UNIVERSIDAD RAFAEL LANDÍVAR
FACULTAD DE INGENIERÍA
INTELIGENCIA ARTIFICIAL

PROYECTO 2: Lectura Lenguaje de Seña

RAFAEL ANDRÉS ALVAREZ MAZARIEGOS 1018419
JOSÉ DANIEL DE LEÓN CHANG 1170419
CARLOS ENRIQUE LAPARRA ROBLEDO 1031120

GUATEMALA DE LA ASUNCIÓN, ABRIL DE 2025
CAMPUS CENTRAL “SAN FRANCISCO DE BORJA, S. J” DE LA CIUDAD DE GUATEMALA

Introducción

La comunicación es una necesidad básica del ser humano. Sin embargo, para muchas personas con discapacidad auditiva, expresarse con facilidad en un mundo diseñado para la comunicación dicha por palabras puede ser un verdadero desafío. Se desarrollo un modelo que identifique las señas ASL para poder encontrar palabras y letras.

Este sistema permite traducir en tiempo real las señas captadas por una cámara, transformándolas en letras, palabras y comandos que la computadora puede entender. Gracias a modelos de inteligencia artificial entrenados con imágenes y puntos clave de la mano, es posible interpretar el alfabeto de señas de manera precisa.

Nosotros como equipo queremos ofrecer una herramienta práctica que hable por el usuario y facilite su interacción con el entorno digital. De este modo, se abre la posibilidad de integrarse mejor en espacios educativos, laborales o personales, usando simplemente sus manos y una cámara.

Objetivos

Objetivo General:

Diseñar e implementar un sistema de reconocimiento de lenguaje de señas que identifique letras del alfabeto y traduzca palabras a texto y comandos utilizando visión por computadora y aprendizaje automático.

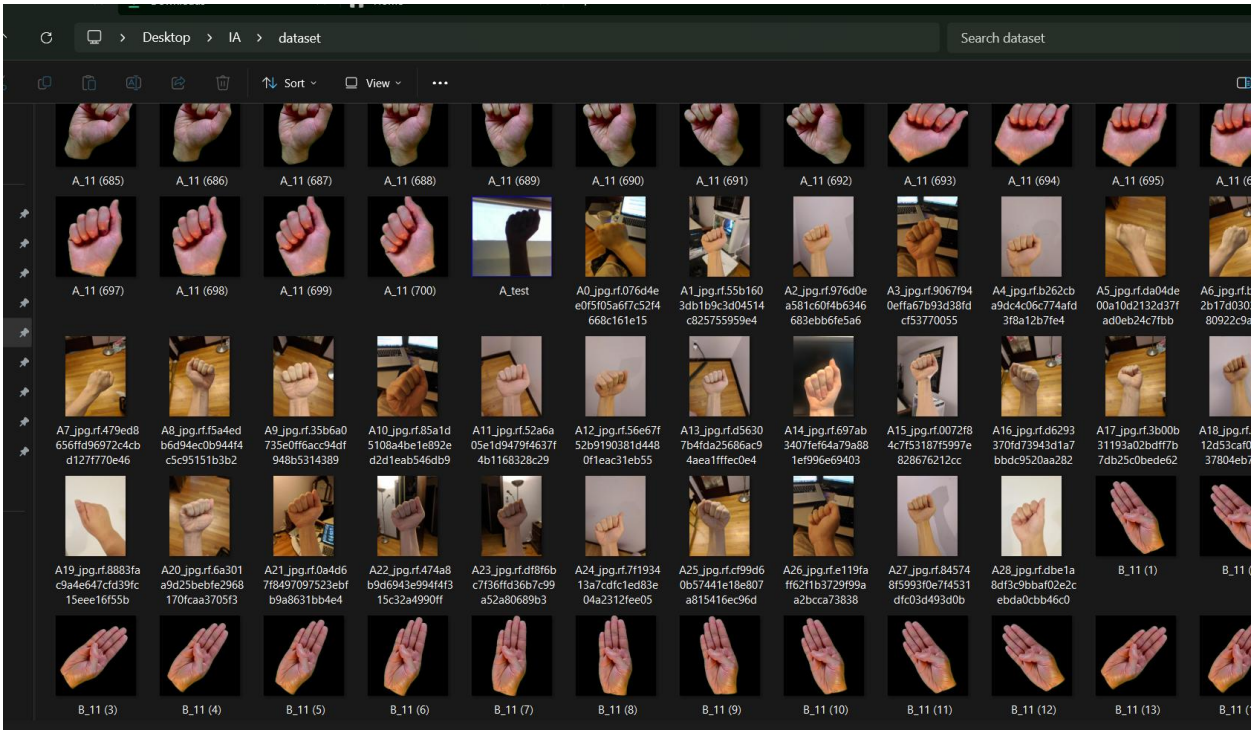
Objetivos Específicos:

- Entrenar y optimizar un modelo de clasificación capaz de reconocer letras del lenguaje de señas a partir de imágenes o puntos clave de la mano.
- Capturar imágenes en tiempo real utilizando una cámara web, aplicando técnicas de preprocesamiento para mejorar la detección.
- Desarrollar una interfaz gráfica intuitiva que muestre la letra reconocida, construya palabras con participación del usuario y permita ejecutar comandos personalizados.
- Evaluar el rendimiento del modelo utilizando métricas como accuracy, recall y F1-score, complementado con visualizaciones como matrices de confusión.

Descripción del dataset utilizado

Se utilizó un dataset estructurado por carpetas llamado DATASET, en donde cada subcarpeta (A, B, ..., Z) contiene imágenes correspondientes a una letra o número. En total se cargaron 900 imágenes distribuidas uniformemente entre las clases.

Cada imagen es una captura estática de una mano haciendo el gesto correspondiente a una letra, en condiciones controladas de iluminación y fondo.



Por otro lado también se utilizo un csv de cada letra que nos ayuda a mejorar el modelo, ya que con ayuda de MediaPie toma la estructura de la mano y con esto puede dar una respuesta mas precisa y eficiente, dentro del modelo, por otro lado también se utilizo redes neuronales y también el modelo se muestra con Tkinter para mostrar una interfaz mas sencilla y sin necesidad de utilizar un frontend.

Preprocesamiento Aplicado

- Conversión a escala de grises.
- Aplicación de ecualización de histograma adaptativa (CLAHE) para mejorar contraste.
- Redimensionamiento a 128x128 píxeles.
- Normalización de valores de píxeles entre 0 y 1.
- División de los datos: 80% entrenamiento, 20% validación.
- Aumento de datos con rotación, desplazamiento, zoom y voltear horizontalmente.

Implementación del Modelo

Se eligió una red neuronal convolucional (CNN) por su capacidad para aprender representaciones espaciales en imágenes. Se implementó una arquitectura ligera de dos capas convolucionales con batch normalization y dropout, ideal para datasets pequeños. El modelo fue entrenado con categorical_crossentropy y optimizador Adam.

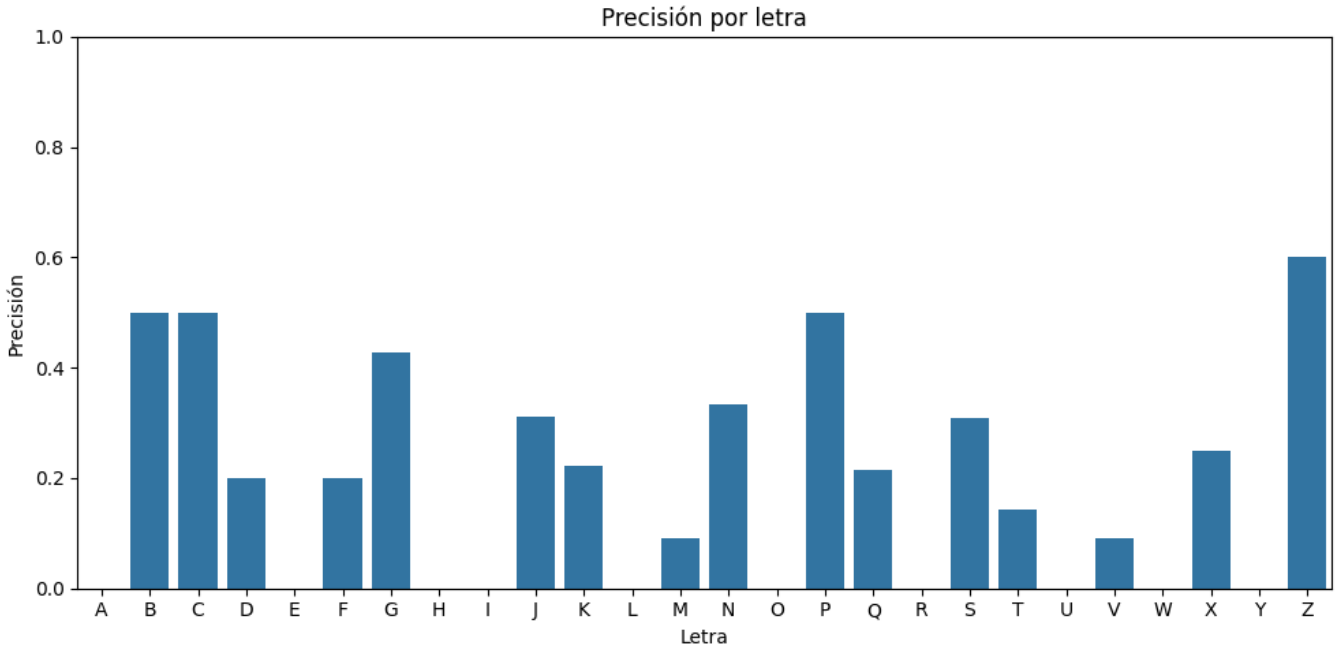
Justificación del algoritmo:

- Las CNN son estándar para tareas de visión por computadora.
- Se priorizó una arquitectura reducida debido al tamaño limitado del dataset.

Evaluación del Modelo y Análisis de Resultados

Precisión por letra

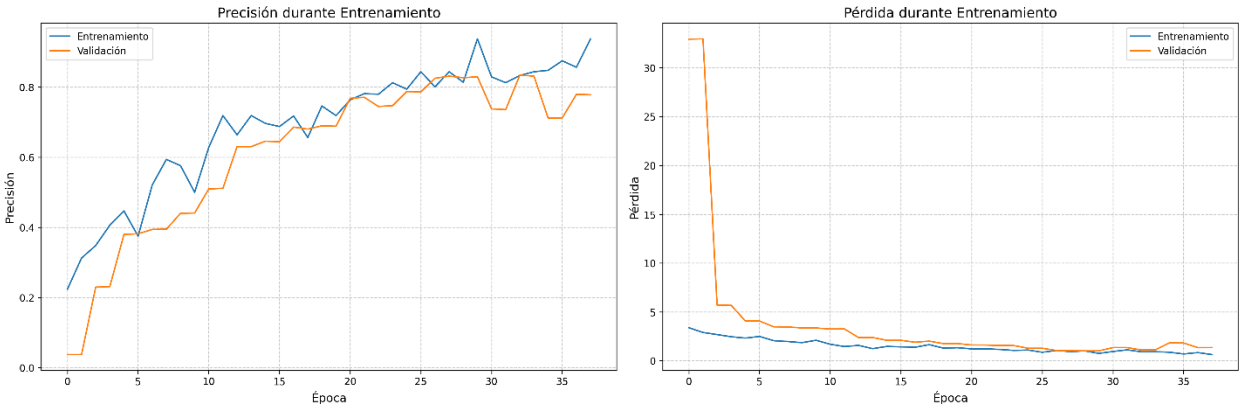
Esta gráfica muestra la precisión individual alcanzada por el modelo para cada letra del alfabeto. Permite identificar qué letras son más fáciles de reconocer por la red neuronal y cuáles presentan mayores errores.



Precisión y pérdida durante el entrenamiento

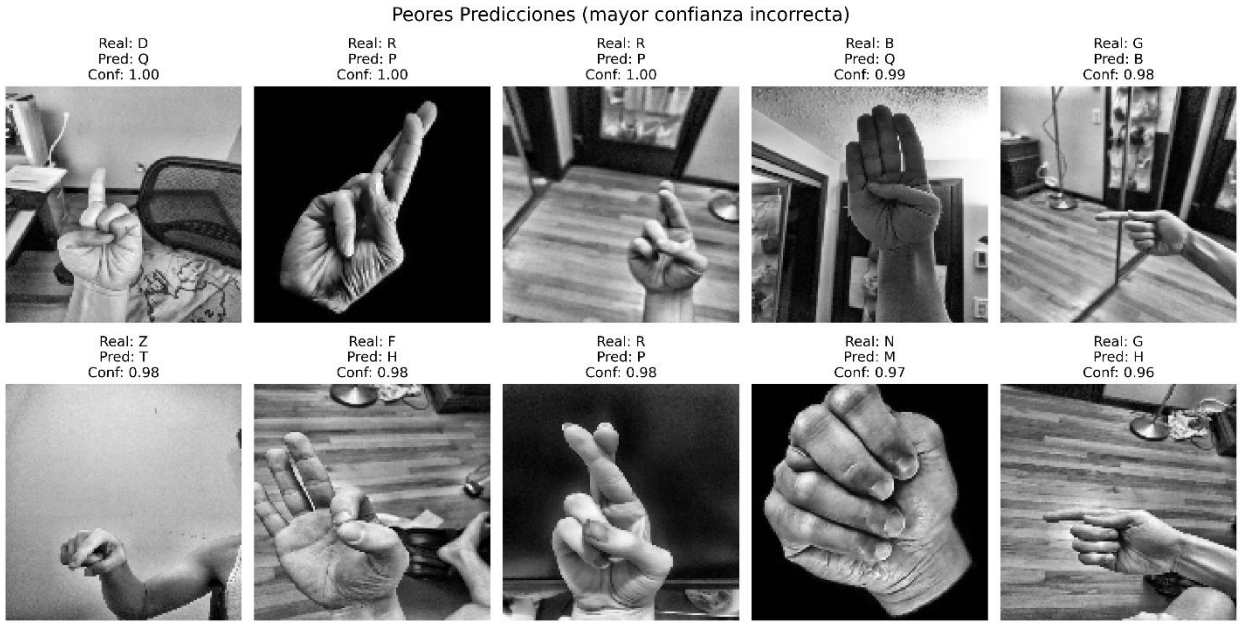
Este gráfico está dividido en dos subgráficas:

- Izquierda: Muestra cómo evoluciona la precisión (accuracy) tanto en entrenamiento como en validación a lo largo de las épocas. Se observa un incremento estable, lo que indica que el modelo está aprendiendo de forma progresiva.
- Derecha: Representa la función de pérdida (loss) en entrenamiento y validación. Un descenso constante sugiere que el modelo se ajusta bien a los datos sin señales claras de sobreajuste.



Peores predicciones con alta confianza

Este conjunto de imágenes muestra los 10 casos en los que el modelo hizo una predicción incorrecta con mayor nivel de confianza. Estos errores son útiles para analizar falsos positivos críticos y entender posibles confusiones entre letras con gestos similares, como **R y P**, o **G y H**.

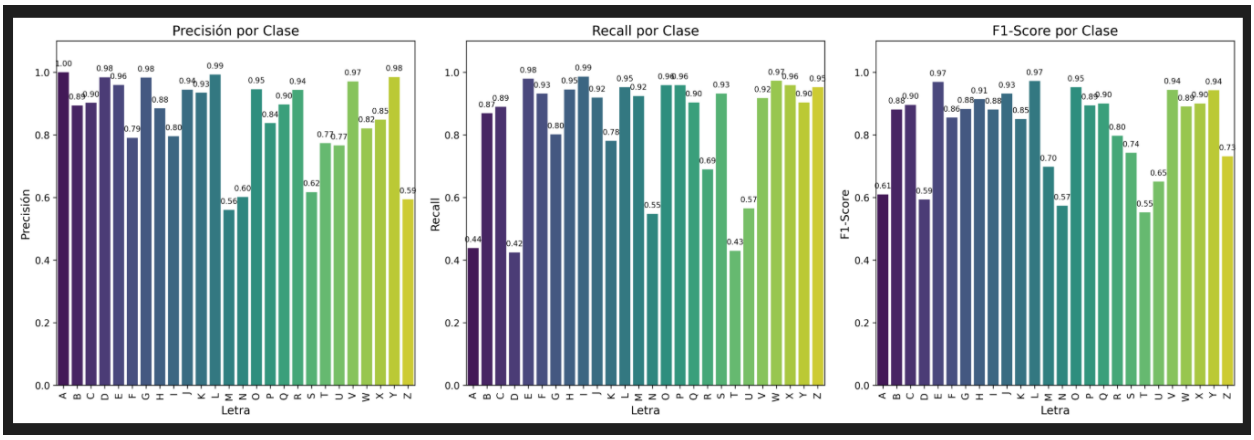


Métricas por clase: Precisión, Recall y F1-Score

Este conjunto de tres gráficas presenta un análisis más detallado por clase:

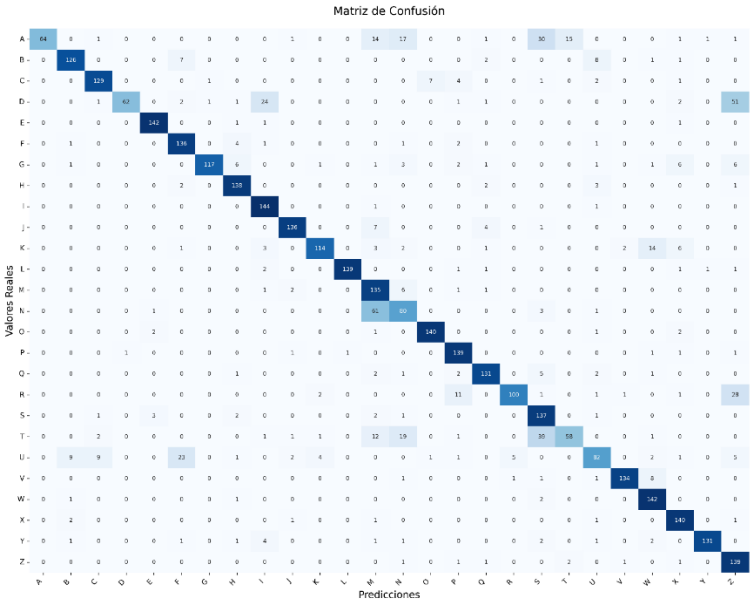
- **Precisión:** Qué tan precisas son las predicciones positivas para cada clase.
- **Recall:** Qué tan bien el modelo recupera los ejemplos verdaderos para cada clase.
- **F1-Score:** Media armónica entre precisión y recall, útil para clases desbalanceadas.

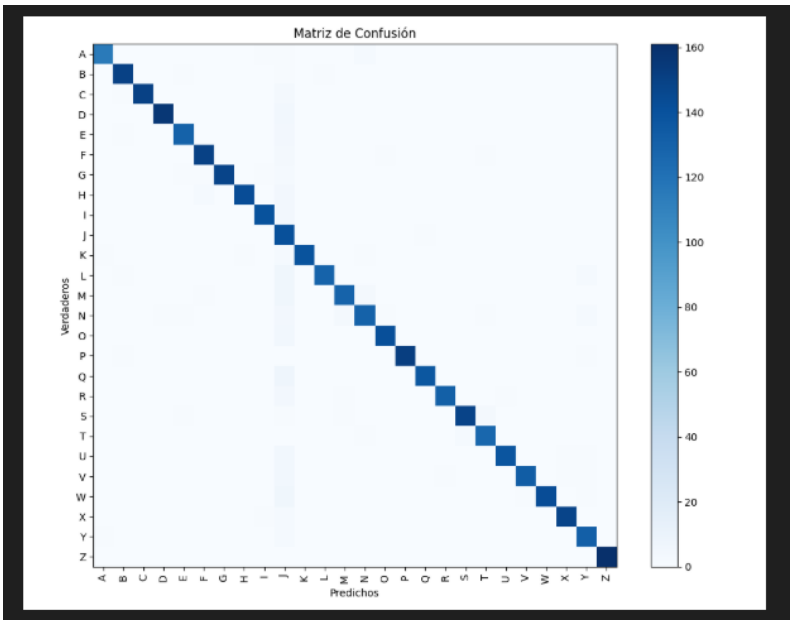
Estas métricas son claves para evaluar el rendimiento del modelo de forma más robusta que con precisión global.



Matriz de Confusión

La matriz de confusión presenta la relación entre predicciones y valores reales. Cada celda muestra cuántas veces una letra fue confundida con otra. Las celdas en la diagonal representan predicciones correctas. Este recurso visual permite identificar patrones de confusión sistemáticos entre ciertas letras, lo cual puede indicar similitud visual o problemas en los datos de entrenamiento.

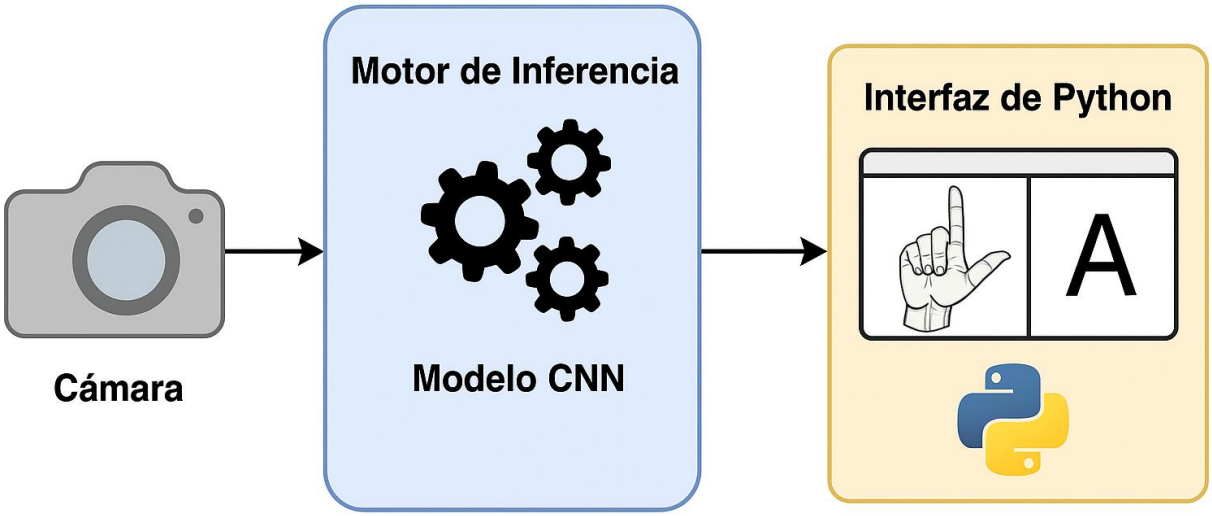




Este es la matriz de confusión de nuestro ultimo intento, lo que se puede ver es que salió una diagonal que significa que nuestro modelo esta en un equilibro con las palabras

Diagramas

- Arquitectura del sistema (modelo CNN + interfaz Python + cámara)



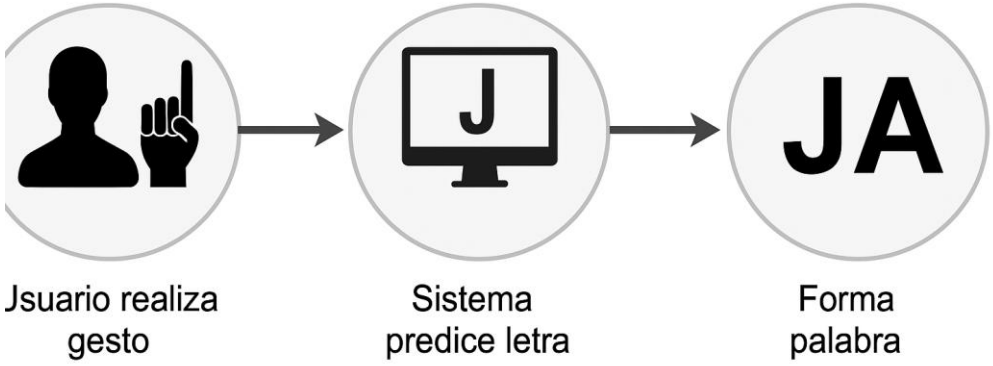
Arquitectura del sistema

- Casos de uso: Usuario realiza gesto → sistema predice letra → forma palabra

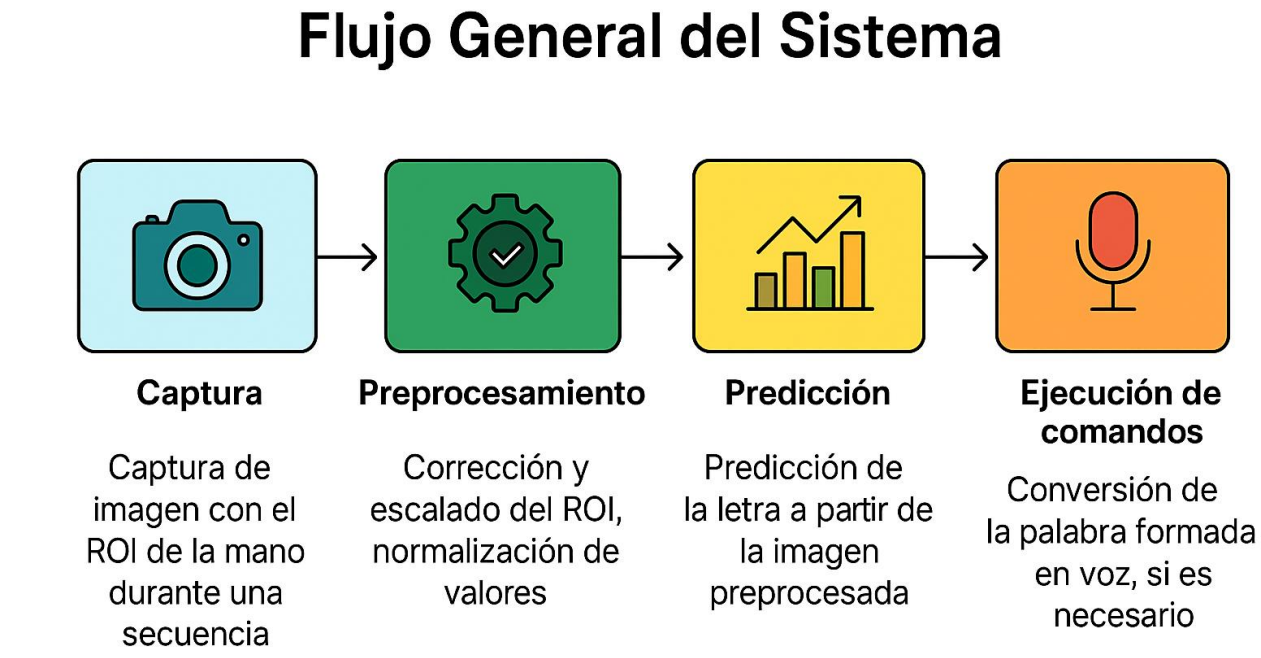
Diagrama de Casos de Uso: Reconocimiento de Señas y Formación de Palabras

Este diagrama representa el flujo básico de interacción entre el usuario y el sistema de traducción de lenguaje de señas implementado en Python con un modelo CNN:

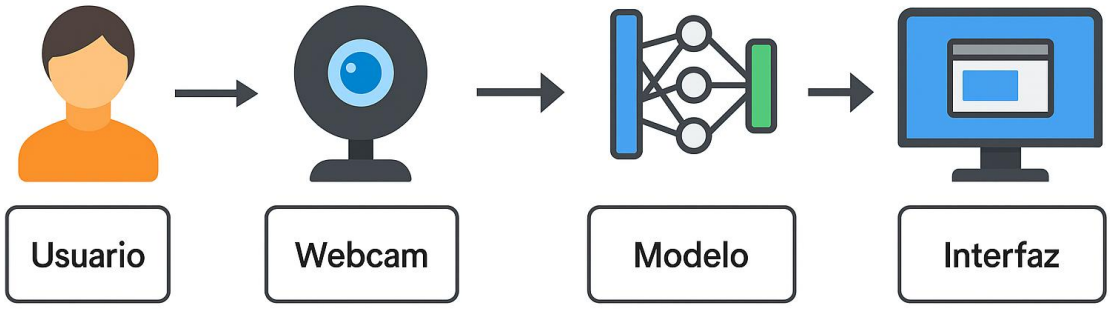
1. Usuario realiza un gesto con la mano
   El usuario se coloca frente a la cámara y realiza una seña correspondiente a una letra del alfabeto.
2. El sistema captura el gesto a través de la cámara
   Se utiliza OpenCV para detectar el ROI (Región de Interés) de la mano. Opcionalmente, MediaPipe se encarga de identificar la posición de los puntos clave (landmarks) para mejorar el recorte.
3. El modelo CNN realiza la predicción
   La imagen preprocesada se pasa al modelo de red neuronal convolucional previamente entrenado, que predice a qué letra corresponde el gesto.
4. Se muestra la letra en la interfaz y se forma la palabra
   La letra reconocida se visualiza en pantalla, y si el usuario lo desea, puede presionar una tecla para agregarla a la palabra en construcción. También puede borrar letras o limpiar la palabra.
5. El sistema puede leer la palabra o ejecutar comandos
   Si el usuario completa una palabra y presiona la barra espaciadora, el sistema verifica si esa palabra está mapeada a una acción en el archivo word_dict.json. Si es así, se ejecuta el comando correspondiente o se usa síntesis de voz para decir la palabra.



- Flujo general: Captura → Preprocesamiento → Predicción → Ejecución de comandos



- Secuencia de interacción: Usuario → Webcam → Modelo → Interfaz



Conclusiones

- Las redes neuronales convolucionales pueden identificar lenguaje de señas con una precisión aceptable aún con pocos datos, si se ajusta bien la arquitectura.
- La calidad del dataset impacta directamente en el desempeño del modelo.
- El preprocesamiento y la simplicidad del modelo fueron claves para generalizar mejor con datos limitados.
- Se evidenció la necesidad de tener mayor volumen y variedad de datos para robustecer el sistema.

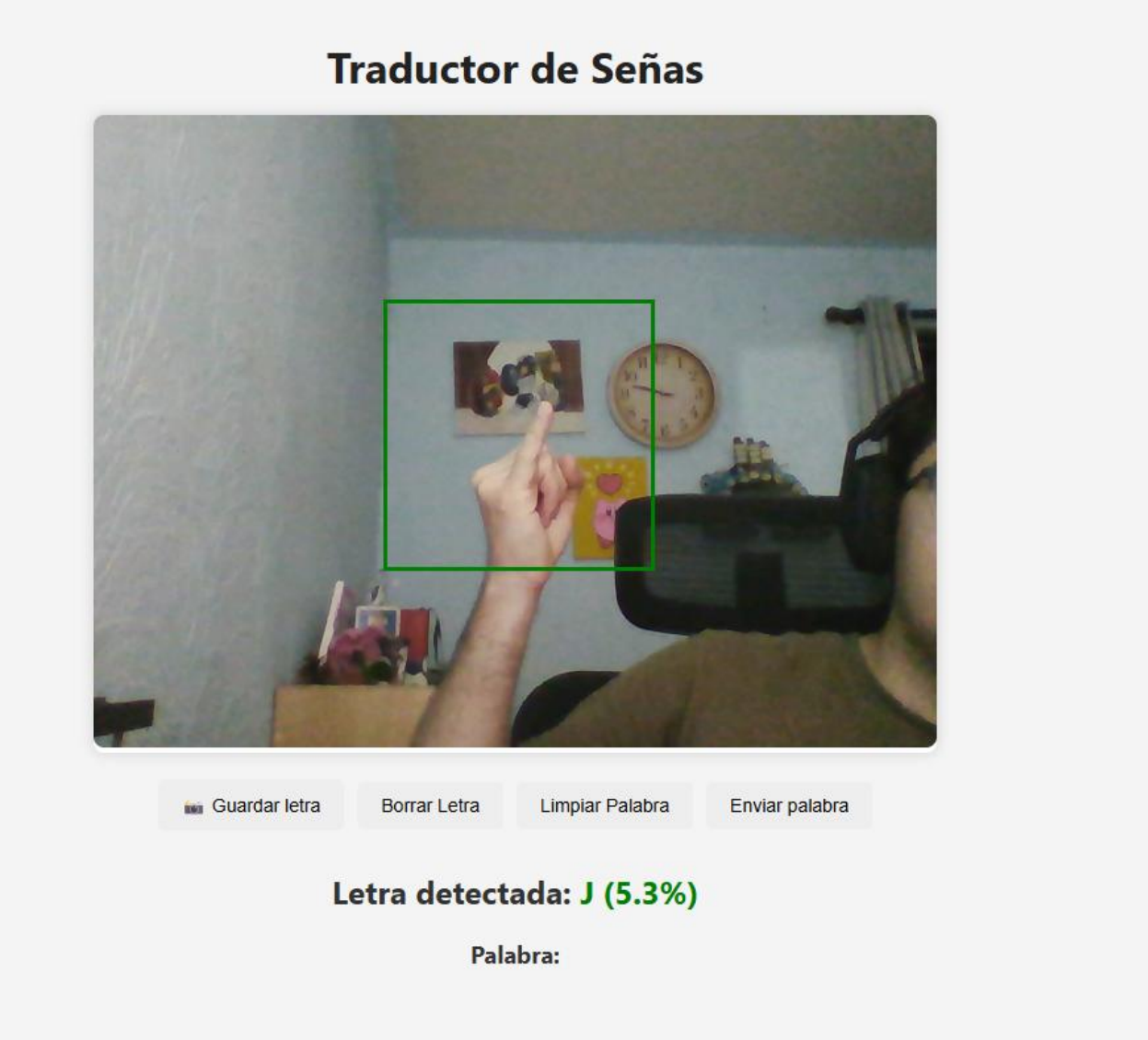
Referencias

- Chollet, F. (2015). *Keras: Deep Learning for humans*. GitHub. https://github.com/keras-team/keras
- Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). TensorFlow: A System for Large-Scale Machine Learning. *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. https://www.tensorflow.org/
- Rosebrock, A. (2018). *OpenCV: Computer Vision Projects with Python*. PyImageSearch. https://pyimagesearch.com/
- MediaPipe. (2023). *Cross-platform, customizable ML solutions for live and streaming media*. Google Developers. https://mediapipe.dev/
- Kaggle. (2020). *Sign Language MNIST Dataset*. Kaggle. https://www.kaggle.com/datasets/datamunge/sign-language-mnist
- Microsoft. (2023). *pyttsx3 Text-to-Speech Conversion Library*. PyPI. https://pypi.org/project/pyttsx3/
- Hunter, J. D. (2007). *Matplotlib: A 2D Graphics Environment*. *Computing in Science & Engineering*, 9(3), 90–95. https://matplotlib.org/
- Waskom, M. L. (2021). *Seaborn: Statistical data visualization*. *Journal of Open Source Software*, 6(60), 3021. https://seaborn.pydata.org/

Anexos

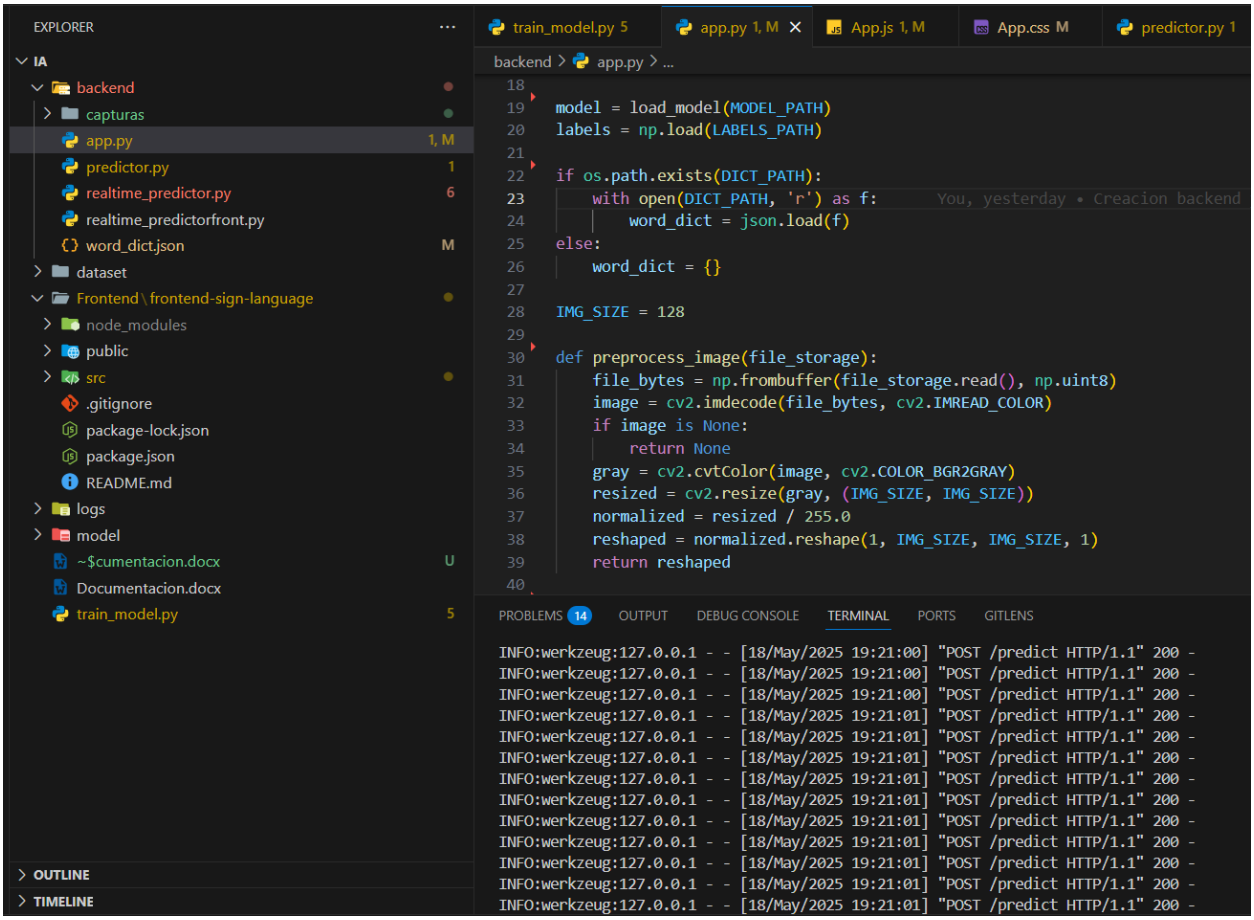
Frontend

El frontend, se Desarrollo en react



Backend

Se utilizó un backend en Python, para poder enviarle la imagen y que responda que letra se encontró.



```python

model = load_model(MODEL_PATH)
labels = np.load(LABELS_PATH)

if os.path.exists(DICT_PATH):
    with open(DICT_PATH, 'r') as f:
        word_dict = json.load(f)
else:
    word_dict = {}

IMG_SIZE = 128

def preprocess_image(file_storage):
    file_bytes = np.frombuffer(file_storage.read(), np.uint8)
    image = cv2.imdecode(file_bytes, cv2.IMREAD_COLOR)
    if image is None:
        return None
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    resized = cv2.resize(gray, (IMG_SIZE, IMG_SIZE))
    normalized = resized / 255.0
    reshaped = normalized.reshape(1, IMG_SIZE, IMG_SIZE, 1)
    return reshaped

```

```
INFO:werkzeug:127.0.0.1 - - [18/May/2025 19:21:00] "POST /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [18/May/2025 19:21:00] "POST /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [18/May/2025 19:21:00] "POST /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [18/May/2025 19:21:01] "POST /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [18/May/2025 19:21:01] "POST /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [18/May/2025 19:21:01] "POST /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [18/May/2025 19:21:01] "POST /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [18/May/2025 19:21:01] "POST /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [18/May/2025 19:21:01] "POST /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [18/May/2025 19:21:01] "POST /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [18/May/2025 19:21:01] "POST /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [18/May/2025 19:21:01] "POST /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [18/May/2025 19:21:01] "POST /predict HTTP/1.1" 200 -
```

.py para mejor entendimiento