varía la capacidad de inferencia en función de los valores asignados al parámetroα, que controla la influencia del modelo maestro durante el entrenamiento.

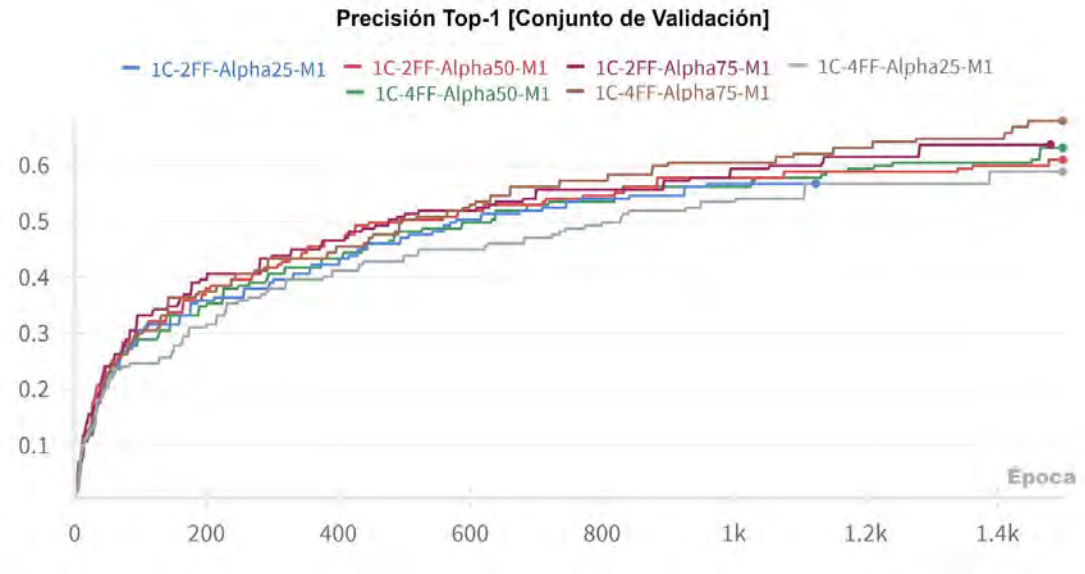

Figura 4.4: Precisión Top-1 de modelos entrenados con destilación de conocimiento.

Los resultados obtenidos indican que un mayor valor del parámetroαse asocia con un mejor desempeño de los modelos reducidos. Sin embargo, ninguna de las configuraciones con destilación

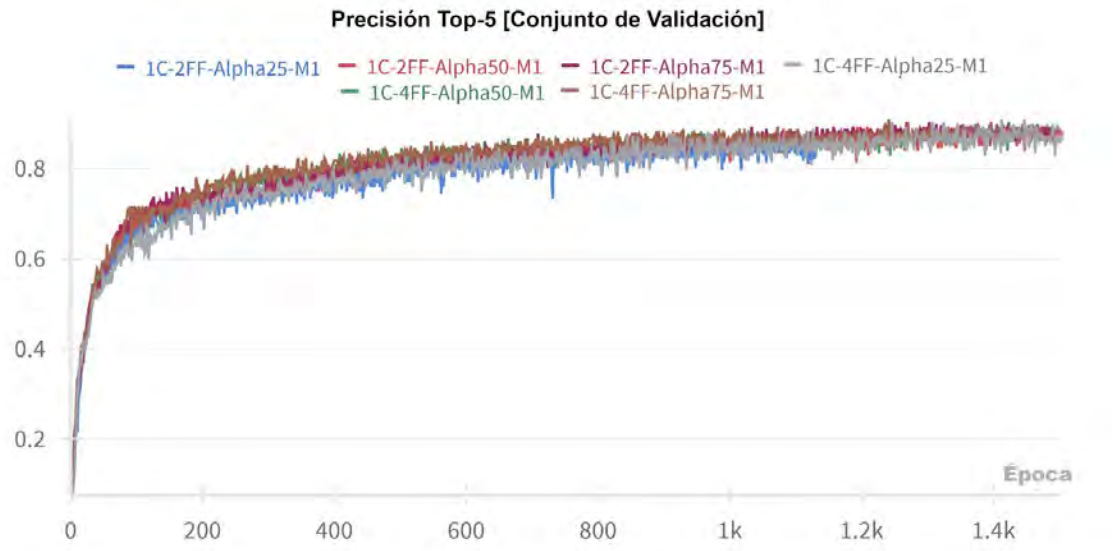

Figura 4.5: Precisión Top-5 de modelos entrenados con destilación de conocimiento.

logra superar los resultados de los modelos originales 1C-2FF y 1C-4FF. Específicamente, los mejores resultados obtenidos con $\alpha = 0,75$ muestran que una menor influencia de la pérdida por divergencia *Kullback-Leibler* tiende a mejorar la precisión. Esto sugiere que el modelo maestro no proporciona una ventaja significativa en el proceso de destilación, y que los modelos reducidos no se benefician de la guía propuesta por el proceso de destilación de conocimiento para el objetivo de reconocimiento de lengua de señas con la base de datos evaluada.

Tabla 4.2: Resumen de precisiones de los modelos entrenados con destilación de conocimiento.

Modelo	Top-1 (%)	Top-5 (%)
1C-2FF-Alpha25-M1	56.1	84.0
1C-2FF-Alpha50-M1	60.4	87.2
1C-2FF-Alpha75-M1	63.6	88.8
1C-4FF-Alpha25-M1	58.8	88.8
1C-4FF-Alpha50-M1	62.0	87.7
1C-4FF-Alpha75-M1	66.8	88.8

Con respecto a la cuantización, esta se aplicó a todas las versiones de modelos entrenadas anteriormente. Para ello, primero se los exportó en formato ONNX para su compatibilidad con inferencia en navegador, y una vez en ese formato, se los cuantizó comprimiendo sus pesos. La Figura 4.6 presenta una comparación del tamaño de los modelos en sus versiones originales, entrenadas con pesos en formato Float32, y sus versiones cuantizadas a pesos en formato Int8. Cabe destacar que las variantes entrenadas con destilación de conocimiento, como 1C-2FF y 1C-4FF, mantienen el mismo tamaño que sus versiones originales debido a que no se realizaron

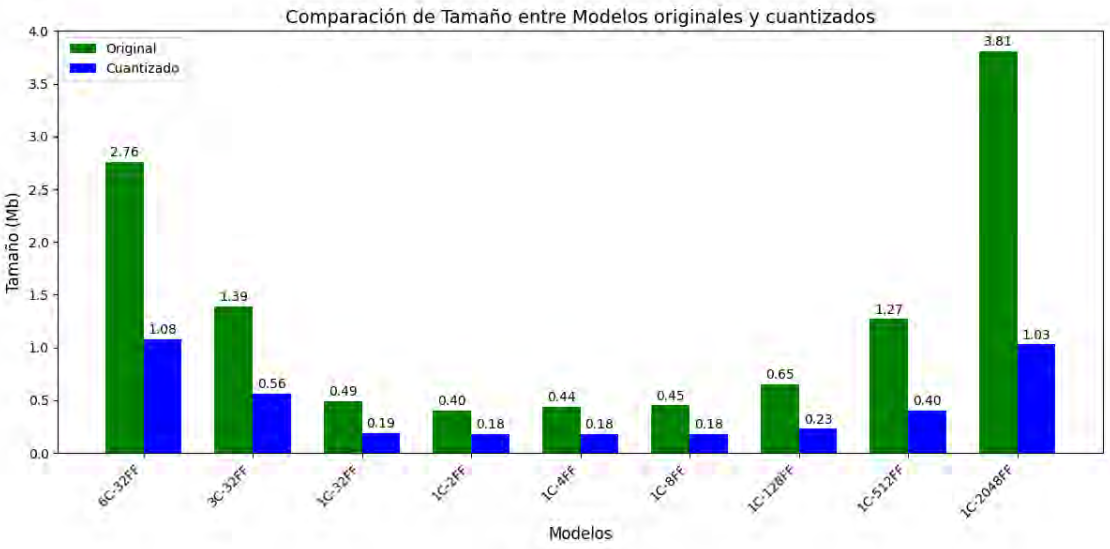

Figura 4.6: Comparación de tamaño de modelos con cuantización en formato ONNX.

modificaciones en su arquitectura durante el proceso de destilación.

Como se puede observar, la cuantización reduce de manera significativa el tamaño de los modelos al comprimir los pesos originalmente aprendidos, especialmente en las arquitecturas más complejas como 6C-32FF y 1C-2048FF. Esta reducción de tamaño supone una ventaja clave para la implementación en dispositivos con recursos limitados, ya que permite mantener la capacidad de inferencia mientras se optimiza el uso de memoria. La Tabla 4.3 compara los modelos en formato ONNX con sus pesos originales con sus versiones cuantizadas, mostrando las precisiones Top-1 y Top-5 alcanzadas por cada configuración, así como las disminuciones en precisión ocasionadas por la cuantización.

Los resultados reflejan que los modelos no son afectados por la exportación a formato ONNX, manteniendo sus capacidades de inferencia intactas luego de la conversión. La cuantización, por otra parte, sí introduce ligeras disminuciones en precisión, variando tanto las capacidades de predicción top-1 y top-5 de los modelos. Sin embargo, estas disminuciones son sumamente reducidas, alcanzando una reducción de 0.6 % en el peor caso para 1C-2FF. Por ello, se encuentra que la cuantización resulta un método viable para comprimir los modelos de forma efectiva.

4.3. Validación en entorno especializado

Se realizó una validación del modelo en un entorno especializado utilizando la implementación en ONNX, evaluando específicamente los tiempos de inferencia en un navegador

Tabla 4.3: Resumen del efecto de la cuantización sobre la precisión de los modelos.

Modelo	Top-1 (%)	Top-5 (%)	Disminución (%)	
			Top-1	Top-5
6C-32FF	59.4	86.1	0.6	0.5
6C-32FF Cuantizado	58.8	85.6		
3C-32FF	63.6	86.6	0.5	0.0
3C-32FF Cuantizado	63.1	86.6		
1C-32FF	70.1	87.7	0.1	0.5
1C-32FF Cuantizado	70.0	87.2		
1C-2FF	70.1	86.7	0.6	0.6
1C-2FF Cuantizado	69.5	86.1		
1C-4FF	67.489.8		0.6	0.5
1C-4FF Cuantizado	66.8	89.3		
1C-8FF	68.4	88.7	0.0	0.0
1C-8FF Cuantizado	68.4	88.7		
1C-128FF	72.2	89.3	0.5	0.6
1C-128FF Cuantizado	71.7	88.7		
1C-512FF	70.1	88.7	0.1	0.0
1C-512FF Cuantizado	70.0	88.7		
1C-2048FF72.386.1			0.0	0.5
1C-2048FF Cuantizado	72.3	85.6		

web. La Tabla 4.4 presenta los resultados obtenidos para las diferentes configuraciones de los modelos entrenados. Como se observa en estos resultados, se identifica que aquellos modelos de 1 capa de encoder y decoder tienen un mejor desempeño en términos de velocidad de inferencia comparados con aquellos más grandes de 3 o 6 capas. Sin embargo, al estandarizar el número de capas del encoder y decoder a una sola, se observa que el número de capas*feedforward* no ejerce un impacto significativo en la velocidad de inferencia, lo que sugiere que la complejidad adicional en esta dimensión no compromete el tiempo de procesamiento para ejecutar las inferencias. En este sentido, resulta ventajoso ajustar el parametro de número de capas *feedforward* dentro del encoder y decoder de los modelo si se desea optimizar una arquitectura sin aumentar el tiempo de inferencia en los modelos resultantes.

4.4. Validación en entorno de producción

Los modelos fueron evaluados en un entorno de producción implementado en React, utilizando métricas como precisión Top-1 y Top-5 para medir su desempeño. Los resultados obtenidos reflejan una distribución de desempeño similar a las pruebas realizadas en entornos especializados, destacándose el modelo de 1 capa con 2048 capas de *feedforward* como el de mejor precisión. La Tabla 4.5 resume las métricas principales recopiladas en este entorno,

Modelo	Tiempo por inferencia (ms)
6C-32FF	0.70±0.32
6C-32FF cuantizado	0.60±0.17
3C-32FF	0.31±0.19
3C-32FF cuantizado	0.31±0.14
1C-32FF	0.15±0.10
1C-32FF cuantizado	0.17±0.15
1C-2FF	0.13±0.08
1C-2FF cuantizado	0.14±0.08
1C-4FF	0.12±0.08
1C-4FF cuantizado	0.14±0.12
1C-8FF	0.13±0.08
1C-8FF cuantizado	0.14±0.09
1C-128FF	0.13±0.07
1C-128FF cuantizado	0.16±0.14
1C-512FF	0.15±0.14
1C-512FF cuantizado	0.17±0.09
1C-2048FF	0.24±0.18
1C-2048FF cuantizado	0.34±0.10

Tabla 4.4: Tiempo de inferencia evaluado en entorno especializado.

abarcando tanto la precisión como los tiempos de inferencia para cada configuración evaluada.

En términos de tiempos de inferencia, se encontró que el promedio por video fue de 4.04± 2.18 segundos, de los cuales aproximadamente 3.80±2.11 segundos corresponden al procesamiento del modelo de estimación de pose. Esto indica que la mayor parte del tiempo requerido para las predicciones recae en Mediapipe, que procesa los*frames*de manera secuencial para identificar los puntos clave en cada uno. No obstante, cabe destacar que el tiempo de inferencia de los modelos entrenados es significativamente mayor cuando se integran al entorno de React en comparación con las mediciones realizadas en el entorno especializado. Esta diferencia puede atribuirse al impacto de la integración del modelo con el resto de los componentes del*framework*, lo que introduce una carga adicional que ralentiza las predicciones.

Al realizar las pruebas comparativas entre el despliegue basado en la nube y el despliegue basado en inference desde navegador, se encuentra que los tiempos de respuesta en el navegador resultan menores a los del sistema de inferencia en la nube actualmente implementado. Tras realizar pruebas en el diccionario con la arquitectura modificada descrita en la sección 3.4.3, se observó un tiempo total promedio de 5.05±2.35 segundos por video con un promedio de 4.27± 2.08 para extraer los*keypoints*usando Mediapipe. De este tiempo total, el tiempo específico que demora el sistema en recibir la predicción solicitada del servidor externo es de 0.77±0.77 segundos, el cuál resulta mayor a todos los tiempos de inferencia medido para las arquitecturas

Tabla 4.5: Resultados de precisión y tiempo de inferencia para pruebas en entorno de producción.

Modelo	Precisión Top-1 (%)	Precisión Top-5 (%)	Tiempo de MediaPipe (s)	Tiempo de Inferencia (s)
6C-32FF	55.08	87.17	4.07±2.01	0.53±0.14
6C-32FF cuantizado	56.68	87.70	3.78±1.99	0.60±0.13
3C-32FF	60.43	87.17	3.93±2.05	0.34±0.13
3C-32FF cuantizado	61.50	87.70	4.01±2.13	0.38±0.13
1C-32FF	63.10	88.77	3.10±2.04	0.13±0.11
1C-32FF cuantizado	66.31	88.77	3.77±1.97	0.17±0.10
1C-2FF	65.78	87.70	3.82±1.98	0.17±1.11
1C-2FF cuantizado	67.91	87.70	3.78±1.99	0.19±0.11
1C-4FF	59.40	86.60	3.78±1.97	0.17±0.11
1C-4FF cuantizado	60.40	86.60	3.80±1.99	0.20±0.12
1C-8FF	59.8989.844.28±2.74		0.16±0.11	
1C-8FF cuantizado	60.43	89.30	3.71±1.94	0.17±0.11
1C-128FF	67.38	89.30	3.82±2.01	0.18±0.11
1C-128FF cuantizado	65.76	88.20	3.80±1.97	0.19±0.12
1C-512FF	65.24	86.63	4.00±2.22	0.19±0.12
1C-512FF cuantizado	66.31	86.10	3.15±2.14	0.14±0.11
1C-2048FF68.9886.63	4.14±2.46	0.21±0.12		
1C-2048FF cuantizado	68.45	87.10	3.67±1.92	0.18±0.10

analizadas en navegador presentadas en la tabla 4.5. En la Figura 4.7 se visualiza una comparación gráfica de la gran diferencia de proporción que existe entre el tiempo de inferencia de ambos métodos de inferencia. Lafigura compara la inferencia en la nube con el caso de la inferencia desde navegador del modelo 1C-128FF con pesos originales y cuantizados. A partir de estos métodos, el sistema consigue una aceleración en su tiempo de inferencia de 4.28 veces en el caso de los pesos originales y de 4.05 con el modelo cuantizado.

Además, el tiempo mencionado para el modelo en la nube no incluye el periodo inicial de 23 segundos necesario para cargar el modelo en la primera llamada, debido al proceso de inicialización de los servidores de inferencia. Una vez que los servidores están activos, el tiempo de predicción se estabiliza en el promedio previamente mencionado. Este resultado subraya las ventajas de realizar la inferencia directamente en el navegador, no solo por la reducción en los tiempos de respuesta, sino también por la eliminación de dependencias externas que deban activarse, haciendo el sistema más eficiente y autónomo.

Un aspecto importante revelado por las pruebas fue la discrepancia entre el modelo de MediaPipe utilizado en Python y su contraparte implementada en el navegador accesible desde Javascript. Esta divergencia resultó en una disminución generalizada en la precisión de los

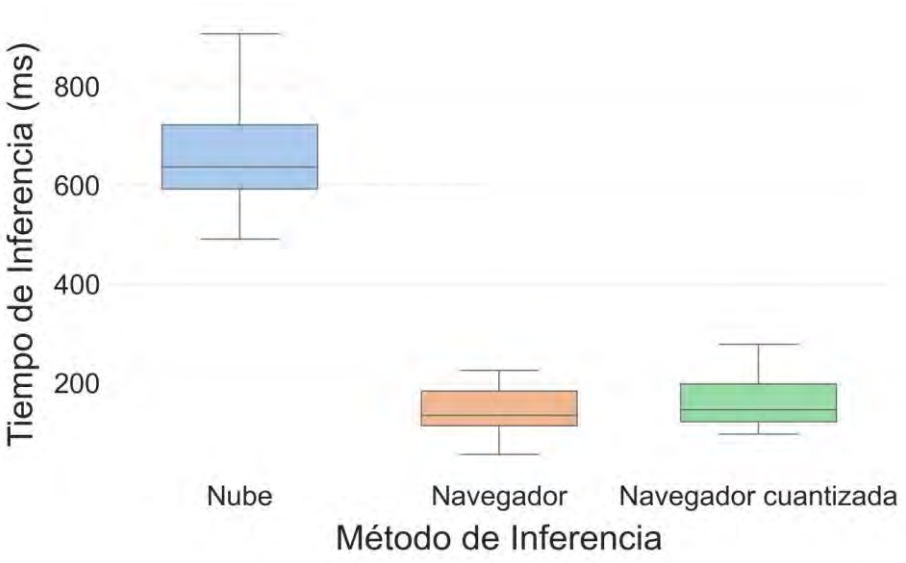

Figura 4.7: Comparación de tiempos de inferencia con distintas métodos de inferencia.

modelos evaluados, comparados con las pruebas realizadas previamente en el entorno de desarrollo en Python. Esto quiere decir, si bien los datos de entrenamiento también fueron preprocesados con Mediapipe, la versión específica del modelo de estimación de pose es distinta a la disponible a través de Javascript. Por ende, habrá una diferencia en el preprocesamiento al desplegar el modelo en navegador que ocasiona una reducción en la capacidad de inferencia de 4.74 % en su precisión Top-1 en promedio para los videos de validación a través de todos los modelos probados en navegador.

Al realizar pruebas sobre cinco videos seleccionados aleatoriamente del conjunto de validación, se encontró que la distancia promedio entre las coordenadas identificadas por los modelos es de 50.62 píxeles. En la Figura 4.8, se presentan ejemplos de diferentes poses en las que se observan discrepancias en la identificación de los puntos clave por parte de los modelos. Estas diferencias pueden variar considerablemente en magnitud. En algunos casos, los puntos clave son identificados en coordenadas próximas pero no exactas, lo que genera ligeros desajustes en la representación de la pose. En situaciones más extremas, uno de los modelos puede identificar correctamente los puntos correspondientes a las manos, mientras que el otro no lo logra, asignando en su lugar las coordenadas al punto de la muñeca, de acuerdo con las reglas definidas durante el preprocesamiento. Esto puede observarse, por ejemplo, en el par de la segundafila de la Figura 4.8, donde el modelo implementado en JavaScript no logra detectar las manos, mientras que el modelo utilizado en Python sí lo hace, lo que genera una discrepancia significativa en las coordenadasfinales identificadas.

Figura 4.8: Divergencias entre modelos de estimación de pose disponibles desde Python y desde Javascript.

Conclusiones

- El entrenamiento de modelos de aprendizaje profundo con arquitectura SPOTER encoder-decoder demostró ser efectivo para la clasificación de señas aisladas en lengua de señas peruana, alcanzando una precisión top-1 de 72.3 % para identificar las 38 palabras señalizadas requeridas.
- Las técnica de cuantización de conocimiento demostró ser efectiva para la compresión de modelos con pérdidas mínimas de precisión. La destilación, por otro lado, resultó inefectiva al dar pérdidas de precisión cuando se integró en el entrenamiento.
- La comparación de modelos en diferentes formatos para *edge computing* reveló que la inferencia directa en navegadores web resulta hasta 4.28 veces más rápida que la realizada mediante servidores externos, no requiere prender un servidor externo la primera vez que se activa el servidor ni esperar por latencia a llamadas remotas.
- La implementación del sistema de validación en un entorno de producción basado en navegadores de escritorio permitió evaluar la inferencia de los modelos en un entorno real. A partir de ello, se demostró que existe una reducción importante en el despliege web debido a las diferencias entre la API de Mediapipe en Python y en Javascript.

Recomendaciones

- Reentrenar el modelo utilizando datos preprocesados directamente con la API web de MediaPipe, asegurando consistencia entre las etapas de entrenamiento e inferencia en entornos reales, lo cual podría reducir discrepancias de precisión.
- Realizar pruebas de desempeño en una mayor cantidad de dispositivos, con el objetivo de evaluar la funcionalidad del sistema bajo distintas condiciones y determinar el efecto específico que el equipo disponible tiene sobre la capacidad del sistema de inferencia en navegador.
- Integrar reconocimiento de señas continuo directamente en el navegador, aprovechando la reducción en tiempos de procesamiento lograda con la inferencia local, para facilitar aplicaciones en tiempo real. Esto implica integrar segmentación como parte delflujo para separar aquellos frames correspondientes a señas específicas.
- Reentrenar el modelo utilizando una técnica de aumento de datos que contemple escenarios en los que las manos del usuario no sean detectadas en*frames*aleatorios, con elfin de mejorar la robustez del sistema ante pérdidas parciales de información.

Bibliografía

[1] V. Bazarevsky, I. Grishchenko, K. Raveendran, T. L. Zhu, F. Zhang, and M. Grundmann, "Blazepose: On-device real-time body pose tracking,"*ArXiv*, vol. abs/2006.10204, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:219793039

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need,"*CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[3] M. Boháček and M. Hrúz, "Sign pose-based transformer for word-level sign language recognition," in*2022 IEEE/CVF Winter Conference on Applications of Computer Vision Workshops (WACVW)*, 2022, pp. 182–191.

[4] INEI, "Censos nacionales 2017: Xii de población, vii de vivienda y iii de comunidades indígenas," 2018. [Online]. Available: https://www.inei.gob.pe/media/MenuRecursivo/publicaciones_digitales/Est/Lib1539/libro.pdf

[5] E. Parks and J. Parks, "Encuesta sociolingüística de la comunidad sorda en perú," SIL International, 2015.

[6] M. Bohacek and S. Hassan, "Sign spotter: Design and initial evaluation of an automatic video-based american sign language dictionary system," in*Proceedings of the 25th International ACM SIGACCESS Conference on Computers and Accessibility*, ser. ASSETS '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3597638.3614497

[7] J. Fink, P. Poitier, M. André, L. Meurice, B. Frénay, A. Cleve, B. Dumas, and L. Meurant, "Sign language-to-text dictionary with lightweight transformer models," in*Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, E. Elkind, Ed. International Joint Conferences on Artificial

Intelligence Organization, 8 2023, pp. 5968–5976, aI for Good. [Online]. Available: https://doi.org/10.24963/ijcai.2023/662

[8] G. Bejarano, J. Huamani-Malca, C. Vasquez, S. Oporto, F. Cerna, C. Ramos, and M. Rodrigues-Mondoñedo, "Lessons from deploying thefirst bilingual peruvian sign language- spanish online dictionary," in*Proceedings of the LREC2024 12th Workshop on the Representation and Processing of Sign Languages: Multilingual Sign Language Resources*. Torino, Italy: European Language Resources Association, 2024.

[9] J. Zheng, Y. Wang, C. Tan, S. Li, G. Wang, J. Xia, Y. Chen, and S. Z. Li, "Cvt-slr: Contrastive visual-textual transformation for sign language recognition with variational alignment," 2023.

[10] Y. Chen, F. Wei, X. Sun, Z. Wu, and S. Lin, "A simple multi-modality transfer learning baseline for sign language translation," 2023.

[11] R. Zuo, F. Wei, and B. Mak, "Natural language-assisted sign language recognition," 2023.

[12] P. Rajendran, S. Maloo, R. Mitra, A. Chanchal, and R. Aburukba, "Comparison of cloud-computing providers for deployment of object-detection deep learning models," *Applied Sciences*, vol. 13, no. 23, 2023. [Online]. Available: https://www.mdpi.com/2076-3417/13/23/12577

[13] OSIPTEL, "Erestel 2022 - encuesta residencial de servicios de telecomunicaciones," 2022. [Online]. Available: https://www.osiptel.gob.pe/media/hkkhfkf3/41661-erestel-2022.pdf

[14] D. Smilkov, N. Thorat, Y. Assogba, A. Yuan, N. Kreeger, P. Yu, K. Zhang, S. Cai, E. Nielsen, D. Soergel, S. Bileschi, M. Terry, C. Nicholson, S. N. Gupta, S. Sirajuddin, D. Sculley, R. Monga, G. Corrado, F. B. Viégas, and M. Wattenberg, "Tensorflow.js: Machine learning for the web and beyond," 2019.

[15] microsoft, "onnxruntime-inference-examples," https://github.com/microsoft/onnxruntime-inference-examples, 2024.

[16] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. Bastien, "Bringing the web up to speed with webassembly," *SIGPLAN Not.*, vol. 52, no. 6, p. 185–200, jun 2017. [Online]. Available: https://doi.org/10.1145/3140587.3062363

[17] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, "Segment anything,"*arXiv:2304.02643*, 2023.

[18] G. Prato, E. Charlaix, and M. Rezagholizadeh, "Fully quantized transformer for machine translation," in*Findings of the Association for Computational Linguistics: EMNLP 2020*, T. Cohn, Y. He, and Y. Liu, Eds. Online: Association for Computational Linguistics, Nov. 2020, pp. 1–14. [Online]. Available: https://aclanthology.org/2020.findings-emnlp.1

[19] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," 2020.

[20] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M. G. Yong, J. Lee, W.-T. Chang, W. Hua, M. Georg, and M. Grundmann, "Mediapipe: A framework for building perception pipelines," 2019.

[21] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," in*Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017, p. 1243–1252.

[22] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate,"*CoRR*, vol. abs/1409.0473, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:11212020

[23] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas, "Predicting parameters in deep learning," 2014.

[24] A. Bhandare, V. Sripathi, D. Karkada, V. Menon, S. Choi, K. Datta, and V. Saletore, "Efficient 8-bit quantization of transformer neural machine language translation model," 2019.

[25] Microsoft, "Onnx runtime documentation," 2024, https://onnxruntime.ai/docs/.

[26] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey,"*International Journal of Computer Vision*, vol. 129, pp. 1789 – 1819, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:219559263

[27] G. Bejarano, J. Huamani-Malca, F. Cerna-Herrera, F. Alva-Manchego, and P. Rivas, "PeruSIL: A framework to build a continuous Peruvian Sign Language interpretation dataset," in*Proceedings of the LREC2022 10th Workshop on the Representation and*

Processing of Sign Languages: Multilingual Sign Language Resources, E. Efthimiou, S.-E. Fotinea, T. Hanke, J. A. Hochgesang, J. Kristoffersen, J. Mesch, and M. Schulder, Eds. Marseille, France: European Language Resources Association, Jun. 2022, pp. 1–8. [Online]. Available: https://aclanthology.org/2022.signlang-1.1

[28] microsoft, "onnxruntime-web-benchmark," https://github.com/microsoft/onnxruntime-web-benchmark, 2024.

[29] Q. Wang, S. Jiang, Z. Chen, X. Cao, Y. Li, A. Li, Y. Ma, T. Cao, and X. Liu, "Anatomizing deep learning inference in web browsers,"*ACM Trans. Softw. Eng. Methodol.*, vol. 34, no. 2, Jan. 2025. [Online]. Available: https://doi.org/10.1145/3688843