# **Language Signal Bridging - Utilizando PyTorch y PyQt5**

## **1. Fundamentación**

### **1.1 Introducción al Problema**

La **comunicación** es un aspecto esencial de la vida cotidiana, y para las personas sordas o con dificultades auditivas, el **lenguaje de señas** es una herramienta crucial para interactuar con el mundo. Sin embargo, el **desconocimiento generalizado** del lenguaje de señas por parte de la población oyente genera una barrera significativa en la comunicación. Es por esto que el desarrollo de **sistemas de traducción de lenguaje de señas** a texto en tiempo real representa una solución innovadora para mejorar la accesibilidad e inclusión social.

Este proyecto tiene como objetivo desarrollar un **sistema de traducción automática** del lenguaje de señas a texto utilizando tecnologías modernas como **PyTorch**, **MediaPipe** y **PyQt5**. Este sistema se enfoca en reconocer y traducir tanto **señas estáticas** como **señas dinámicas**, abarcando desde el **alfabeto dactilológico** hasta palabras básicas y números. Esta propuesta no solo busca ser una herramienta inclusiva, sino que también pretende demostrar cómo el **aprendizaje profundo** puede aplicarse en el ámbito de la **interacción humano-computadora** para resolver problemas de accesibilidad.

### **1.2 Justificación Teórica**

El modelo que proponemos se basa en el **Transformer Encoder**, un bloque de la popular arquitectura de **Transformers** introducida por Vaswani et al. en el paper **"Attention is All You Need"**. Los Transformers son conocidos por su capacidad de procesar secuencias y modelar relaciones temporales, lo cual es ideal para la tarea de clasificación de señas, donde es crucial identificar no solo los movimientos individuales de las manos, sino la secuencia completa en la que ocurren.

**Mediapipe**, por su parte, es una herramienta poderosa para la captura de **landmarks corporales y de manos** que facilita el procesamiento de imágenes en tiempo real. Esta librería permite obtener **puntos clave** de las manos y el cuerpo, lo que nos brinda información relevante sobre la posición de los dedos y las articulaciones, necesaria para identificar las señas realizadas.

El uso de técnicas de **Data Augmentation** es clave para mejorar la generalización del modelo. Dado que el conjunto de datos es limitado en comparación con otros problemas de aprendizaje profundo, se aplicaron técnicas como el cálculo de **distancias entre puntos de la mano** y el análisis de **ángulos entre articulaciones**, lo que proporciona características adicionales y enriquecidas para mejorar el rendimiento del modelo.

Este proyecto se enmarca dentro de la temática de **Construcción de sistemas basados en diversidad**, ya que tiene como objetivo mejorar la accesibilidad para las personas sordas y permitir una comunicación más inclusiva entre oyentes y personas sordas.

### **1.3 Aplicación Práctica**

El software que desarrollamos no solo reconoce y traduce señas, sino que está diseñado para ser **fácilmente adaptable** a diferentes entornos de uso. Algunas aplicaciones prácticas incluyen:

* **Aulas educativas**, donde estudiantes sordos pueden interactuar de manera más sencilla con sus compañeros y profesores.
* **Ambientes laborales**, donde la integración de personas con dificultades auditivas sería más fluida.
* **Entornos domésticos**, facilitando la comunicación entre familiares sordos y oyentes.

La propuesta abarca el desarrollo de una **interfaz gráfica intuitiva** que permite a los usuarios iniciar el reconocimiento de señas con un simple clic, seleccionando la cámara adecuada y visualizando los resultados de la predicción en tiempo real.

## **2. Wireframes y Navegabilidad**

### **2.1 Diseño de Interfaces Gráficas**

Las **interfaces gráficas** serán diseñadas con **PyQt5**, proporcionando una estructura modular y escalable para futuras actualizaciones. El diseño sigue los principios de accesibilidad, asegurando que las opciones sean visibles y que la interacción sea sencilla para todos los usuarios.

### **Pantallas Principales:**

1. **Pantalla Principal:**
   * Un botón grande de **"Iniciar"**, que permite al usuario comenzar la captura de video.
   * Opción para seleccionar la **cámara** (0 para la cámara interna, o selección de cámaras externas).
2. **Pantalla de Captura y Traducción:**
   * Una ventana de video que muestra la transmisión en tiempo real de la cámara.
   * Un cuadro de texto en la parte inferior donde se va concatenando el texto predicho en base a las señas detectadas.
   * Indicaciones visuales del progreso durante la captura de los 30 fotogramas por seña.
3. **Pantalla de Resultados:**
   * Un área donde se muestran las **palabras o frases predichas** a partir de las señas capturadas.
   * Botones de **reiniciar** o **cerrar** el programa.

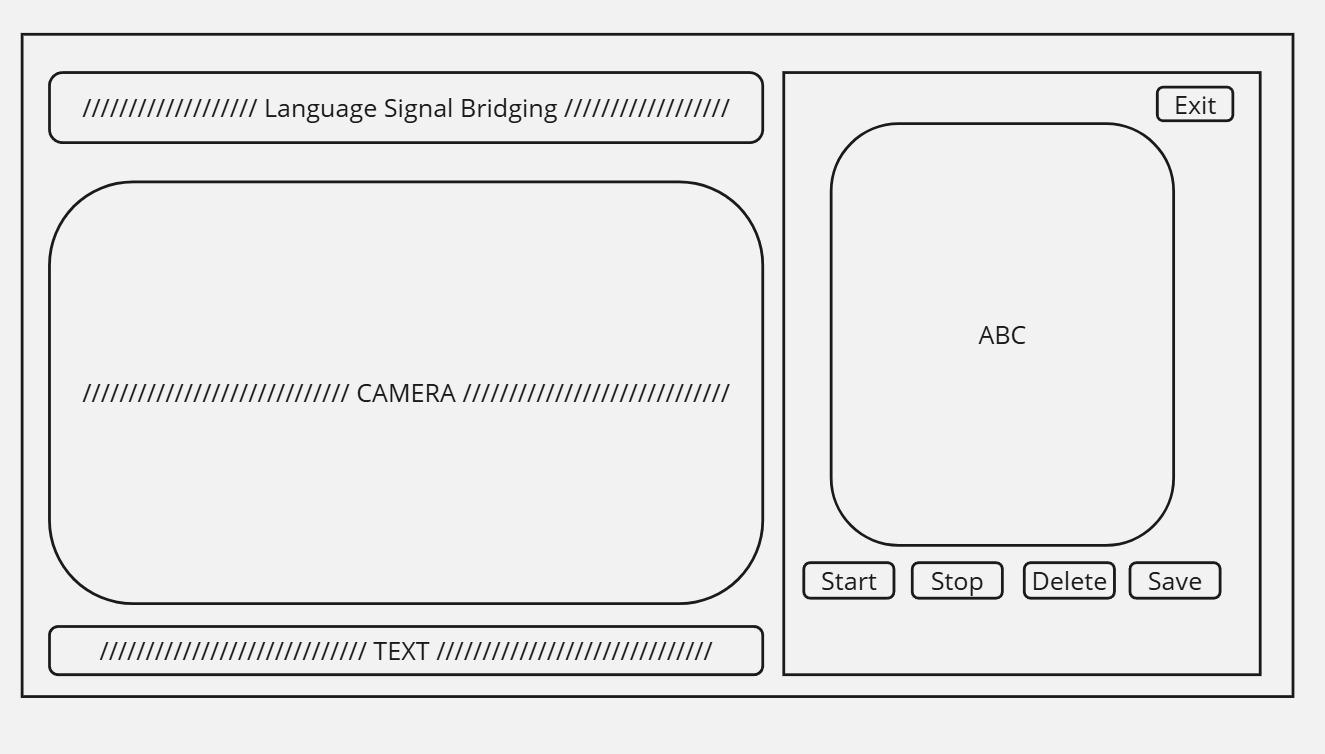
### **2.2 Navegabilidad**

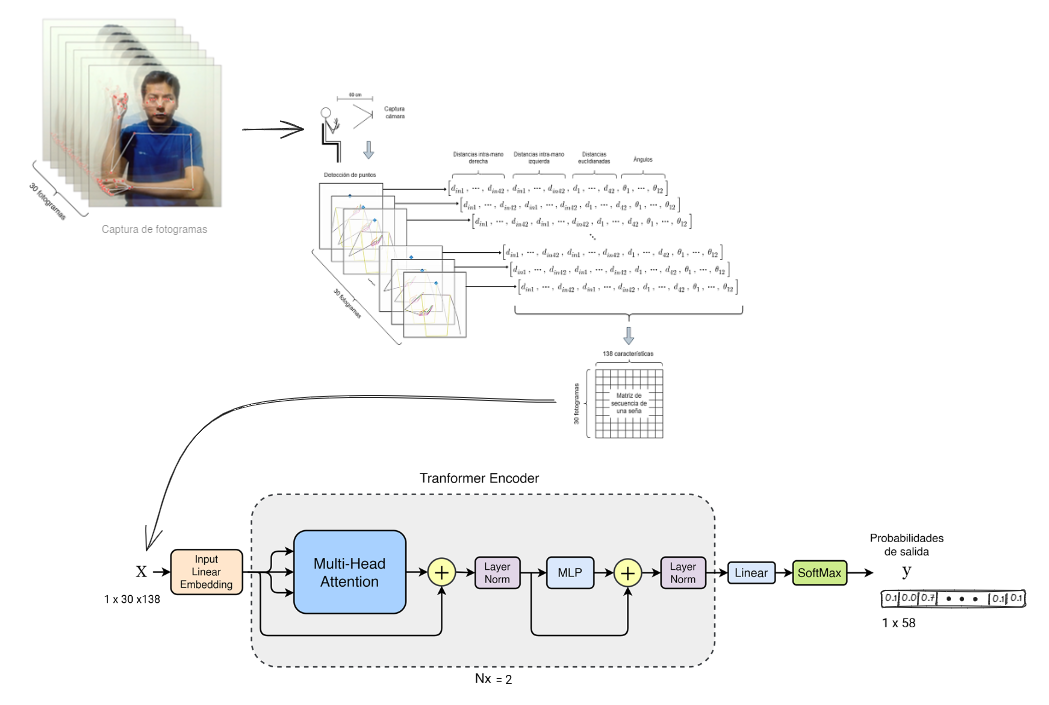
La navegación del programa es sencilla e intuitiva. A continuación, se describe el flujo de interacción con el sistema:

1. El usuario inicia el programa y accede a la **pantalla principal**.
2. Selecciona la cámara y presiona el botón **"Iniciar"**.
3. Se despliega la **pantalla de captura**, que graba los movimientos del usuario durante 30 fotogramas.
4. Una vez capturada la seña, el sistema predice y muestra la letra, número o palabra correspondiente en el área de resultados.
5. El texto predicho se va concatenando en una oración, la cual se puede copiar o guardar para su posterior uso.

### **2.3 Predicción de Diseño Final**

A continuación presentaremos el diseño y el flujo de comportamiento que pre-visualizamos para el desarrollo del software final (siempre sujeto a posibles y futuros cambios):





## **3. Plan de Trabajo**

### **3.1 Fases del Proyecto**

El proyecto se divide en las siguientes fases para asegurar su correcto desarrollo:

1. **Fase 1: Investigación y Planificación (Semana 1)**
   * Revisión de la arquitectura **Transformer**.
   * Inicio de un modelo básico de **reconocimiento** de señas
   * Estudio de **MediaPipe** y su uso en la captura de posiciones.
2. **Fase 2: Entrenamiento del Modelo (Semanas 2)**
   * Preprocesamiento de datos (Data Augmentation).
   * Entrenamiento del modelo utilizando **PyTorch** y **PyTorch Lightning**.
   * Evaluación y ajuste del modelo para mejorar la precisión.
3. **Fase 3: Desarrollo de la Interfaz Gráfica (Semana 3)**
   * Implementación de la interfaz gráfica usando **PyQt5**.
   * Integración de la cámara con **OpenCV** para capturar y visualizar los gestos en tiempo real.
4. **Fase 4: Integración y Pruebas (Semana 4)**
   * Integración del modelo entrenado con la interfaz gráfica.
   * Pruebas del sistema con diferentes usuarios para asegurar su funcionalidad.
   * Elaboración de la documentación técnica final.
   * Optimización del sistema y corrección de errores.
   * Preparación para la entrega final.

### **3.2 Tecnologías a Utilizar**

* **PyTorch** y **PyTorch Lightning**: Para el desarrollo y entrenamiento del modelo de aprendizaje profundo.
* **MediaPipe**: Para la captura de los puntos de referencia de las manos y el cuerpo.
* **PyQt5**: Para el desarrollo de la interfaz gráfica del programa.
* **OpenCV**: Para la captura de video en tiempo real.

## **4. Prototipos y Documentaciones Técnicas**

En esta sección, se proporcionarán los enlaces correspondientes a cada fase de desarrollo del proyecto, organizados por semana. Cada enlace permitirá acceder tanto al **código fuente** como a la **documentación técnica** de las distintas versiones del proyecto, garantizando la trazabilidad del avance y las decisiones tomadas a lo largo del proceso.

### **4.1 Semana 01**

Semana 01: [PGE\_2024/Tarea\_Parcial02/Semana01 at master · GastMolina267/PGE\_2024 (github.com)](https://github.com/GastMolina267/PGE_2024/tree/master/Tarea_Parcial02/Semana01)

Este proyecto combina el reconocimiento de gestos utilizando **MediaPipe** y **OpenCV** con un **juego de Snake** controlado por gestos de la mano. El sistema captura en tiempo real los gestos de la mano utilizando la cámara, permite el movimiento de la ventana de la aplicación con un gesto de mano abierta, y controla la serpiente del juego de Snake con el dedo índice.

El objetivo principal es demostrar cómo las tecnologías de visión por computadora pueden integrarse para controlar aplicaciones de manera fluida y precisa, haciendo uso de gestos simples y accesibles.

**Funcionalidades Principales:**

1. **Detección de Gestos con MediaPipe**: Se captura la posición de la mano y los dedos en tiempo real, permitiendo el control del Snake mediante movimientos del dedo índice. Los gestos reconocidos incluyen:
   * **Gesto de mano abierta** para mover la ventana.
   * **Pulgar hacia arriba** para cerrar la aplicación.
   * **Dedo índice levantado** para abrir el juego Snake.
2. **Control del Juego Snake**: El juego se controla moviendo el dedo índice y utiliza un promedio móvil de las últimas posiciones del dedo para suavizar el control del movimiento. El Snake cambia de dirección en función de la posición del dedo en el área de la pantalla.
3. **Interacción con la Ventana**: El programa permite mover la ventana de la aplicación utilizando gestos de la mano, siempre y cuando el juego Snake no esté activo.

**Requisitos del Proyecto:**

* **Python 3.x**.
* Librerías requeridas:
  + **OpenCV** para la captura de video.
  + **MediaPipe** para la detección de gestos.
  + **Pygame** para el desarrollo del juego Snake.
  + **PyAutoGUI** y **Win32** para la manipulación de la ventana.

**Instrucciones para Ejecutar el Proyecto:**

1. Clonar el repositorio y asegurarse de instalar las dependencias necesarias.
2. Ejecutar el script principal para activar la cámara y detectar los gestos.
3. Controlar el Snake con los movimientos del dedo índice y utilizar los gestos para manipular la ventana.

**Controles del Juego Snake:**

* **Dedo índice levantado**: Inicia el juego de Snake.
* **Mover el dedo índice**: Controla la dirección del Snake (izquierda, derecha, arriba, abajo).
* **Mano abierta**: Mueve la ventana de la aplicación (solo si el juego Snake no está activo).
* **Pulgar hacia arriba**: Cierra la aplicación.

**Detalles Técnicos:**

* El control del Snake se basa en la posición del dedo índice, capturada por **MediaPipe**. Las coordenadas del dedo se actualizan continuamente y el sistema mantiene un promedio móvil de las últimas posiciones del dedo, lo que garantiza un movimiento más fluido y estable.
* **Pygame** es utilizado para manejar la lógica del juego Snake, incluyendo la generación de la serpiente y la comida, la detección de colisiones, y el manejo de Game Over.

**Futuras Mejoras:**

1. Ampliar la capacidad de detección de gestos para controlar otras funcionalidades dentro y fuera del juego.
2. Optimizar la interfaz gráfica y encaminar a la idea original del proyecto Language Signal Bridging.

### **4.2 Semana 02**

Semana02: [PGE\_2024/Tarea\_Parcial02/Semana02 at master · GastMolina267/PGE\_2024 (github.com)](https://github.com/GastMolina267/PGE_2024/tree/master/Tarea_Parcial02/Semana02)

Durante la **semana 02** de desarrollo de nuestro proyecto, el enfoque principal fue evaluar la **resistencia** y **reacción** del sistema a los eventos generados por el reconocimiento de señas en un contexto interactivo. Decidimos aplicar este concepto a un juego clásico: **Piedra, Papel, Tijeras**, utilizando gestos de la mano como entradas para las jugadas.

**Objetivos:**

El objetivo central de esta semana era implementar un **flujo lógico robusto** que pueda manejar la recepción de eventos y asegurarnos de que el sistema reaccione de manera eficiente y precisa a las jugadas del jugador, detectadas mediante señas, frente a las predicciones hechas por la IA del programa.

Este proceso incluyó varios pasos clave:

1. **Recepción y Procesamiento de Señas**:
   * El sistema utiliza la cámara para capturar la imagen de la mano del jugador y procesa los gestos mediante **MediaPipe**.
   * El jugador selecciona su jugada realizando **tres movimientos** (para reforzar la certeza de la elección), y la IA selecciona una opción al azar.
   * Durante la semana, evaluamos cómo el sistema reaccionaba a cada seña, optimizando la detección para que los gestos como **Piedra (mano cerrada), Papel (mano abierta), Tijeras (índice y medio levantados)** fueran correctamente clasificados.
2. **Comparación entre Jugadas**:
   * Tras detectar la seña final del jugador, el sistema compara la jugada contra la selección aleatoria de la IA y muestra el resultado.
   * Evaluamos la lógica de la IA para asegurarnos de que el resultado (ganador, perdedor o empate) fuera correcto en cada ronda.
3. **Resistencia de Eventos**:
   * Este término se refiere a la capacidad del sistema para mantenerse estable y reactivo ante la recepción de múltiples eventos en secuencia rápida (es decir, varios gestos consecutivos en un corto período de tiempo).
   * Queríamos garantizar que el reconocimiento de señas no solo fuera preciso, sino también **consistente**, es decir, que al realizar un gesto de manera repetida, el sistema fuera capaz de identificarlo correctamente en cada instancia.
4. **Pruebas de Flujo**:
   * Probamos cómo el sistema manejaba la interacción del jugador desde que se levanta la mano hasta que se realiza la jugada final.
   * Se evaluó la rapidez y estabilidad del proceso para que el juego fuera fluido y que no hubiera **retrasos innecesarios** que afectaran la experiencia del jugador.

**Conclusiones de la semana:**

El sistema mostró ser **sensible** a los eventos, capturando correctamente los gestos del jugador en la mayoría de las pruebas. Sin embargo, detectamos ciertos problemas en la consistencia de las jugadas si los gestos no se mantenían durante un corto período.

* La lógica de comparación entre la IA y el jugador funcionó de manera correcta, pero también observamos la necesidad de optimizar el tiempo de respuesta entre la identificación de la seña y la ejecución de la acción.

Por lo tanto, el trabajo de esta semana nos permitió comprender mejor las limitaciones del reconocimiento de señas y cómo el sistema responde a eventos consecutivos y rápidos. En las próximas semanas, enfocaremos nuestros esfuerzos en mejorar la **precisión** del reconocimiento de señas y la **resistencia** del sistema ante múltiples eventos, optimizando los tiempos de respuesta.

### **4.3 Semana 03**

Semana03: [PGE\_2024/Tarea\_Parcial02/Semana03 at master · GastMolina267/PGE\_2024 (github.com)](https://github.com/GastMolina267/PGE_2024/tree/master/Tarea_Parcial02/Semana03)

Durante la **Semana 03** del proyecto, nuestro objetivo principal fue introducir los conceptos fundamentales de **PyTorch** y empezar a trabajar con modelos de **aprendizaje automático** y **deep learning**.

**Objetivo clave:**

1. **Familiarización con PyTorch**: Iniciar con la implementación de un modelo basado en redes neuronales utilizando PyTorch. Esta introducción cubrió la estructura general de un modelo, el manejo de tensores y los aspectos esenciales del framework.
2. **Desarrollo del agente IA**: Utilizamos **aprendizaje por refuerzo** para que la IA pudiera tomar decisiones sobre el movimiento de la serpiente en función del estado del entorno (posición de la serpiente, ubicación de la comida, etc.).
3. **Evaluación de eventos y lógica**: El propósito era evaluar cómo la IA reaccionaba a las diferentes condiciones del entorno, determinando si el agente estaba aprendiendo de manera efectiva. Se utilizaron episodios repetidos para ajustar el comportamiento y refinar las decisiones de la IA.
4. **Exploración de Q-Learning**: Implementamos un enfoque de **Deep Q-Network (DQN)** para permitir que la IA aprenda de las recompensas y sanciones asociadas con sus acciones en el juego. La memoria de repetición fue un componente clave, permitiendo al modelo recordar y aprender de experiencias pasadas.

**Resistencia de Eventos y Respuesta del Modelo:**

Evaluamos cómo el modelo reaccionaba a una serie de eventos mientras jugaba Snake. El modelo recibió feedback sobre la efectividad de sus acciones (recoger comida o evitar colisiones). En cada paso del juego, registramos si la IA lograba cumplir con los objetivos de maximizar el tiempo de vida y recoger la mayor cantidad de comida posible.

A medida que avanzaba el entrenamiento, se observaron mejoras en la resistencia de los eventos, es decir, la capacidad de la IA para mantenerse más tiempo en el juego y reducir la frecuencia de colisiones.

**Configuración del proyecto:**

El modelo de IA fue entrenado con un enfoque en la **política ε-greedy**, que se utiliza para equilibrar la **exploración** y **explotación** de acciones durante el juego. Además, se generaron gráficos para visualizar la evolución del rendimiento del agente a lo largo de los episodios, ayudándonos a entender mejor los patrones de aprendizaje.

### **4.4 Semana 04**

[PGE\_2024/Tarea\_Parcial02/Semana04 at master · GastMolina267/PGE\_2024](https://github.com/GastMolina267/PGE_2024/tree/master/Tarea_Parcial02/Semana04)

#### **Introducción**

Durante la última semana del proyecto, nos enfocamos en la implementación de un **software traductor de lengua de señas a texto** utilizando las tecnologías **PyTorch** y **PyQt5**. Este desarrollo forma parte del proyecto de traducción de lenguaje de señas argentino, y se basa en el uso de **aprendizaje profundo** para clasificar secuencias de movimientos de las manos y el cuerpo, capturados mediante la cámara del ordenador. Además, hemos explorado los conceptos fundamentales de **PyTorch** y el inicio del **Deep Learning** para llevar a cabo esta tarea.

#### **Especificaciones del Proyecto**

1. **Tamaño del conjunto de datos:** El conjunto de datos utilizado para entrenar el modelo consistió en **5800 secuencias** capturadas mediante **MediaPipe**. Estas secuencias contienen información clave sobre la posición de las manos y la pose corporal de una persona haciendo diversas señas, tanto estáticas como dinámicas. Cada secuencia consta de múltiples puntos clave que describen el movimiento y la configuración de la mano y el cuerpo.
2. **Modelo y framework utilizados:** El modelo de aprendizaje profundo fue desarrollado utilizando el framework **PyTorch** y la librería **PyTorch Lightning** para simplificar la implementación y entrenamiento del modelo. Para realizar la clasificación de secuencias, empleamos el bloque **Transformer Encoder**, basado en la arquitectura de **Transformers** descrita en el artículo **Attention Is All You Need**.
   * **Data Augmentation**: Durante el preprocesamiento de los datos, no se entrenó el modelo directamente con las coordenadas brutas de los puntos capturados. En su lugar, se utilizó **data augmentation** para generar más características útiles, como distancias entre puntos de las manos y ángulos formados por la configuración de las manos y el cuerpo. Este proceso mejoró la capacidad del modelo para captar relaciones importantes entre los movimientos de las señas.
3. **Interfaz de usuario:** La **interfaz gráfica** fue desarrollada utilizando **PyQt5**, lo que permitió la creación de una aplicación sencilla y funcional que permite interactuar con el modelo de predicción de señas. La interfaz permite seleccionar la cámara que se utilizará (ya sea la interna del ordenador o una cámara externa conectada), y puede ser actualizada a la versión 6 de PyQt en futuras implementaciones.

#### **Funcionamiento del Programa**

El programa traductor de señas a texto tiene un funcionamiento sencillo y eficiente que puede resumirse en los siguientes pasos:

1. **Inicio**: El programa se inicia pulsando el botón **Iniciar**. Luego, el usuario selecciona la cámara que utilizará para capturar las señas; un valor de 0 indica que se usará la cámara interna del ordenador, y otros valores indican una cámara externa conectada.
2. **Captura de secuencias**: Una vez seleccionada la cámara, el programa captura una secuencia de **30 fotogramas** de la persona realizando una seña. El sistema soporta tanto señas **estáticas** (como letras individuales) como **dinámicas** (señas que involucran movimiento a través del tiempo).
3. **Predicción**: Después de capturar los 30 fotogramas, los datos se envían al modelo para que realice la **predicción** de la seña observada. El modelo procesa las secuencias utilizando el bloque **Transformer Encoder** para identificar la letra, número o palabra representada por la seña.
4. **Salida**: El resultado de la predicción se muestra debajo de la pantalla de captura en la interfaz gráfica. El texto resultante se **concatena** automáticamente para formar oraciones más largas, permitiendo así una traducción fluida de frases completas en lenguaje de señas.

## **5. Conclusión**

Este proyecto de reconocimiento y traducción de lenguaje de señas a texto es una solución innovadora que busca **romper las barreras de comunicación** entre personas sordas y oyentes. Utilizando tecnologías modernas y técnicas avanzadas de procesamiento de imágenes y aprendizaje profundo, el sistema ofrecerá una herramienta poderosa para facilitar la interacción en diversos contextos. A lo largo de este proceso, se demuestra cómo los conceptos de **programación genérica**, **manejo de eventos** y **aprendizaje profundo** se aplican a problemas del mundo real para crear soluciones accesibles e inclusivas.

Durante estas semanas, logramos integrar múltiples tecnologías para desarrollar un programa funcional que utiliza **aprendizaje automático** y **visión por computadora** para traducir señas en tiempo real. Este proyecto proporciona una base sólida para futuras mejoras, como la incorporación de un vocabulario más extenso o la mejora del rendimiento del modelo. La experiencia obtenida durante esta última semana nos ha permitido afianzar conceptos clave sobre **aprendizaje profundo** y su aplicación práctica en **traducción de lenguaje de señas**.

## **6. Referencias**

* Vaswani, A., et al. (2017). **Attention is All You Need**.
* MediaPipe: **Framework para la captura de landmarks**.
* PyTorch: **Framework de aprendizaje profundo**.