TECNICATURA UNIVERSITARIA EN INTELIGENCIA ARTIFICIAL

Visión por Computadora y Procesamiento del Lenguaje Natural

INFORME TP FINAL

Estudiante:

- **Valeri, Lara**

Docentes:

- **Juan Pablo Manson**
- **Lucas Augusto Brugé**
- **Constantino Ferrucci**

Fecha de entrega:

11-12-2024

Repositorio: https://github.com/LaraV15/TP-CV

Índice

Función de la aplicación / Caso de Negocio

La aplicación está diseñada para interpretar el alfabeto del lenguaje de señas en imágenes y videos, facilitando la comunicación entre personas con discapacidades auditivas y aquellos que no conocen este lenguaje.

Visión de la Aplicación

Este proyecto tiene como objetivo desarrollar un detector de letras del alfabeto en lenguaje de señas utilizando la última versión de YOLO disponible y compararlo con versiones anteriores utilizadas en investigaciones previas. Esto busca contribuir a la inclusión social mediante soluciones tecnológicas accesibles y eficaces.

En el futuro, contando con los resultados obtenidos y evaluando que modelo realiza mejor la tarea, la visión sería evolucionar hacia una integración con aplicaciones de video llamada, como por ejemplo Google Meet, Zoom.

Alcance

Historias de Usuario / Casos de Uso

El alcance del proyecto incluye el desarrollo, entrenamiento y evaluación de un modelo basado en YOLO11 para la detección y clasificación del alfabeto del lenguaje de señas americano (ASL). Este modelo se evaluará mediante métricas clave como precisión, mAP y velocidad de inferencia, y se comparará con versiones anteriores.

Casos de uso:

1. Procesar imágenes para detectar la letra correspondiente del alfabeto en lenguaje de señas y devolver una imagen etiquetada.
2. Procesar videos para detectar letras del alfabeto en lenguaje de señas.

Metodología de Trabajo

El desarrollo del proyecto se llevó a cabo en las siguientes etapas:

1. **Recopilación de Información:** Se revisaron trabajos previos relacionados con la detección del lenguaje de señas utilizando modelos de la familia YOLO. Este análisis permitió identificar métricas relevantes y posibles áreas de mejora.
2. **Planteamiento de Hipótesis:** Basándonos en la revisión bibliográfica, se propuso que un modelo basado en YOLO11 podría mejorar la precisión y la velocidad de detección del alfabeto en lenguaje de señas en comparación con versiones anteriores.
3. **Desarrollo del Modelo:** Se implementaron y entrenaron dos versiones del modelo YOLO11 en imágenes y videos del alfabeto en lenguaje de señas americano. El

proceso incluyó la preparación del dataset, la configuración de hiperparámetros y la ejecución en Google Colab.
4. **Análisis de Resultados:** Los resultados obtenidos se analizaron utilizando métricas como precisión, mAP y velocidad de inferencia, evaluando el desempeño del modelo en imágenes y videos.
5. **Comparación:** Se compararon los resultados del modelo YOLO11 con los de YOLOv8 y YOLOv5, tomando como referencia los resultados reportados en trabajos previos, para validar las hipótesis planteadas.

Entorno de trabajo

Google Colab Pro para desarrollar el proyecto ya que ofrece acceso a GPU de alto rendimiento, dado que la PC local no cuenta con estos recursos.

Roboflow, para la gestión del dataset, es una herramienta que facilita la descarga y preparación de datos etiquetados.

Google Drive se utilizó como almacenamiento adicional para respaldar información relevante del proyecto.

GitHub como repositorio principal para el control de versiones. En GitHub se almacenaron el código del proyecto, el dataset procesado, y demás materiales relevantes.

Visual Studio Code para poder probar los modelos con la cámara de la computadora, ya que Google Colab no permite encender la misma.

Hipótesis de Trabajo

Hipótesis

El modelo basado en YOLO11 permitirá mejorar la precisión, velocidad de inferencia y capacidad de generalización en la detección y clasificación del alfabeto en lenguaje de señas en imágenes y videos, en comparación con los modelos basados en versiones anteriores.

Objetivo

Desarrollar y evaluar un modelo basado en YOLO11 para la detección y clasificación del alfabeto del lenguaje de señas americano (ASL) en imágenes y videos. El objetivo incluye comparar su desempeño con otras versiones de YOLO, utilizando métricas como precisión, mAP y velocidad de inferencia, para determinar cuál es más eficiente.

Planificación

- Analizar papers base
- Evaluar dataset existentes
- Entrenar dos variantes de YOLO11
- Evaluar los modelos entrenados, entre ellos y con los resultados obtenidos en el paper.

Analizar Paepers Base

Como parte de la planificación del proyecto, se analizaron dos papers. A continuación se resumen información que se considera importante para el posterior análisis.

Transfer learning with YOLOV8 for real-time recognition system of American Sign Language Alphabet

El análisis de este trabajo no solo ayudó a definir los parámetros y métricas para este proyecto, sino que también marcó la dirección para una comparación consistente con los resultados.

La principal diferencia de este modelo con los anteriores es que integra YOLOv8 con MediaPipe. En este sistema, YOLOv8 se utiliza para la detección de objetos, específicamente para identificar y localizar las regiones de interés que corresponden a los gestos de las manos en las imágenes.

Por otro lado, MediaPipe se utiliza para el seguimiento de puntos de referencia, proporcionando información detallada sobre la posición y el movimiento de la mano al anotar 21 puntos clave en cada imagen.

Con respecto al dataset, se utilizaron tres conjuntos de datos distintos, contando con un total de 29.000 imágenes.

El entrenamiento se realizó con 2000 épocas y no hay datos sobre el tiempo empleado.

Resultados del paper:

	Métricas - YOLOv8 con Media Pipe				
	Precisión	Tasa de recuperación	F1	mAp	mAp50
Test	98%	98%	99%	98%	93%

Otro dato importante es que el sistema procesa **11.78 imágenes por segundo**. Además se ha determinado que el **tiempo promedio de detección es de 84.9 ms por imagen.**

El artículo menciona que para el procesamiento en tiempo real se necesita un tiempo de menos de 100 milisegundos por imagen, lo que equivale a 10 FPS, es decir 10 frames por segundo.

American Sign Language Detection using YOLOv5 and YOLOv8

Este artículo se eligió ya que utiliza el mismo dataset que en el presente proyecto.

Tabla resumen:

	Métricas			
	Modelo	Bounding Box Loss	Classification Loss	mAp
Entrenamiento	Yolov5	0.3265	0.191	93,60%
	Yolov8	0.312	0.1735	96,00%

En el artículo no hay valores de las métricas en validación pero sí gráficas que se utilizarán para hacer las comparaciones.

Otro dato importante es que el entrenamiento total tuvo una duración de 6 hs, el paper no distingue cuánto tiempo tomó cada modelo.

El paper no menciona tiempos de predicción de las señas en testing o en video.po toma predecir cada imagen

En resumen, el artículo concluye que, a pesar de las ventajas de YOLOv8 durante el entrenamiento y validación, YOLOv5 demuestra ser más robusto y confiable en escenarios de prueba prácticos que involucran videos.

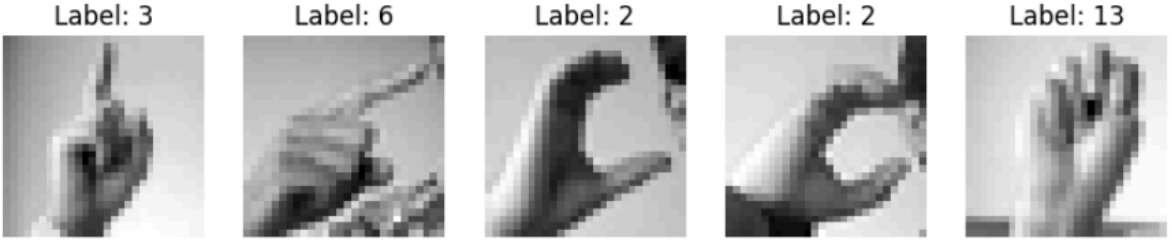
Otras preguntas:
Luego del análisis de los artículos surgen algunas preguntas: con los avances de YOLO11, **¿supera este modelo a YOLOv8? ¿Se pueden obtener resultados similares a los logrados al integrar YOLO con Media Pipe?**

Evaluar conjuntos de datos existentes

Para este trabajo se evaluaron y probaron distintos conjuntos de datos que se detallan a continuación.

- *Sign Language MNIST*
  El conjunto de datos Sign Language MNIST tiene un formato donde cada muestra es una imagen en escala de grises de 28x28 píxeles, representada por 784 valores de píxeles (uno por cada píxel de la imagen). Las etiquetas corresponden a letras del alfabeto en lenguaje de señas estadounidense (ASL), con un total de 24 clases (A-X), ya que las letras J y Z requieren movimientos y no se incluyen.
  El conjunto de datos está dividido en:
    - Datos de entrenamiento: 27,455 muestras
    - Datos de prueba: 7,172 muestras

  Cada fila del archivo CSV contiene una etiqueta (0-25) y los 784 valores de píxeles.
  En principio esto parecía un dataset con mucho potencial por la cantidad de datos, pero al reconstruir las imágenes a partir del valor de sus píxeles las mismas tenían muy mala calidad y al procesarlas, en muchos casos no se reconocían las manos.
  Ejemplo de algunas Imágenes generadas:



  En primer lugar, se intentó marcar los keypoints de las imágenes utilizando media pipe, pero se perdieron muchas imágenes en el camino, ya que no lograba reconocer la mano debido a la calidad de la imagen. El dataset contaba con más de 27,000 datos mientras que en el dataset nuevo, con los keypoints contaba con 334, imágenes de train y 128 de test.

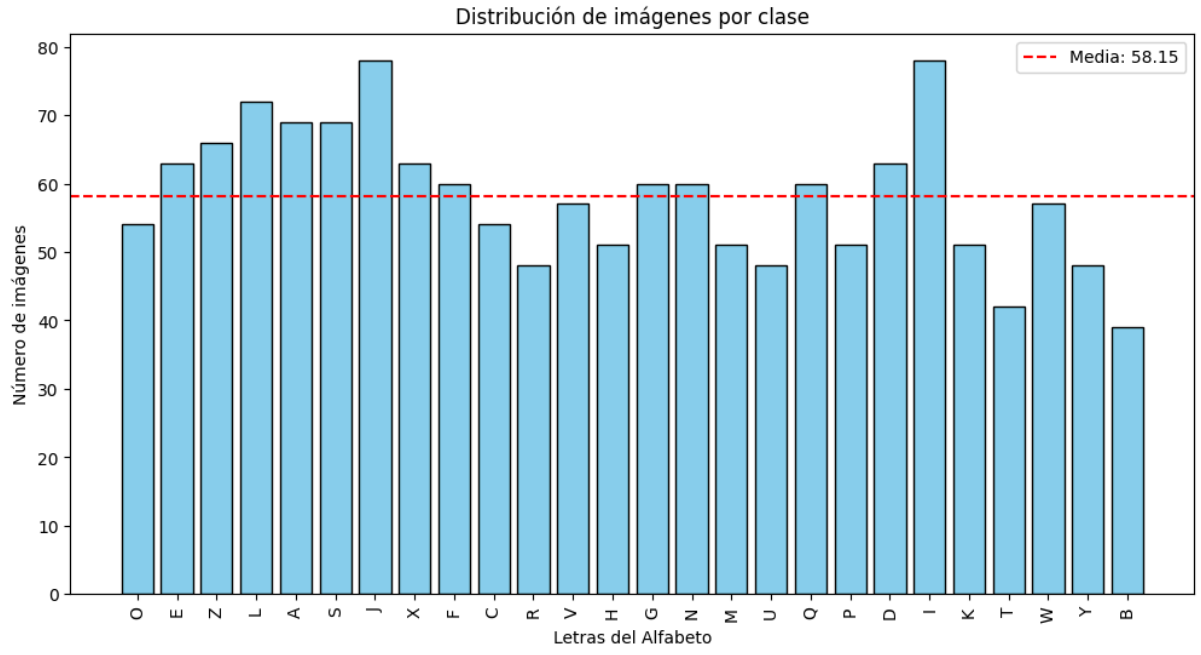
Se entrenaron algunos modelos sencillos, con redes neuronales con capas densas para poder analizar las métricas y no utilizar toda la posibilidad de cómputo de Google Colab. Si los resultados obtenidos eran buenos se iba a proceder a entrenar otro modelo pero las métricas han demostrado un bajo rendimiento del dataset. Los archivos correspondiente a este análisis se pueden ver en el repositorio de GitHub dentro de la carpeta *Sign Language MNIST.*

- Dataset de Roboflow.
  El dataset de Roboflow cuenta con 1512 imágenes de train, 144 de validación y 72 de test.

  El dataset cuenta con tres carpetas train, test, valid donde cada una de ellas contiene dos subcarpetas, images y labels, donde la primera contiene todas las imágenes y la segunda todos los archivos txt que contienen la clase y las coordenadas del bounding box.

  Se realizó un análisis sobre si el dataset esta balanceado o no.

Gráfico obtenido en el notebook



Se puede observar que algunas letras tienen una cantidad de imágenes significativamente mayor o menor que el promedio (media = 58.15), como la letra I (muy alta) o B (baja).

Clases como K e I tienen un número de imágenes mucho mayor que la media. Clases como B y V están por debajo de la media.

Esto indica que el dataset no está completamente balanceado.

En primer lugar, vamos a entrenar el modelo **sin balancear** los datos, en línea con lo que realiza uno de los artículos tomados como base y luego veremos, para futuras iteraciones si sería beneficioso balancear.

Entrenar los modelos

Antes de mostrar los resultados del entrenamiento se realiza un resumen de las métricas y característica que se analizan:

Métrica	Qué mide	Importancia	Ideal
Precisión	Porcentaje de las detecciones que fueron correctas	Muestra qué tan confiables son las detecciones correctas	Porcentaje alto
Recall	Porcentaje de objetos reales que el modelo detectó correctamente	Muestra qué tanto detecta de lo que debería	Porcentaje alto
mAP 0.5	Precisión promedio del modelo al considerar un solapamiento (IoU) del 50% como mínimo entre las cajas predichas y las reales	Indica la calidad general de las predicciones con un IoU mínimo de 50%	Porcentaje alto
mAP 50-95	Precisión promedio del modelo al considerar solapamientos (IoU) más estrictos, entre el 50% y el 95%	Indica la capacidad del modelo en detecciones más estrictas	Porcentaje alto
GFLOPs	Número de operaciones matemáticas (en miles de millones) que un modelo realiza para procesar una imagen o un conjunto de datos	Refleja la complejidad computacional del modelo	Depende del hardware y objetivo
Bounding Box Loss	Evalúa qué tan bien el modelo predice las coordenadas de las cajas delimitadoras en comparación con las cajas reales (ground truth)	Mejora el ajuste de las cajas delimitadoras	Valor bajo
Classification Loss	Evalúa qué tan bien el modelo clasifica los objetos dentro de las cajas delimitadoras	Mejora la clasificación correcta	Valor bajo
DFL Loss (Distribution Focal Loss)	Ajusta la distribución de probabilidad para mejorar la precisión en la ubicación de las cajas	Suaviza los errores en la ubicación de las cajas	Valor bajo
Inferencia	Mide el tiempo que el modelo tarda en procesar una imagen y generar predicciones	Crucial para determinar la viabilidad de tiempo real	Tiempo bajo (ms/imagen)

La inferencia está inversamente relacionada con los FPS (Frames por Segundo).

$$FPS = \frac{1000}{Tiempo\ de\ inferencia}$$

Otra forma de calcular los Frames por segundo

$$FPS = \frac{1}{Tiempo\ por\ imagen\ (en\ seg)}$$

Como los artículos mencionados no hacen referencia a que variante de YOLO utilizan, small, nano, large, medium, etc. En el presente trabajo se entrenan dos variantes de YOLO11: small y large, para realizar comparaciones entre ellas.

Ambos modelos se entrenaron con 80 épocas, misma cantidad que en el segundo artículo y con imágenes de 320x320.

En las siguientes tablas se detallan los resultados obtenidos del entrenamiento y la validación
Características del entrenamiento:

Modelo	Tiempo de ejecución	Parámetros	Capas	GFLOPs
Yolo11l	41 minutos	25,299,358	464	86.7
Yolo11s	24 minutos	9,422,862	238	21.4

	Métricas				
	Modelo	**Bounding Box Loss**	**Classification Loss**	**mAp**	**DFL Loss**
Entrenamiento	Yolo11l	0.3275	0.1722	93,60%	0.9141
	Yolo11s	0.3339	0.1893	96,00%	0.8558

	Métricas					
	Precisión	**Precisión**	**Recall**	**mAp50**	**mAp50-95**	**Inferencia**
Validación	Yolo11l	94.20%	91.70%	96.60%	82.91%	4.2 ms
	Yolo11s	93.20%	89.98%	95.50%	80.9%%	2.9 ms

De los resultados obtenidos podemos concluir que YOLO11s es más rápido en inferencia, lo que se necesita para detección en tiempo real.
Por otro lado, la variante small tiene un rendimiento competitivo en mAP y recall, aunque ligeramente menor que YOLO11l.
En el Notebook, en la sección de 'Resultados en train y valid' se pueden ver algunos gráficos como la matriz de confusión, las pérdidas de entrenamiento e imágenes para un batch de validación.

Evaluar y comparar los modelos

En primer lugar se analizan ambas variantes de YOLO11 entrenadas en este proyecto, comparándolas entre ellas.

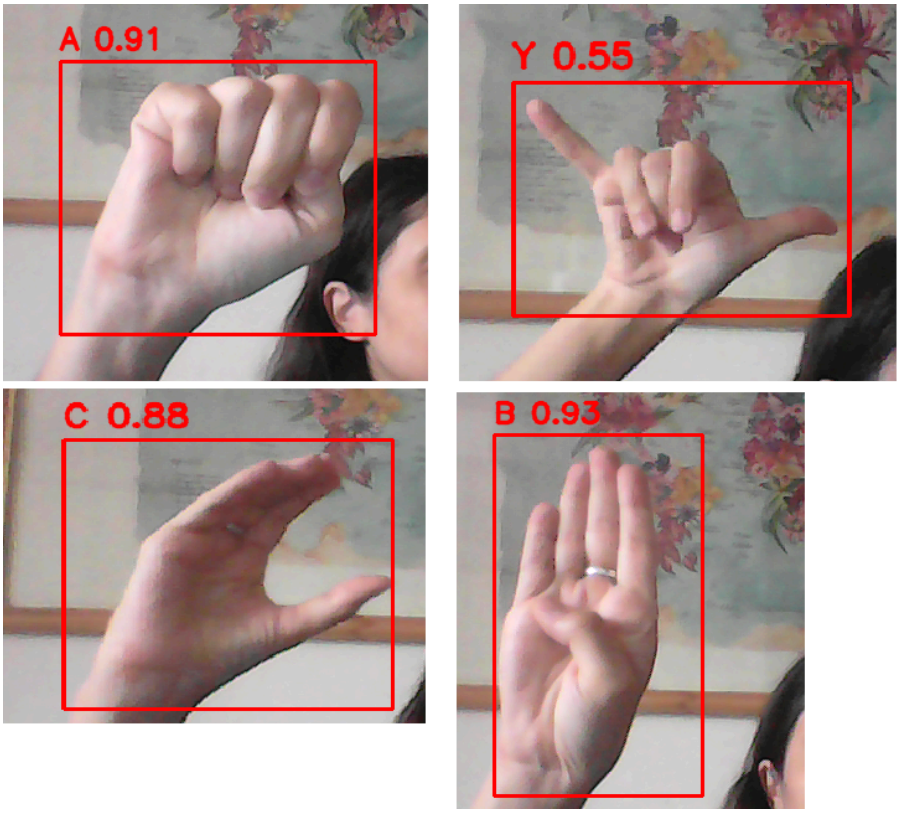
		Test	
Métrica	**YOLO11l**	**YOLO11s**	**Comentario**
Precisión (P)	91,3	91,4	Ambas tienen una precisión muy similar, con YOLO11s ligeramente mejor.
Recall (R)	91,2	88	YOLO11l tiene mejor capacidad para detectar objetos correctamente.
mAP50	95,6	95,5	Rendimiento casi idéntico en mAP50 para ambos modelos.
mAP50-95	79,8	80,4	YOLO11s es ligeramente mejor en detección más estricta (IoU entre 50% y 95%).

		Test		
Métrica	**YOLO11l (ms)**	**YOLO11s (ms)**	**Comentario**	
Preprocesamiento	3,9	3,6	Ambos son rápidos en preprocesar imágenes.	
Inferencia	1013	328,7	YOLO11s es 3 veces más rápido que YOLO11l en inferencia.	
Postprocesamiento	2,3	0,8	YOLO11s es más eficiente en el procesamiento posterior.	

Como para detectar imágenes en tiempo real la velocidad es crucial, se considera que es mejor entre estos dos modelos YOLO11s.

Al probar **YOLO11s con la cámara de una PC,** se observaron tiempos de inferencia rápidos, con un promedio de 161.3 ms a 179.0 ms por imagen, utilizando resoluciones de 256x320, lo que equivale a entre 5 y 6 FPS. Estos resultados reflejan un rendimiento eficiente y consistente, permitiendo una detección fluida en tiempo real sin interrupciones ni retrasos perceptibles. Además, el modelo demostró capacidad para adaptarse a cambios rápidos en los gestos, garantizando una experiencia de uso óptima incluso en hardware estándar.

Se adjuntan algunas recortes de capturas tomadas en estas pruebas.



Comparar los modelos

- **YOLO11s vs YOLOv8 y YOLOv5**

Analizado las métricas en común, ya que en el segundo paper se analizan sólo algunas podemos concluir que YOLOv8 es el modelo más equilibrado, con la mejor precisión en Bounding Box Loss y Classification Loss, además de igualar el mAP de YOLO11s.

Si bien no se tiene información en el artículo sobre tiempos de entrenamiento y validación, en versiones anteriores de este mismo trabajo[1] se pudo observar que YOLOv8s tenía una inferencia de 31.4 ms mientras que YOLO11s tiene una inferencia de 2.9 ms, es decir, aproximadamente 10 veces más rápido, en imágenes.

- **Yolo11s vs YOLOv8 con Media Pipe**

YOLOv8 (con MediaPipe) tiene una precisión del 98%, mientras que YOLO11s alcanza un 91.4%. Esto indica que YOLOv8 genera menos falsos positivos en las detecciones. En cuanto a recall, YOLOv8 también lidera con un 98%, frente al 88% de YOLO11s, lo que refleja que detecta un mayor porcentaje de objetos reales. Sin embargo, en mAP@50, YOLO11s supera ligeramente a YOLOv8 con un 95.5% frente al 93%, demostrando una mejor capacidad de detección en IoU ≥ 50%.

En el primer artículo se menciona que para un correcto procesamiento en tiempo real generalmente se espera que el sistema responda con un tiempo de procesamiento de **menos de 100 milisegundos (ms) por cuadro**, lo que equivaldría a una velocidad de 10 cuadros por segundo (FPS).
En los resultados obtuvieron que el sistema de reconocimiento de ASL posee un tiempo promedio de detección es de **84.9 ms por imagen**.

En YOLO11s, el tiempo de inferencia promedio en tiempo real fue **150 ms por imagen** lo que equivale a aproximadamente 7 FPS. Comparado con el modelo de YOLOv8 del artículo, que tiene un tiempo promedio de 84.9 ms por imagen (11.8 FPS), YOLOv8 es significativamente más rápido y supera el umbral de los 10 FPS mencionados para tiempo real, mientras que YOLO11s no.

Conclusión General

YOLOv8 es el modelo más equilibrado en precisión y rendimiento general:

- Sin MediaPipe: Destaca en Bounding Box Loss y Classification Loss, igualando el mAP de YOLO11s. Sin embargo, su inferencia es significativamente más lenta (31.4 ms vs. 2.9 ms de YOLO11s), lo que lo hace menos adecuado para aplicaciones en tiempo real.
- Con MediaPipe: Mejora aún más en precisión (98%) y recall (98%), mostrando un excelente desempeño en detección de objetos reales y minimizando falsos positivos.

YOLOv8 con MediaPipe sobresale en velocidad:

Los resultados obtenidos muestran que YOLOv8 logra tiempos menores que los que se esperan para tiempo real, mientras que YOLO11s los supera.

[1] Los resultados se pueden ver en el notebook de Google Colab denominado TP-FINAL-YOLOv8: https://drive.google.com/file/d/1cyHaDJ5uT5ppWx-WHSZKrPEyclkaQ5Bw/view?usp=sharing

Limitaciones

YOLO11s tiene limitaciones más marcadas en las clases I, V, P, y N, probablemente debido a similitudes visuales entre algunas de ellas o a la cantidad de datos con los que se entrenó.

Conclusión sobre la hipótesis y objetivos plateados

La hipótesis planteaba que YOLO11 mejoraría la precisión, velocidad de inferencia, y capacidad de generalización en comparación con modelos anteriores. Sin embargo, los resultados indican que:

- Precisión: Aunque YOLO11s alcanza un buen desempeño en mAP@50 (95.5%), no supera a YOLOv8 con MediaPipe, que tiene mayor precisión (98%) y recall (98%). Esto sugiere que la precisión no cumple completamente con la hipótesis.

- Velocidad de Inferencia: YOLO11s tiene un tiempo de inferencia promedio de 150 ms (7 FPS), lo que no cumple el umbral de tiempo real (<100 ms o 10 FPS), mientras que YOLOv8 (84.9 ms) sí lo logra.

- Capacidad de Generalización: YOLO11s muestra un balance sólido en métricas de detección (especialmente mAP@50), lo que confirma su buena capacidad de generalización.

La hipótesis no se cumple completamente. Aunque YOLO11s muestra una buena generalización, no supera a YOLOv8 con MediaPipe en precisión ni en velocidad de inferencia para tiempo real. Futuras optimizaciones en velocidad y detección de clases específicas podrían ayudar a cumplir los objetivos planteados.

Trabajos futuros

1. **Optimización de Clases Problemáticas:** Mejorar la detección de clases con bajo mAP@50-95 (por ejemplo, "I" y "V"), posiblemente mediante aumento de datos específicos y refinamiento del modelo.
2. **Integración con MediaPipe u otros frameworks:** Explorar si la combinación de YOLO11 con MediaPipe puede llevar a un mejor rendimiento y velocidad de inferencia.
3. **Entrenamiento con dataset con mayor cantidad de datos**
4. **Integración con Interfaces de Usuario (UI/UX):** Integrar el modelo con plataformas de videollamadas o aplicaciones móviles, de manera que pueda ofrecer interpretación en tiempo real del lenguaje de señas en entornos cotidianos.

Bibliografía

1. Alsharif, B., Alalwany, E., & Ilyas, M. (2024). Transfer learning with YOLOV8 for real-time recognition system of American Sign Language Alphabet. *Franklin Open, 8*, 100165. Recuperado de https://www.elsevier.com/locate/fraope

2. Tyagi, S., Upadhyay, P., Fatima, H., Jain, S., & Sharma, A. K. (2023). American Sign Language Detection using YOLOv5 and YOLOv8. *Research Square*. Recuperado de https://doi.org/10.21203/rs.3.rs-3126918/v1

3. Ultralytics. (n.d.). Can YOLO11 be deployed on edge devices? Recuperado de https://docs.ultralytics.com/es/models/yolo11/#can-yolo11-be-deployed-on-edge-devices