SiLant (The Silent Assistant)

Proyecto IABD

Pablo Vacas García

Ángel Talavera Garrido

Juan Manuel García Moyano

Índice de contenidos

1. SiLant

SiLant es un proyecto enfocado en la interpretación de lenguaje no verbal orientado a personas con discapacidad en el habla. El proyecto combina modelos de aprendizaje profundo con técnicas de visión por computadora para detectar y clasificar gestos manuales y dirección de la mirada. Incluye una API desarrollada con Django para el procesamiento y predicción de gestos, y una aplicación que permite la detección en tiempo real desde una cámara.

El propósito principal de SiLant es proporcionar una interfaz alternativa de comunicación que permita a personas con impedimentos del habla interactuar con dispositivos inteligentes utilizando únicamente gestos visuales.

A. Modelo de negocio:

El modelo de negocio de SilAnt se basa en un sistema de **suscripción mensual** que permite a los usuarios acceder al servicio sin coste inicial por los dispositivos, los cuales se entregan como parte del plan contratado. Esta suscripción incluye tanto el hardware necesario (como cámaras y pantallas inteligentes) como el acceso a actualizaciones automáticas y soporte técnico continuo, garantizando así una experiencia fluida y siempre actualizada.

La comercialización se realiza principalmente a través de venta online en la página web oficial, permitiendo una distribución directa y eficiente. Además, se contempla la presencia en tiendas especializadas de tecnología y seguridad, con el objetivo de ofrecer demostraciones del producto y cerrar alianzas estratégicas con distribuidores del sector. También se podría incluir el servicio en colaboraciones con empresas de seguridad y alarmas, con gestos personalizados para llamar a emergencias, por ejemplo.

Este enfoque multicanal facilita la adopción del sistema por parte de clientes que valoran tanto la comodidad del canal digital como el contacto directo con el producto.

B. Público Objetivo:

SiLant está diseñado para atender a diversos segmentos de usuarios que pueden beneficiarse significativamente de una interfaz de comunicación basada en gestos. A continuación, se describen los principales grupos objetivo:

1. **Personas con discapacidad en el habla**
   El público principal son individuos con discapacidades que afectan su capacidad de

comunicación verbal, incluyendo personas con parálisis cerebral, esclerosis lateral amiotrófica (ELA), afasia, autismo no verbal, lesiones cerebrales traumáticas o que se estén recuperando de accidentes cerebrovasculares. SiLant les ofrece una herramienta accesible y adaptable para interactuar con su entorno digital y doméstico.

2. **Familias y cuidadores**
   Los familiares y cuidadores de personas con discapacidad encontrarán en SiLant un recurso que facilita la comunicación cotidiana, reduciendo la frustración y mejorando la calidad de vida en el hogar. Además, los gestos personalizados permiten cubrir necesidades específicas, como pedir asistencia, controlar el entretenimiento o gestionar rutinas diarias.

3. **Instituciones educativas inclusivas**
   Escuelas y centros educativos que integran alumnos con necesidades especiales pueden utilizar SiLant como herramienta pedagógica y de integración, facilitando que los estudiantes interactúen con sus compañeros, profesores y el entorno escolar de forma más autónoma.

4. **Centros médicos y de rehabilitación**
   Clínicas, hospitales, centros de rehabilitación y logopedas pueden emplear SiLant como apoyo en terapias de rehabilitación del habla y la comunicación. El uso de gestos puede acelerar procesos de recuperación o servir como puente mientras los pacientes desarrollan o recuperar habilidades verbales.

5. **Empresas de seguridad y servicios de emergencia**
   SiLant puede integrarse en soluciones de seguridad doméstica o institucional, permitiendo a usuarios con movilidad o comunicación reducida emitir señales de alerta o activar alarmas mediante gestos predefinidos. Esta función resulta especialmente útil en hogares de personas mayores o con discapacidad, donde la autonomía y la seguridad son prioritarias.

6. **Distribuidores de tecnología asistiva**
   Tiendas y empresas especializadas en tecnología para la accesibilidad representan un canal estratégico para la expansión del producto. Estos actores no solo pueden facilitar la distribución, sino también realizar demostraciones y adaptar las soluciones a diferentes perfiles de usuario.

2. Metodología

A. Recolección de imágenes:

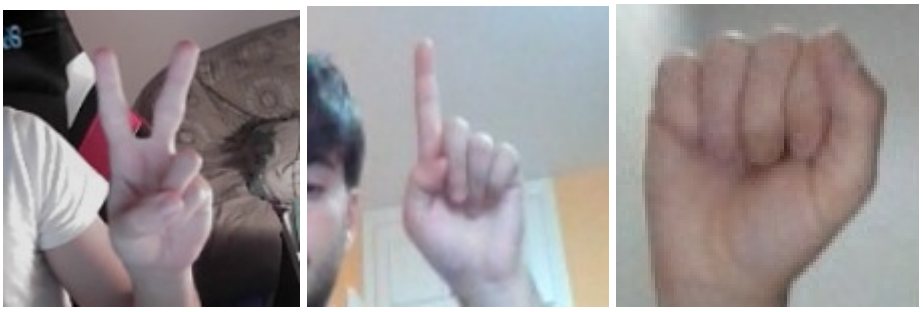
- La recopilación de datos se realizó utilizando la plataforma web **Teachable Machine** de Google. Esta herramienta permite la captura de imágenes desde la cámara del dispositivo.

- Mediante esta web, obtenemos todos los gestos que necesitamos para la elaboración del proyecto y los almacenamos en una carpeta para ser recortadas posteriormente.

- Todas las imágenes de cada gesto, estarán a su vez almacenadas en una carpeta con el nombre de dicho gesto. Esta organización facilita la gestión del dataset y la posterior preparación de los datos para el entrenamiento del modelo.

B. Preparación de Datos:

- Una vez recopiladas las imágenes, se procedió al preprocesamiento mediante un script desarrollado en Python. Este script utiliza técnicas de segmentación de imágenes para detectar y recortar automáticamente la región que contiene la mano del usuario, en caso de haber varias, cogerá la más alta, para evitar confusiones, reduciendo así el ruido visual y centrando la información relevante. Por ejemplo:



- Las imágenes recortadas se almacenan en carpetas organizadas por gesto en la ruta (`Images/train`), permitiendo una estructuración eficiente para su uso en el entrenamiento supervisado.

C. Entrenamiento de Modelos:

- Se implementaron scripts específicos para dividir los datos en conjuntos de entrenamiento y validación, entrenar el modelo, y generar métricas visuales como la función de pérdida y la precisión durante el entrenamiento (`entrenar_modelo.py`).

- El proceso de entrenamiento se realizó mediante redes neuronales convolucionales (CNN), elegidas por su eficacia en tareas de clasificación de imágenes. Los modelos fueron desarrollados utilizando **TensorFlow/Keras**, librerías ampliamente utilizadas por su versatilidad, documentación y soporte en la comunidad de desarrolladores.

- Cada modelo recibe imágenes pequeñas en escala de grises de tamaño 48x48 con un solo canal. Utiliza varias capas convolucionales con filtros de tamaño 3x3 para detectar patrones visuales, comenzando con 32 filtros y aumentando hasta 128, lo que le permite identificar desde características simples hasta más complejas.

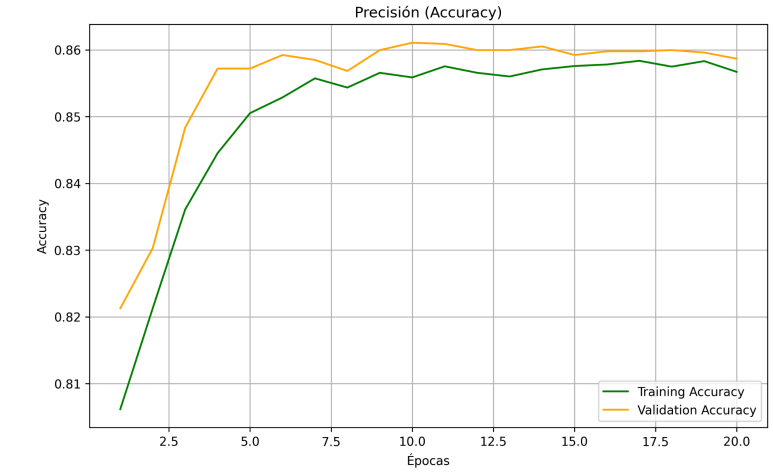
  Todas las capas convolucionales usan la función de activación ReLU, que mejora la capacidad de aprendizaje del modelo al introducir no linealidad. Para reducir el tamaño de los datos y mantener la información más relevante, se aplican capas de agrupamiento (MaxPooling) con un tamaño de 2x2, lo que reduce a la mitad las dimensiones espaciales.

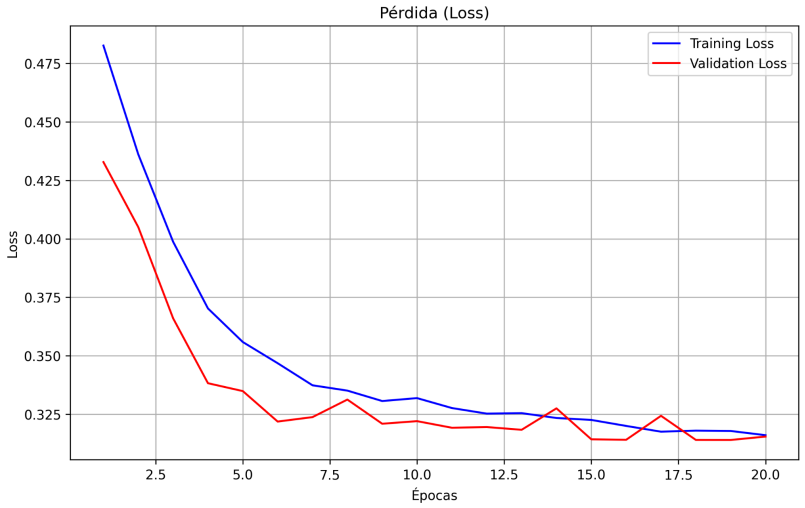
  Además, se emplean capas Dropout con valores de 0.25 y 0.5 para evitar el sobreajuste, desactivando aleatoriamente algunas neuronas durante el entrenamiento.

  Después de extraer las características, se aplanan y pasan por una capa densa de 1024 neuronas, seguida de una capa de salida con una única neurona y activación sigmoide, lo que permite generar una probabilidad y realizar una clasificación binaria.

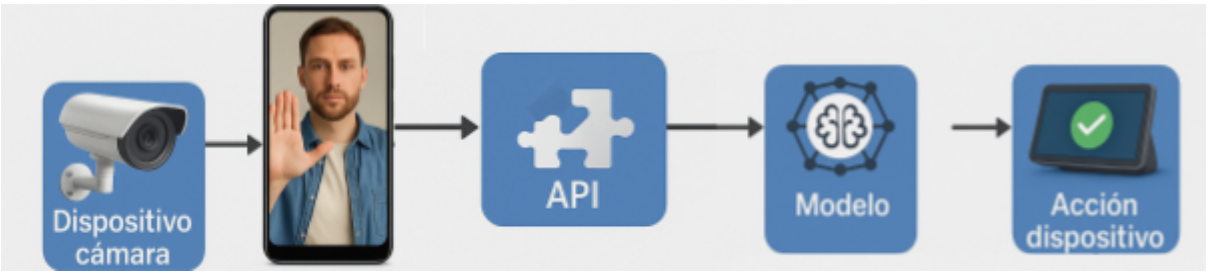
  Finalmente, el modelo se compila con el optimizador Adam, la función de pérdida binary_crossentropy y utiliza la precisión como métrica de evaluación.

- Los modelos se guardaron en la carpeta `Modelos/` y también se generan las gráficas de pérdida y acierto para tener un registro de cada uno y compararlos. Por ejemplo, estas son las gráficas de acierto y pérdida del último modelo entrenado para el gesto Down:





D. Aplicación y API:



La interfaz de usuario en tiempo real es una aplicación encargada de capturar el video desde la cámara del usuario, enviar el frame a la API y realizar la acción definida para ese gesto. Este programa utiliza la cámara del ordenador para reconocer gestos de la mano y detectar si el usuario está mirando a la pantalla. Su

objetivo es permitir el control de ciertas acciones, como subir o bajar el volumen, pausar la música o abrir aplicaciones como YouTube y Spotify, todo sin necesidad de tocar nada.

El funcionamiento general es el siguiente:

1. **Captura de vídeo en tiempo real:** Se usa la cámara para obtener imágenes en vivo.

2. **Detección del rostro y la mirada:** El sistema verifica si el usuario está mirando directamente a la cámara mediante **MediaPipe** para evitar que sea manipulada de forma no intencionada. Hemos elegido este en concreto, debido a su rendimiento en tareas de visión por computadora.

3. **Reconocimiento de gestos con la mano:** Cuando se confirma que el usuario está mirando, se toma una imagen y el envío de la imagen a la API se realiza mediante un hilo secundario (asincronía), consiguiendo que el hilo principal no se quede pillado y el vídeo no se vea cortado. Para finalizar, la API predice qué gesto está haciendo y devuelve el gesto predicho.

4. **Ejecución de acciones:** Según el gesto detectado, se realiza una acción como controlar la música o abrir una aplicación.

5. **Menú por gestos:** Si se hace un gesto específico (como “abrir”), se muestra un pequeño menú en pantalla para que el usuario elija otra acción también con gestos. Por ejemplo, hace el gesto “Abrir” y luego selecciona con gestos la opción 2, Youtube.

Asimismo, se integró **OpenCV** para el procesamiento de imágenes y manipulación de secuencias de vídeo.

La API, diseñada con Django y utilizando MediaPipe y Keras, tiene la intención de reconocer gestos manuales en imágenes enviadas por los usuarios con el fin de realizar una predicción de un gesto. Esta recorta y procesa la imagen enviada, para asemejarse a las imágenes de entrenamiento y la procesarán los modelos, devolviendo en formato JSON la respuesta. El flujo general es el siguiente:

1. El usuario envía una imagen a través de una solicitud POST.

2. La imagen es procesada para detectar y recortar la mano visible.

3. El recorte es preprocesado (escala de grises, redimensionado y normalizado).

4. Cada modelo cargado evalúa el gesto. Si uno de ellos detecta su clase con confianza (> 0.5), se considera reconocido.

5. Se devuelve un resultado en formato JSON con el gesto detectado o una etiqueta de error.

Endpoints disponibles:

- / (GET): Verifica que la API está activa. La respuesta es un h1 con el texto “Hola mundo”.
- /procesar/imagen/ (POST): Permite enviar una imagen para que la API detecte un gesto de la mano. Es necesario un parámetro (img) que contiene la imagen. Este endpoint puede devolver lo siguiente:
    - El gesto detectado.
    - nh, cuando no se ha detectado ninguno.
    - Imagen no proporcionada, cuando no se envía una imagen.
    - Solo se permite el método POST, cuando se usa otro método HTTP.

3. Problemas y Soluciones

Durante el desarrollo de **SiLant**, surgieron diversos desafíos técnicos y prácticos que fue necesario abordar para garantizar el correcto funcionamiento del sistema. A continuación, se detallan los principales problemas encontrados y las soluciones implementadas:

A. Incompatibilidades de versiones (Python, TensorFlow, OpenCV, MediaPipe)

- **Problema:**
  Al integrar distintas librerías como TensorFlow, OpenCV, y MediaPipe en el proyecto, surgieron conflictos de compatibilidad entre versiones. Por ejemplo, TensorFlow 2.x no siempre es compatible con versiones antiguas de Python, mientras que algunas versiones de OpenCV generan errores con MediaPipe al manejar capturas de cámara en tiempo real.

- **Soluciones:**

  - Se definió un entorno virtual (usando venv) específico para el proyecto, lo que permitió aislar las dependencias.

  - Se creó un archivo requirements.txt con las versiones exactas de las librerías probadas para evitar inconsistencias entre entornos.

  - Se realizó una investigación de compatibilidad y se fijaron versiones estables.

  - Se documentó el proceso de instalación paso a paso para que nuevos colaboradores puedan replicar el entorno sin problemas (README).

B. Bajo rendimiento del modelo (poco acierto en la detección de gestos)

- **Problema:**
  El modelo presentaba un bajo nivel de acierto en la clasificación de gestos, especialmente en condiciones de iluminación variable, fondos complejos o con usuarios diferentes.

- **Soluciones:**

  - Se amplió el dataset incluyendo imágenes capturadas en diversas condiciones de iluminación y con participantes variados para mejorar la generalización.

- Se implementó un preprocesamiento más robusto (normalización, redimensionado consistente, reducción de ruido).

- Se ajustaron hiperparámetros como la tasa de aprendizaje y el número de épocas, y se aplicaron técnicas como *early stopping* para evitar sobreajuste.

C. Latencia en tiempo real

- **Problema:**
  El procesamiento de vídeo en tiempo real generaba retardos perceptibles y caídas de frames, especialmente en equipos con poca capacidad de procesamiento.

- **Soluciones:**

  - Se optó por separar el procesamiento del vídeo en tiempo real, con la predicción de gestos en los frames, Dividir en interfaz y API, para que el vídeo no colapsase al intentar predecir en tiempo real. De esta manera hay cierto retardo entre el gesto y la respuesta, pero no hay esas caídas de frames o vídeo congelado, que empeora la interacción con el usuario.

4. Decisiones Tomadas

Modelos

- Se decidió usar las redes neuronales convolucionales debido a que eran las más eficaces en el reconocimiento de imágenes. También se optó por TensorFlow/Keras debido a su facilidad de implementación, documentación extensa y compatibilidad con múltiples plataformas.

Procesamiento de video

- Se empleó **MediaPipe** por su precisión y eficiencia en la detección de manos y rostros. Se usó tanto para detectar si el usuario está mirando a la cámara, para evitar órdenes involuntarios, como en el recorte de imágenes, seleccionando solo el espacio ocupado por la mano, para mejorar la precisión del modelo, esto se usa tanto para preparar las imágenes del dataset, como para preprocesar los frames que le llegan a la API, para que ambas sean lo más semejante posible.

- **OpenCV** se utilizó para operaciones básicas de procesamiento de imagen y captura de video. Tanto para indicar al usuario cuando se detecta que mira a la cámara, con un círculo verde, como para mostrar el menú de opciones. Además se usó para depurar y hacer pruebas antes de elaborar la interfaz de usuario final.

Lenguajes y Frameworks

- **Python** ha sido utilizado para gran parte del proyecto, debido a su facilidad de uso, flexibilidad y compatibilidad con las tecnologías usadas.
- **Django** fue elegido para el desarrollo de la API REST, debido a su robustez, escalabilidad y facilidad para manejar endpoints y modelos de datos complejos.

5. Conclusiones

El sistema SiLant cumple con el objetivo de proporcionar una herramienta funcional para la detección y reconocimiento de gestos y detección de la mirada en tiempo real. Ofrece una alternativa tecnológica de bajo costo para mejorar la comunicación asistida en personas con discapacidad en el habla.

Ventajas:

- **Modularidad:** La arquitectura permite operar la API y la aplicación web de manera independiente.

- **Escalabilidad:** Es posible agregar nuevos gestos al sistema mediante la recolección de datos y el reentrenamiento del modelo.

Limitaciones:

- La precisión del sistema depende directamente de la calidad y diversidad del conjunto de datos de entrenamiento.

- El rendimiento en tiempo real puede disminuir significativamente en dispositivos con baja capacidad de procesamiento gráfico o memoria limitada.

Futuras Mejoras:

- Expansión del dataset para incluir más gestos, mayor variedad de condiciones de iluminación, diferentes tonos de piel y variaciones de fondo.

- Incorporación de mecanismos de calibración automática de cámara y seguimiento más preciso de la mirada.

- Optimización de los modelos existentes para la mejora del rendimiento en distintas plataformas.

- Reentrenar un modelo pre entrenado como ResNet.