Escuela
Politécnica
Superior

Intérprete "artificial" de lengua de signos

Grado en Ingeniería en Sonido e Imagen en Telecomunicación

Trabajo Fin de Grado

Autor:
Andrés Torres López
Tutor/es:
María Alfaro Contreras
Encarnación Gimeno Nieves

Julio 2023

Universitat d'Alacant
Universidad de Alicante

Intérprete “artificial” de lengua de signos

Autor
Andrés Torres López

Tutor/es
María Alfaro Contreras
Departamento de Lenguajes y Sistemas Informáticos
Encarnación Gimeno Nieves
Departamento de Física, Ingeniería de Sistemas y Teoría

Grado en Ingeniería en Sonido e Imagen en Telecomunicación

Escuela
Politécnica
Superior

Universitat d’Alacant
Universidad de Alicante

ALICANTE, Julio 2023

Resumen

Este Trabajo de Fin de Grado (TFG) busca crear un modelo de Inteligencia Artificial (IA) basado en el uso de Redes Neuronales Convolucionales (CNN) que sea capaz de detectar las diferentes letras del alfabeto de la Lengua de Signos Americana (LSA) que sean estáticas, es decir, que no requieran movimiento para ser reconocidas.

Para lograr este objetivo, lo primero que se hizo fue realizar una serie de tutoriales para obtener un mayor conocimiento sobre el funcionamiento de las CNN. Con este conocimiento adquirido, se comenzó a preparar el modelo añadiendo distintas capas convolucionales, capas densas y capas de aumento de datos, las cuales resultaron fundamentales para obtener buenos resultados, con una tasa de acierto de hasta el 94%. Una vez que el modelo estuvo preparado, se creó una interfaz de reconocimiento en tiempo real mediante el uso de la cámara.

Tras la implementación conjunta del modelo con el funcionamiento de la cámara, se obtuvieron resultados muy satisfactorios en el reconocimiento de los signos del alfabeto de la LSA, logrando así el objetivo inicial de crear un modelo capaz de reconocer la LSA.

Abstract

This Final Degree Project (FDP) aims to create an Artificial Intelligence (AI) model based on the use of Convolutional Neural Networks (CNN) that is capable of detecting the different static letters of the American Sign Language (ASL) alphabet, meaning those that do not require movement to be recognized.

To achieve this objective, the first step was to go through a series of tutorials to gain a better understanding of how CNN work. With this acquired knowledge, we created a model by adding various convolutional layers, dense layers, and data augmentation layers, which proved to be essential in obtaining good results, with an accuracy rate of up to 94%. Once the model was trained, a real-time recognition interface was created using a camera.

The joint implementation of the model and camera functionality led to highly satisfactory results in recognizing the signs of the ASL alphabet, thus achieving the initial goal of creating a model capable of recognizing ASL.

Agradecimientos

Antes de comenzar, me gustaría agradecer todo el apoyo que he recibido, no solo durante la realización de este trabajo, sino también a lo largo de toda mi etapa universitaria.

Dar las gracias a todas esas personas que han llegado a mi vida gracias a la Delegación de Estudiantes, por enseñarme a trabajar en grupo y a mirar por el bien de los estudiantes.

Gracias al CEET, por todo lo que me ha aportado tanto como representante como al ser presidente, pero sobretodo por permitirme conocer a mi otra familia.

Como no dar las gracias a las nuevas amistades que me ha dado la carrera, gracias a «Grupo 9 SAA» y a «Putos Lokos» por formar parte de mi día a día, pero sobretodo dar las gracias a «Urgensia» por todo el apoyo que me han dado.

Gracias a todos mis compañeros de la carrera, por todas esas fiestas de más de 50 personas en una discoteca enana, pero sobretodo por permitirme representarlos a todos hasta el final.

Gracias a mis tutoras, María y Encarna, por responder a todas mis dudas (que no han sido precisamente pocas) y ayudarme en todo lo que he necesitado.

Pero por encima de todo, dar las gracias a mi madre, quien me ha impulsado desde pequeño a ser la mejor versión de mí mismo, a no rendirme y a seguir hasta el final.

A mis compañeros, amigos y familia Burgos, MG y Poyato,
sin los cuales no habría podido acabar la carrera.

Bueno pues en Julio la sacas bro

Francisco Morant Poyato

Índice general

Índice de figuras

Índice de tablas

1. Introducción

Hoy en día, la Inteligencia Artificial (IA) está muy presente en el mundo. Aplicaciones como YouTube, Instagram y TikTok utilizan IA para recomendar publicaciones. Del mismo modo, programas como ChatGPT o Bing están siendo ampliamente utilizados a pesar de encontrarse en fases prematuras de desarrollo. No es sorprendente, por tanto, que en los últimos años el uso de la IA haya aumentado vertiginosamente en diversos campos de la sociedad. Se utiliza en música, arte, cine, análisis de datos, procesamiento de imágenes, educación y, en este trabajo, para potenciar la accesibilidad.

Actualmente, una parte de la sociedad se enfrenta a problemas de accesibilidad debido a diversos factores, como la ceguera, la baja movilidad o la sordera. Este trabajo se centra principalmente en las personas sordomudas o con problemas de audición, quienes a menudo dependen de la lengua de signos como su principal medio de comunicación. Esta comunidad emplea este lenguaje gestual y visual, que es tan rico y complejo como cualquier lenguaje hablado, para expresarse y entender a los demás.

Este Trabajo Final de Grado (TFG) buscar desarrollar un algoritmo, basado en Redes Neuronales Convolucionales (CNN), capaz de clasificar diferentes signos de la Lengua de Signos Americana (LSA). La LSA es la lengua de signos dominante en Estados Unidos, en la parte anglófona de Canadá, y es utilizada, incluso, en algunas partes de México.

La decisión de trabajar con la LSA se debe principalmente a que existen conjuntos de datos públicos de gran tamaño (alrededor de 60.000 muestras). Esto nos permitirá verificar que el enfoque basado en IA es correcto y evitar los problemas asociados con la falta de datos.

1.1. Objetivos

Tal y como se ha mencionado anteriormente, el objetivo principal de este proyecto es crear un algoritmo capaz de reconocer los signos de la LSA. Para llevar a cabo el proyecto, se han definido los siguientes objetivos específicos:

- Aprender sobre tecnología de vanguardia en IA.
- Ser capaz de definir y entrenar una red neuronal, en concreto, una CNN.
- Construir un modelo que reconozca la LSA en imágenes, ya sean estáticas o capturadas por una *webcam*.

1.2. Estructura del trabajo

Para facilitar su lectura, el contenido de este documento se encuentra organizado en capítulos y secciones. A continuación, se detalla de forma ordenada la estructura junto con los correspondientes contenidos del resto del documento.

En el **Capítulo 2**, se presenta de forma teórica los conceptos de LSA, CNN y cómo estos pueden ser tratados de forma conjunta.

En el **Capítulo 3**, se explica el problema que se tratará de resolver en este trabajo, así como la solución que se le dará y las herramientas empleadas.

En el **Capítulo 4**, se presenta el conjunto de datos o *dataset* utilizado durante la realización de este trabajo, así como las métricas y los distintos casos de estudio.

En el **Capítulo 5**, se muestran los diferentes resultados obtenidos en los escenarios de evaluación definidos en el Capítulo 4.

Finalmente, en el **Capítulo 6**, se realiza una conclusión sobre el proyecto, recapitulando el problema abordado, los objetivos alcanzados y las posibles nuevas vías de mejora que se abren con este proyecto.

2. Marco teórico

En esta sección, se busca comentar todo lo relacionado con los conceptos necesarios para comprender el contexto de este TFG. A continuación, presentaremos un breve resumen de la historia de la LSA, así como una serie de conceptos teóricos sobre las CNN. Por último, analizaremos cómo se pueden relacionar ambos campos.

2.1. Lengua de signos americana

La LSA es un lenguaje visual-gestual utilizado por la comunidad sorda en Estados Unidos y en la parte anglófona de Canadá. Se utiliza también en algunas partes de México. Aunque se cree que la lengua ha existido durante cientos de años, su reconocimiento oficial es relativamente reciente.

En Estados Unidos, la educación de los sordos ha sido un tema controvertido mucho tiempo. Durante gran parte del siglo XIX, la educación de los sordos se basó en el método oralista, que se centraba en enseñar a los estudiantes a hablar y leer los labios en lugar de usar la lengua de señas. Sin embargo, a finales del siglo XIX, la lengua de señas comenzó a ser aceptada como un medio de comunicación legítimo y valioso para la comunidad sorda (Newport y Meier, 1985).

La aceptación de la lengua de señas se debió principalmente a la creación de escuelas específicas para la educación de los sordos. La primera de estas escuelas fue la *American School for the Deaf*, fundada en 1817 en Hartford, Connecticut. La escuela fue fundada por un clérigo sordo llamado Thomas Hopkins Gallaudet, quien viajó a Europa en busca de métodos de educación para sordos. En Francia, Gallaudet conoció a Laurent Clerc, un sordo francés que fue educado en la lengua de señas francesa. Gallaudet invitó a Clerc a venir a los Estados Unidos para ayudar a establecer la *American School for the Deaf*. Juntos, Gallaudet y Clerc desarrollaron una lengua de señas que combinaba elementos de la lengua de señas francesa con la lengua de señas utilizada por los sordos estadounidenses.

A medida que se establecieron más escuelas para la educación de los sordos en todo el país, la lengua de señas se hizo más común y se desarrolló un lenguaje más estandarizado. En 1864, la *Gallaudet University* fue fundada en Washington D.C. como la primera universidad para la educación de los sordos. La universidad es considerada un centro de investigación y educación líder en la LSA (Newport y Meier, 1985).

Hoy en día, la LSA es reconocida como un lenguaje independiente y completo con su propia gramática y vocabulario. Se utiliza en una variedad de situaciones, desde la educación hasta el entretenimiento y la comunicación diaria. La LSA también es utilizada por muchas personas que no son sordas, incluyendo intérpretes y demás personas que trabajan con la comunidad sorda.[1]

2.2. Redes neuronales convolucionales

La Inteligencia Artificial es una rama de la ingeniería que se centra en el desarrollo de sistemas capaces de realizar tareas que normalmente requerirían habilidades humanas (Rouhiainen, 2018). Dentro de esta disciplina, existen dos conceptos muy importantes: el aprendizaje automático o *machine learning* y el aprendizaje profundo o *deep learning*.

El *machine learning* es una técnica que permite a los sistemas informáticos aprender de forma autónoma a partir de datos, sin necesidad de programar explícitamente todas las reglas que deben seguir (Géron, 2022). En lugar de eso, se les proporciona un conjunto de datos de entrenamiento y un algoritmo que les permite aprender a partir de ellos. De esta forma, el sistema puede identificar patrones y hacer predicciones con una precisión cada vez mayor.

Por otro lado, el *deep learning* es un tipo de *machine learning* que se basa en el uso de redes neuronales artificiales, un conjunto de algoritmos inspirados en el funcionamiento del cerebro humano (LeCun y cols., 2015). Las redes neuronales profundas son capaces de aprender a partir de datos estructurados o no estructurados, y se utilizan en una amplia variedad de aplicaciones, desde la visión por computador hasta el procesamiento del lenguaje natural. Este TFG se encuadra dentro del campo de la visión por computador, ya que se emplearán Redes Neuronales Convolucionales, un algoritmo de aprendizaje profundo, para clasificar los signos de la LSA a partir de imágenes.

Las Redes Neuronales Convolucionales son un tipo de arquitectura de redes neuronales profundas que se utilizan comúnmente en tareas de reconocimiento de imágenes, vídeo, voz y texto. Su evolución se remonta a finales de la década de 1990, cuando fueron desarrolladas originalmente por LeCun y cols. como una solución para resolver problemas de reconocimiento de caracteres manuscritos. Sin embargo, no fue hasta la década de 2010 que las CNN comenzaron a popularizarse, gracias a su uso en concursos de reconocimiento de imágenes (Krizhevsky y cols., 2012).

La idea de las CNN se basa en la organización del sistema visual de los seres vivos, en el que las neuronas en el cerebro se activan para detectar características específicas de una imagen. Por ejemplo, las neuronas en la corteza visual primaria se activan para detectar bordes, formas y otros patrones visuales.

[1] Se recomienda, como lectura complementaria, la entrada del *blog* aportada por un colegio oficial de LSA: `https://www.startasl.com/history-of-american-sign-language/`

Las CNN están formadas por dos bloques: el bloque extractor de características, compuesto por una serie de capas de convolución y submuestreo o *pooling* utilizadas para extraer características de las imágenes; y el bloque clasificador de características, compuesto por capas completamente conectadas (también conocidas como densas) utilizadas para realizar la clasificación final.

La convolución es una operación matemática que se utiliza para extraer características de una imagen. En las CNN, una capa convolucional utiliza un conjunto de filtros o *kernels* que se aplican a la imagen de entrada. Cada filtro se desplaza por toda la imagen, calculando un producto punto a punto entre el *kernel* y la región de la imagen que se encuentra debajo del *kernel*. El resultado es una imagen filtrada que resalta las características relevantes de la imagen original (Albawi y cols., 2017).

Además de las capas convolucionales, las CNN cuentan con capas de *pooling*, que se utilizan para reducir la dimensionalidad de la imagen y hacer que la red sea más eficiente en términos computacionales. En una capa de *pooling*, se divide la imagen en regiones no superpuestas y se toma el valor máximo o promedio de cada región. El resultado es una imagen reducida que mantiene las características más importantes de la imagen original.

Tras las capas de convolución y *pooling*, las capas completamente conectadas se utilizan para realizar la clasificación final. Se puede observar un ejemplo de la estructura de una red neuronal convolucional en la Figura 2.1.

Las redes neuronales convolucionales han demostrado ser muy efectivas en tareas de reconocimiento de imágenes gracias a su capacidad para extraer características relevantes de las mismas. Actualmente, las CNN se utilizan en una amplia gama de aplicaciones, desde reconocimiento facial hasta la detección de enfermedades en imágenes médicas, e incluso en la identificación de objetos en fotografías (Gu y cols., 2018).

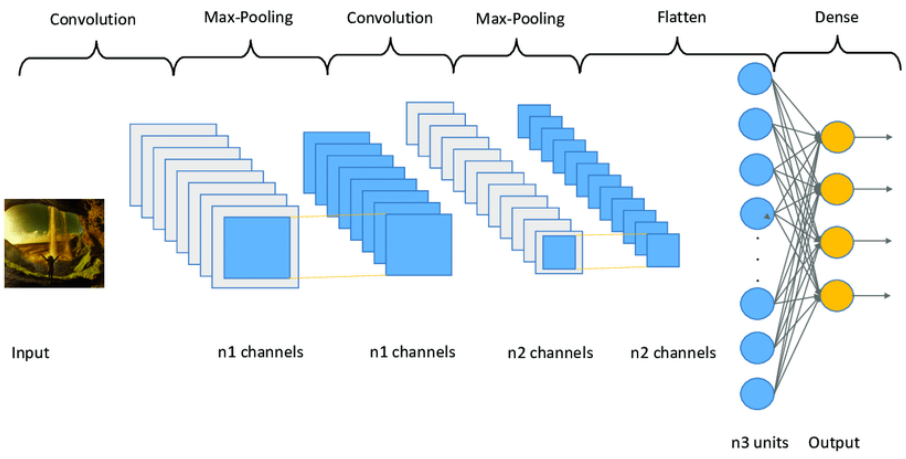


Figura 2.1: Redes Neuronales Convolucionales. Tomada de Benítez-Andrades (s.f.).

2.3. Las redes neuronales convolucionales y la lengua de signos

El reconocimiento automático de la lengua de signos ha sido un desafío significativo para la investigación en aprendizaje automático, debido a la complejidad y variabilidad de los gestos. Sin embargo, con el avance de las técnicas de aprendizaje profundo y el uso de las CNN, se ha logrado un progreso significativo en esta área.

Las CNN han demostrado ser una herramienta eficaz para la detección y reconocimiento de gestos de manos en el contexto de la lengua de signos (Cheok y cols., 2019; Pigou y cols., 2015). Estas redes pueden detectar y extraer características de los gestos de manos, como la forma y posición de las manos, la orientación de los dedos y el movimiento. Estas características resultan esenciales para la correcta identificación de los diferentes gestos de cualquier lengua de signos.

Más allá del reconocimiento de gestos, las CNN también se han aplicado en la generación de gestos de manos (Ferstl y cols., 2020). En este caso, se emplea un modelo generativo para producir gestos de manos realistas que puedan ser utilizados en aplicaciones de comunicación.

El uso de las CNN en la lengua de signos ha permitido un avance significativo en la comunicación entre personas sordas o con problemas de audición y el resto de la sociedad. La capacidad de las CNN para detectar y reconocer los gestos de manos de la lengua de signos ha abierto la puerta a nuevas aplicaciones y tecnologías que permiten a las personas con discapacidad auditiva comunicarse de manera más efectiva y participar plenamente en la sociedad.

Sin embargo, aún quedan desafíos por superar en esta área, como la variabilidad en los gestos de la lengua de signos entre diferentes personas y culturas, así como la necesidad de desarrollar modelos más precisos y eficientes para la detección y reconocimiento de gestos. A medida que la investigación en aprendizaje automático continúa avanzando, es probable que se siga logrando un progreso significativo en la aplicación de las CNN en la lengua de signos, mejorando aún más la comunicación para las personas con discapacidad auditiva.

3. Metodología

En el presente capítulo se explicará el problema abordado por el TFG, así como la solución propuesta, junto con las diferentes herramientas y librerías utilizadas durante el desarrollo del proyecto. Asimismo, se describirá la planificación llevada a cabo a lo largo de este trabajo.

3.1. Descripción del problema

Este trabajo busca crear un programa capaz de reconocer la LSA. Su funcionamiento se asemejará a lo que se muestra en la Figura 3.1. Por lo tanto, se pretende desarrollar una interfaz que tenga acceso a la cámara del dispositivo del usuario. Se asume que el dispositivo de uso es un ordenador que cuenta con una *webcam*. La interfaz mostrará un recuadro donde el usuario deberá posicionar su mano para que el algoritmo basado en CNN pueda identificar el signo correspondiente. Es decir, cada uno de los fotogramas capturados por la *webcam* será introducido en la CNN, devolviendo el valor del signo que se está intentando reproducir en cada instante. Es importante tener en cuenta que la mano no será reconocida si no se coloca dentro del recuadro.

Figura 3.1: Esquema de funcionamiento.

3.2. Propuesta de solución

Para obtener el algoritmo, se plantea el uso de una CNN capaz de reconocer algunos de los signos de la LSA, en concreto aquellos que no requieren movimiento para su interpretación. Las imágenes de las manos serán capturadas mediante el uso de una *webcam*.

Algunos de los principales problemas que pueden aparecer durante el desarrollo del proyecto son, por ejemplo, que el número de capas a utilizar quede corto o sea excesivo, así como no utilizar las suficientes épocas durante el entrenamiento de la red.

Para conseguir una alta precisión, se ha implementado el uso de aumentado de datos o *data augmentation*, una técnica empleada en el procesamiento de datos en el aprendizaje automático. Consiste en aplicar transformaciones y variaciones aleatorias a los datos existentes para generar nuevos ejemplos. Con esto se amplía el tamaño y la diversidad del conjunto de datos, mejorando el rendimiento y la generalización de los modelos de *machine learning*.

Para poder medir las pérdidas y permitir que la red pueda aprender correctamente, se ha utilizado la entropía cruzada, que se emplea para cuantificar la diferencia entre dos distribuciones de probabilidad. Se utiliza comúnmente como función de pérdida en problemas de clasificación, donde mide la discrepancia entre las predicciones del modelo y los valores reales.

En cuanto al empleo de la cámara y el desarrollo de la interfaz, se utilizará Open Source Computer Vision Libreary (OpenCV) (Bradski, 2000). Se trata de una librería que tiene implementada la función `cv2.VideoCapture()`[1], la cual permite grabar *frame* a *frame* utilizando la cámara del dispositivo donde se ejecuta. Este algoritmo se conecta al creado para el proyecto de forma que detecte, sin problemas, los distintos signos de la LSA. El resultado se final se puede observar en la Figura 3.2.

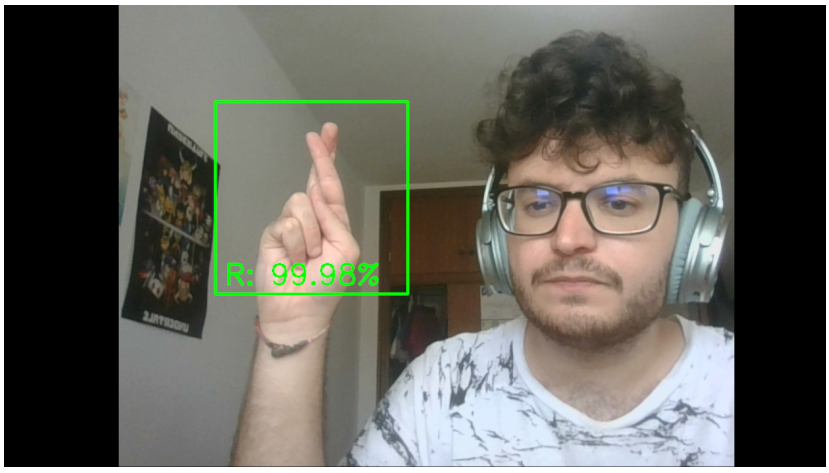


Figura 3.2: Resultado real.

[1] *Capture Video from Camera*. `https://docs.opencv.org/3.4/dd/d43/tutorial_py_video_display.html`

A la hora de ejecutar este código, lo primero que aparecerá será un mensaje en la terminal que pedirá introducir las letras «D» o «I» para elegir qué mano utilizar, si la derecha o la izquierda, respectivamente. Después de esto, se activará la cámara, donde se podrán realizar los diferentes signos estáticos de la LSA. En caso de que el usuario desconozca los posibles signos a realizar, podrá pulsar la tecla «Espacio». Al hacerlo, aparecerá en pantalla la Figura 4.1, lo cual servirá como referencia para los signos que se pueden realizar. Para cerrar el programa, será necesario pulsar la tecla «Enter».

3.3. Herramientas

3.3.1. Python

Una de las bases para la realización del algoritmo ha sido el uso del lenguaje de alto nivel Python. Python es conocido por su sintaxis clara y concisa, así como por su capacidad de ser utilizado en una amplia variedad de aplicaciones, desde ciencia de datos hasta desarrollo web y automatización de tareas (Challenger-Pérez y cols., 2014). Es uno de los lenguajes de programación más populares en la actualidad y su comunidad de desarrolladores activos sigue creciendo.

3.3.2. Google Colaboratory

Para la ejecución del código, se ha aprovechado el uso de Google Colaboratory, conocido también como Google Colab[2]. Se trata de una plataforma de cuadernos interactivos basada en la nube que permite escribir, ejecutar y compartir código en Python. Se utiliza ampliamente para proyectos de aprendizaje automático y ciencia de datos, ya que ofrece acceso gratuito a unidades de procesamiento gráfico (GPU). Google Colab permite a los usuarios colaborar en tiempo real y compartir sus cuadernos a través de Google Drive.

3.4. Librerías

3.4.1. TensorFlow

Como principal librería se utilizará TensorFlow (TF).[3] El uso de TF se debe a que es una de las librerías de aprendizaje automático de código abierto más populares en Python. Esta librería se utiliza para construir y entrenar modelos de aprendizaje automático, especialmente redes neuronales profundas.

Dentro de TF también se encuentra la librería Keras (Chollet, 2021), que es una librería de redes neuronales de alto nivel para Python. Keras se enfoca principalmente en la experimentación rápida y sencilla de las redes neuronales, lo que la ha convertido en una herramienta muy popular en la comunidad de aprendizaje automático. Al estar integrada con TF, Keras ofrece una amplia gama de funciones y herramientas con las que se pueden desarrollar y entrenar modelos de aprendizaje profundo.

[2] `https://colab.research.google.com`
[3] `https://www.tensorflow.org/`

3.4.2. Matplotlib

Matplotlib[4] es una biblioteca de visualización de datos en Python creada por John D. Hunter. Es una herramienta poderosa y ampliamente utilizada que permite crear gráficos de alta calidad, diagramas y visualizaciones interactivas. Matplotlib ofrece una amplia gama de funciones para representar datos en diferentes formatos, como gráficos de líneas, barras, dispersión, histogramas y más. Es una herramienta esencial en el análisis y exploración de datos, ya que proporciona una forma efectiva de comunicar información de manera visual y comprensible.

3.4.3. OpenCV

Tal y como se comentó en la sección anterior, se ha utilizado la librería OpenCV[5] para el uso de la cámara en tiempo real. OpenCV es una biblioteca de código abierto ampliamente utilizada en aplicaciones de visión por computador y procesamiento de imágenes (Bradski, 2000). Fue desarrollada por Intel en 1999 y actualmente es considerada un estándar en la comunidad de la visión por computador.

OpenCV proporciona una amplia gama de funciones y algoritmos para manipular imágenes y vídeos, incluyendo operaciones de procesamiento de imágenes y reconocimiento de objetos. Se utiliza en diversos campos como la robótica, la realidad aumentada, la automatización industrial, la vigilancia y el análisis médico, lo que la convierte en una biblioteca muy completa y potente para el procesamiento de imágenes.

3.5. Desarrollo

El proceso de desarrollo de este TFG ha sido un proceso con etapas de formación, pruebas, errores y soluciones.

La primera parte de este proyecto consistió en una formación sobre las CNN, TF y Keras. Por un lado, se realizaron distintos tutoriales encontrados en línea para aprender cómo funcionan las CNN en distintos ámbitos. Por otro lado, se llevó a cabo un estudio y recopilación de información sobre las utilidades y el funcionamiento de las librerías de TF y Keras.

Tras esto se comenzó con las pruebas, preparando unas primeras versiones donde se trataba de conseguir que el algoritmo reconociera la LSA pese a no tener un gran porcentaje de aciertos. Una vez se comprobó que el algoritmo trabajaba bien con el *dataset* de LSA, se comenzó a trabajar en aumentar su precisión, llegando a niveles de un 90% de precisión.

[4] `https://matplotlib.org/`
[5] `https://opencv.org/`

Dentro del contexto del aprendizaje automático, el entrenamiento es el proceso de ajustar los parámetros de un modelo utilizando un conjunto de datos de entrenamiento. Consiste en presentar iterativamente ejemplos de entrada al modelo, calcular sus predicciones, compararlas con las etiquetas verdaderas y ajustar los parámetros para minimizar la diferencia entre las predicciones y las etiquetas de referencia. Este proceso de ajuste se repite varias veces, actualizando gradualmente los parámetros del modelo para mejorar su rendimiento.

El objetivo del entrenamiento es lograr que el modelo aprenda a reconocer patrones y regularidades en los datos, con el fin de realizar inferencias precisas y tomar decisiones basadas en estos. Gracias a ello, se espera que una vez entrenado, el modelo sea capaz de generalizar los conocimientos adquiridos y aplicarlos sobre datos nuevos no vistos previamente.

Para lograr el objetivo anterior, es necesario encontrar la configuración óptima de los hiperparámetros del modelo. Los hiperparámetros son parámetros externos al modelo que afectan su rendimiento y comportamiento, pero no son aprendidos durante el entrenamiento. Ejemplos comunes de hiperparámetros son el número de épocas de entrenamiento, el número de capas, etc. La búsqueda de esta configuración óptima implica realizar una serie de pruebas y ajustes. En el siguiente capítulo, se describirá con detalle el proceso de ajuste que se ha realizado para lograr que la CNN implementada reconozca de manera efectiva los signos de la LSA. Se explicará cómo se han seleccionado y ajustado los hiperparámetros, así como los resultados obtenidos en cada etapa del proceso.

3.6. Acceso al código

Para poder acceder a la totalidad del código utilizado para este TFG, se ha preparado el siguiente enlace a *GitHub*: `https://github.com/Andres213/TFG`. Dentro de este enlace se encontrarán dos archivos: uno referente al modelo llamado `ModelLSA.ipynb` y otro llamado `CamaraTFG.py`. El primer archivo contiene todo lo relacionado con cómo se ha entrenado el modelo. Se recomienda ejecutarlo desde Google Colaboratory. El segundo archivo contiene todo lo relacionado con la función de la cámara. Para ejecutar este archivo, se recomienda utilizar un compilador de escritorio, como por ejemplo *VSCode*.[6]

[6] *Visual Studio Code* o *VSCode* es un editor de código fuente desarrollado por Microsoft. Es ampliamente utilizado por programadores y desarrolladores debido a su interfaz intuitiva, su amplia gama de extensiones y su capacidad de personalización.

4. Experimentación

A lo largo de este capítulo se presentará la sección más técnica del proyecto, abarcando el conjunto de datos utilizado, las métricas consideradas, la configuración y el entrenamiento del modelo, así como los diferentes casos de estudio que se tendrán en cuenta en el siguiente capítulo.

4.1. Dataset

Un conjunto de datos o *dataset* es una colección estructurada de datos que representa una fuente de información específica. Un *dataset* se compone de múltiples instancias o muestras, cada una de las cuales contiene atributos o características que describen un elemento. Los *datasets* se utilizan en el aprendizaje automático y en la ciencia de datos para entrenar y evaluar modelos y así extraer conocimientos a partir de los datos. Para la realización de este TFG, se ha utilizado un único *dataset* basado en la LSA.

4.1.1. Lengua de signos americana

El *dataset* de LSA utilizado se llama *Sign Language MNIST*.[1] Consiste en un conjunto de 27.455 imágenes de entrenamiento y 7.172 imágenes de test, que es aproximadamente la mitad de un *dataset* MNIST[2] estándar. Es importante destacar que durante las pruebas se utiliza un 20% aleatorio de los datos de entrenamiento como conjunto de validación para evaluar el rendimiento del modelo durante el entrenamiento y ajustar los hiperparámetros. Esto ayuda a asegurarse de que el modelo no se esté sobreajustando (memorizando) a los datos de entrenamiento y pueda generalizar bien a nuevos datos.

El *dataset* de la LSA contiene 23 de las 25 letras utilizadas en el alfabeto inglés. Esto se debe a que a la hora de signar las letras J y Z se ha de realizar un movimiento con las manos, el cual no puede capturarse en una única imagen.

Es importante tener en cuenta que este *dataset* está formado por imágenes con buena iluminación, piel muy clara y fondo blanco. Además, todas las manos representadas son zurdas. La Figura 4.1 ilustra una muestra de las diferentes imágenes que se encuentran en el *dataset*.

[1] Disponible en: `https://www.kaggle.com/datasets/datamunge/sign-language-mnist`

[2] La base de datos MNIST es un conjunto de datos ampliamente utilizado en el campo de la visión por computador y el aprendizaje automático. Consiste en un conjunto de 60.000 imágenes de dígitos escritos a mano para entrenamiento y 10.000 imágenes adicionales para pruebas. La base de datos MNIST ha sido un punto de referencia popular para evaluar algoritmos y modelos de reconocimiento de dígitos durante muchos años.

Figura 4.1: *Dataset* de LSA.

En la Tabla 4.1 se aprecia de forma sintetizada las características más importantes del *dataset* utilizado.

	Número de muestras	**Número de clases**	**Tamaño de las imágenes**
LSA	34.627	24	28x28

Tabla 4.1: Datos del *dataset* de LSA.

4.2. Métricas

Para poder medir correctamente el rendimiento y la precisión del modelo, se utilizan diferentes métricas. Estas métricas proporcionan información cuantitativa sobre cómo se está desempeñando el modelo en términos de exactitud y valor de F_1, entre otras.

4.2.1. Tasa de acierto o exactitud

La tasa de acierto, también conocida como exactitud, es una métrica utilizada para evaluar el rendimiento de un modelo de clasificación. Se calcula dividiendo el número de predicciones correctas realizadas por el modelo entre el total de predicciones realizadas, y luego multiplicando el resultado por 100 para expresarlo como un porcentaje.

La fórmula para calcular la tasa de acierto es:

$$\text{Tasa de acierto} = \frac{\text{Predicciones correctas}}{\text{Total de predicciones}} * 100 \tag{4.1}$$

Esta métrica proporciona una medida de la proporción de predicciones correctas realizadas por el modelo en relación con el conjunto total de predicciones. Una tasa de acierto alta indica que el modelo está realizando predicciones precisas, mientras que una tasa de acierto baja indica que el modelo está cometiendo errores en sus predicciones.

4.2.2. Valor de F_1

En el contexto de la clasificación de imágenes, es habitual utilizar los conceptos de falsos positivos, falsos negativos, verdaderos positivos y verdaderos negativos.

- **Falsos positivos**: Se refiere a los casos en los que el modelo clasifica erróneamente una muestra como perteneciente a una clase determinada, cuando en realidad no lo es. Es decir, el modelo predice positivo cuando debería ser negativo.
- **Falsos negativos**: Se refiere a los casos en los que el modelo clasifica erróneamente una muestra como no perteneciente a una clase determinada, cuando en realidad sí lo es. Es decir, el modelo predice negativo cuando debería ser positivo.
- **Verdaderos positivos**: Son los casos en los que el modelo clasifica correctamente una muestra como perteneciente a una clase determinada. Es decir, el modelo predice positivo y realmente es positivo.
- **Verdaderos negativos**: Son los casos en los que el modelo clasifica correctamente una muestra como no perteneciente a una clase determinada. Es decir, el modelo predice negativo y realmente es negativo.

El valor de F_1 es una métrica utilizada en la evaluación de modelos de clasificación que combina la precisión y el *recall* (también conocido como sensibilidad) para proporcionar una medida equilibrada del desempeño del modelo. Se calcula utilizando la siguiente fórmula:

$$F_1 = 2 * \frac{\text{precisión} + \textit{recall}}{\text{precisión} + \textit{recall}} \tag{4.2}$$

Donde la precisión se define como el cociente entre los verdaderos positivos y la suma de verdaderos positivos y falsos positivos, y el *recall* se define como el cociente entre los verdaderos positivos y la suma de verdaderos positivos y falsos negativos.
En escenarios de clasificación multiclase, como el de este TFG, calculamos el valor de F_1 para cada clase individualmente. Luego, para obtener una medida general del desempeño del modelo en todas las clases, se realiza un promedio de los valores de F_1 obtenidos para cada clase.

4.2.3. Matriz de confusión

La matriz de confusión es una herramienta utilizada en el campo de la evaluación de modelos de clasificación para visualizar el rendimiento del modelo. Esta matriz organiza las predicciones del modelo en una tabla que muestra la cantidad de instancias clasificadas correctamente e incorrectamente para cada clase (véase Figura 4.2).

En una matriz de confusión, las filas representan las clases reales, mientras que las columnas representan las clases predichas por el modelo. Cada celda de la matriz muestra el recuento de ejemplos que caen en una determinada combinación de clase real y clase predicha. En un caso ideal, la diagonal principal de la matriz debería tener valores altos, lo cual indica una clasificación correcta, mientras que los valores fuera de la diagonal deberían ser lo más bajos posible, ya que representan clasificaciones fallidas.

La matriz de confusión proporciona información valiosa sobre los errores de clasificación cometidos por el modelo, como falsos positivos, falsos negativos y aciertos.

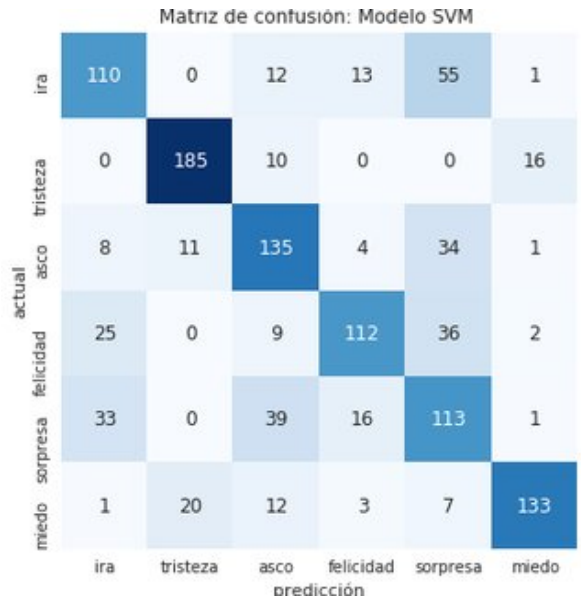


Figura 4.2: Ejemplo de matriz de confusión. Tomada de Pérez (s.f.).

4.3. Configuración y entrenamiento del modelo

En esta sección, se proporcionará una explicación más detallada sobre la configuración del modelo durante el proceso de entrenamiento y trabajo con el mismo. Se abordarán algunos conceptos clave, como el entrenamiento del modelo, la configuración de las capas, así como la aplicación de técnicas de regularización y aumentado de datos.

4.3.1. Técnicas de regularización

El sobreajuste es un fenómeno común en el aprendizaje automático, donde un modelo se ajusta demasiado a los datos de entrenamiento y pierde su capacidad para generalizar en nuevos datos. Es decir, memoriza los datos de entrenamiento. Para abordar este problema, se utilizan técnicas de regularización como el *batch normalization* y el *dropout*.

El *batch normalization* consiste en normalizar la salida de una capa anterior de la red neuronal, asegurando que los valores de los datos se mantengan en un rango adecuado. Esto acelera la convergencia del modelo durante el entrenamiento y mejora su capacidad para generalizar los patrones aprendidos. Normalmente, la capa de *batch normalization* se agrega después de las capas de convolución para normalizar los datos de un lote o *batch* de entrenamiento. Un *batch* es un grupo de datos de entrenamiento que se procesan simultáneamente en una iteración del entrenamiento. El tamaño del *batch* o *batch size* indica cuántos ejemplos se procesan en cada lote.

Otra técnica comúnmente utilizada es el *dropout*. Consiste en desactivar (poner a cero) aleatoriamente un porcentaje de los vectores de características extraídos por, en este caso, el bloque convolucional. Al hacerlo, se reducen las interdependencias, lo que ayuda a prevenir el sobreajuste.

Como se verá en las siguientes secciones, en la CNN creada para este TFG, se emplean ambas técnicas, *batch normalization* y *dropout*, para mitigar el sobreajuste y mejorar el rendimiento y generalización del modelo durante el entrenamiento y la prueba.

4.3.2. Data augmentation

Como se introdujo en el capítulo anterior, el aumento de datos, o *data augmentation*, es una técnica que se aplica en el procesamiento de datos para incrementar la cantidad y diversidad de ejemplos en un conjunto de entrenamiento. Esta técnica implica aplicar una serie de transformaciones a las muestras existentes, incluyendo rotaciones, desplazamientos, efectos de *zoom* y cambios de brillo, con el objetivo de generar nuevas instancias que, aunque presenten características ligeramente distintas, conserven la esencia del dato original.

El *data augmentation* se emplea para mejorar la generalización y el rendimiento de los modelos de aprendizaje automático. Al generar más ejemplos de entrenamiento a partir de las muestras existentes, se incrementa la diversidad y variabilidad del conjunto de datos. Esto, a su vez, ayuda a prevenir el sobreajuste, mejorando la capacidad del modelo para reconocer y generalizar patrones en nuevos ejemplos.

Incluiremos distintas técnicas de *data augmentation* como capas adicionales del modelo. Estas se colocarán al principio, de manera que cada vez que llegue una nueva imagen durante el entrenamiento, se le aplicarán una serie de distorsiones. En concreto, se rotarán las imágenes hasta 30^{o} tanto en sentido horario como antihorario, se aplicará un efecto espejo el 50% de la veces y un efecto de *zoom* aleatorio que puede ampliar o reducir la imagen hasta un 10%. Los espacios que queden vacíos en la imagen debido a estas transformaciones se llenarán con el valor del píxel más cercano.

Además, al modelo se le aplicará un *data augmentation* de brillo. Esto permitirá obtener resultados más robustos ante variaciones en la iluminación del entorno en el que se realicen las pruebas. Concretamente, se variará el factor de brillo de la imagen original en un 10%, lo que permitirá que, durante las pruebas con el modelo, no sea necesaria una iluminación tan detallada como la que se emplea en las imágenes del *dataset*.

4.3.3. Arquitectura del modelo

El modelo se crea a partir de la unión de distintas capas. Para este caso concreto, se han utilizado un total de dos capas de convolución, cada una con sus respectivas capas de *max-pooling* (que toman el valor máximo de cada región de 2x2 para reducir el tamaño de la imagen) y *batch normalization*. Además, se ha incluido una capa de *dropout* al final del bloque de extracción de características para evitar el sobreajuste.

El bloque clasificador de características, compuesto de dos capas completamente conectadas o densas, solo puede procesar datos unidimensionales. Por lo tanto, antes de llegar a este bloque, se ha empleado una capa *flatten* que convierte las imágenes (matrices tridimensionales) en vectores de una sola dimensión. Para lograr esto, la capa *flatten* reorganiza los datos en un vector lineal, preservando el orden.

Se puede consultar la Tabla 4.2 para comprobar las características de las distintas capas del modelo empleado para reconocer los signos de la LSA.

Capa	**Tamaño de salida**	**Parámetro entrenables**
Aumentado de datos	28x28x1	0
Convolucional	28x28x16	160
Batch normalization	28x28x16	64
Max-pooling	14x14x16	0
Convolucional	14x14x32	4.640
Batch normalization	14x14x32	128
Max-pooling	7x7x32	0
Dropout	7x7x32	0
Flatten	1.568	0
Densa	512	803.328
Densa	26	23.338

Tabla 4.2: Modelo de la LSA.

4.3.4. Hiperparámetros del modelo

Para el entrenamiento de este modelo se han considerado los siguientes hiperparámetros:

- **Función de perdida**. Se utiliza la entropía cruzada, ya que sirve para medir la discrepancia entre la distribución de probabilidades predicha por el modelo y la distribución real de los datos.

- **Optimizador**. Se utiliza el optimizador Adam (Kingma y Ba, 2015), el cual permite obtener resultados de entrenamiento más eficientes y una convergencia más rápida. Keras se encarga internamente de realizar el proceso de optimización o ajuste de los parámetros del modelo para minimizar, en este caso, el valor de la pérdida de entropía cruzada.

- **Número de épocas de entrenamiento**. Se emplean 5 épocas de entrenamiento. Debido a que, por un lado, los datos están «limpios» (las imágenes no están distorsionadas y los fondos son uniformes) y, por otro lado, el conjunto de datos es de gran tamaño, no se han requerido un número elevado de épocas de entrenamiento para ajustar los parámetros del modelo. Es importante destacar que durante cada época de entrenamiento se recorren todos los datos del conjunto de entrenamiento.

- ***Batch size***. Se emplean 128 muestras en cada paso de entrenamiento para ajustar los parámetros del modelo. En el siguiente capítulo se observará cómo este valor ha dado los mejores resultados.

Es importante destacar que, para acelerar el entrenamiento, se ha hecho uso de una Unidad de Procesamiento Gráfico (GPU) debido a su capacidad para realizar cálculos complejos de manera eficiente. Las GPU son especialmente útiles en el entrenamiento de modelos de aprendizaje automático, ya que permiten procesar grandes volúmenes de datos y realizar operaciones matemáticas complejas a altas velocidades. En el caso de este TFG, la CNN implementada ha sido entrenada utilizando una de las GPU gratuitas ofrecidas por Google Colab.

4.4. Casos de estudio

A la hora de preparar el modelo, se deben realizar distintas pruebas para asegurar su correcto funcionamiento. Principalmente, en estos casos de estudio, se prueba distintas configuraciones del modelo que han sido propuestas a medida que se avanzaba en el trabajo. Por lo tanto, algunas pruebas arrojan mejores resultados que otras.

Principalmente, las pruebas se han realizado teniendo en cuenta los siguientes aspectos:

- Número de épocas
- Técnicas de *data augmentation*
- Número de capas
- *Batch size*

En otras palabras, al estudiar los diferentes casos, se tiene en cuenta si cada prueba tiene más o menos épocas durante el entrenamiento, el tamaño del (*batch size*) en cada caso, el tipo de *data augmentation* aplicado y sus parámetros, así como el número de capas (convolucionales o densas) que presenta el modelo.

5. Resultados

En esta sección, se presentan los distintos resultados y experimentos realizados, los cuales se encuentran divididos en dos grandes categorías: los resultados obtenidos para el uso de LSA, lo cual permitirá comprobar cómo ha evolucionado el modelo, y una sección adicional donde se presentará el resultado final con el uso de la cámara incorporado.

Dentro de cada uno de los apartados de los resultados sobre el *dataset* de la LSA, se puede apreciar que, en la mayoría de los casos, se ha variado el número de épocas de entrenamiento, el número de capas y el *batch size*, así como el tipo de *data augmentation*. El objetivo de este apartado es encontrar el mejor modelo posible, con el mayor porcentaje de aciertos.

En cuanto a las pruebas en un escenario real, aquel en el que las imágenes a clasificar son capturas por la cámara de un ordenador, se realizaron pruebas pruebas relacionadas con la posición de la mano para verificar el correcto funcionamiento del modelo. También se evaluó el efecto de la luz, el fondo y otros aspectos con el fin de probar la eficiencia del modelo.

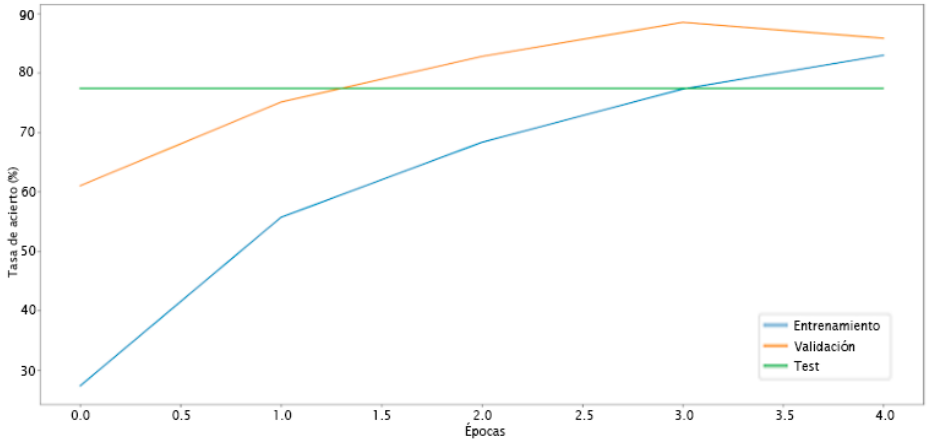
5.1. Resultados sobre el dataset de la LSA

A continuación, se presentan los distintos casos de estudio realizados para la preparación del modelo de la lengua de signos americana.

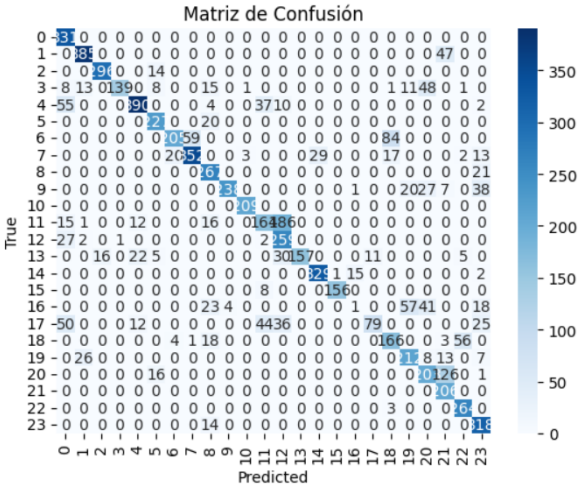
5.1.1. Prueba 1

La primera prueba del modelo de la LSA se realizó empleando 5 épocas de entrenamiento y un tamaño de *batch* de 128 muestras. Es importante destacar que en este primer prototipo carece de capas de *batch normalization*. Se introdujo un aumentado de datos, que incluye variaciones en el *zoom*, el ángulo de las imágenes y un efecto espejo. El valor de F_1 alcanzado por el modelo es del 83%, lo que indica un buen resultado inicial.

En la Figura 5.1a se muestra un tasa de acierto de entorno el 80% en el conjunto de test. La matriz de confusión que se muestra en la Figura 5.1b revela una distribución uniforme, lo que indica que la mayoría de las letras se clasifican correctamente, con algunas excepciones.



(a) Evolución de la tasa de acierto para los conjuntos de entrenamiento, validación y test.



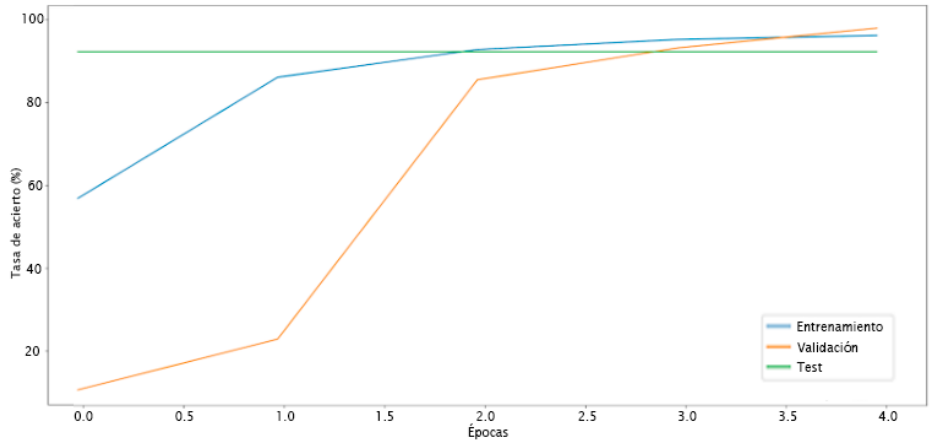
(b) Matriz de confusión para el conjunto de test.

Figura 5.1: Prueba 1.

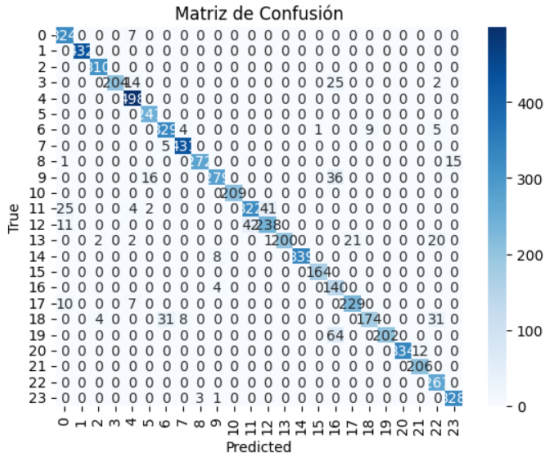
5.1.2. Prueba 2

En esta segunda prueba, se utilizaron nuevamente 5 épocas. El *data augmentation* se mantuvo igual que en la prueba anterior, al igual que el *batch size* de 128. La diferencia en este caso radica en la adición de capas de *batch normalization* al modelo. Se puede apreciar que, para esta prueba, el valor de F_1 del modelo ha aumentado hasta un 93%, lo que indica que la técnica de regularización de *batch normalization* mejora el rendimiento del modelo.

En la Figura 5.2a se observa una pequeña mejora debido a los cambios previamente mencionados, ya que la tasa de acierto de test se asemeja más a las de entrenamiento y validación. En la matriz de confusión que se muestra en la Figura 5.2b, se pueden apreciar muchos menos errores en comparación con la prueba anterior.



(a) Evolución de la tasa de acierto para los conjuntos de entrenamiento, validación y test.



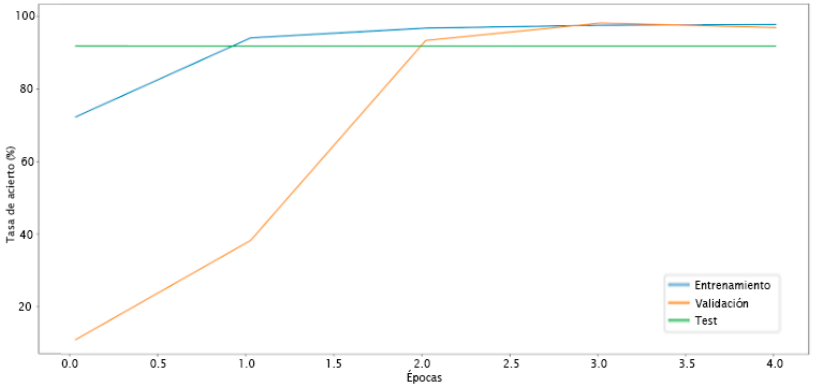
(b) Matriz de confusión para el conjunto de test.

Figura 5.2: Prueba 2.

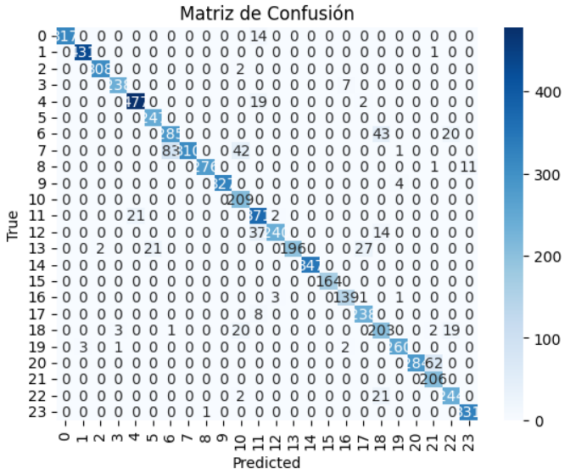
5.1.3. Prueba 3

Para la tercera prueba se utilizaron nuevamente 5 épocas, se mantuvo el *batch size*) en 128 y las capas del modelo no variaron en comparación con la prueba anterior. En esta ocasión, se eliminó el efecto espejo del *data augmentation* utilizado en las dos pruebas anteriores, y se detalló mejor la distorsión por *zoom* y rotación de la imagen.

En esta prueba se obtuvo un valor de F_1 del 92%, lo cual podría parecer indicar una leve empeoramiento del modelo. Sin embargo, al observar la Figura 5.3a, se puede apreciar que los valores de entrenamiento, validación y test son mucho más similares entre sí. De igual manera, en la matriz de confusión de la Figura 5.3b se observan menos errores, lo que indica un resultado más limpio. Por lo tanto, podríamos decir que el efecto espejo aplicado a las imágenes de entrenamiento estaba confundiendo al modelo en algunas ocasiones.



(a) Evolución de la tasa de acierto para los conjuntos de entrenamiento, validación y test.

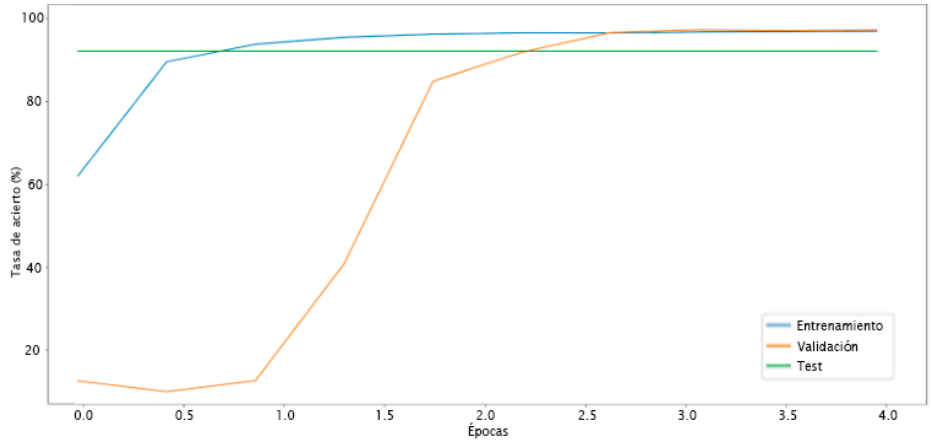


(b) Matriz de confusión para el conjunto de test.

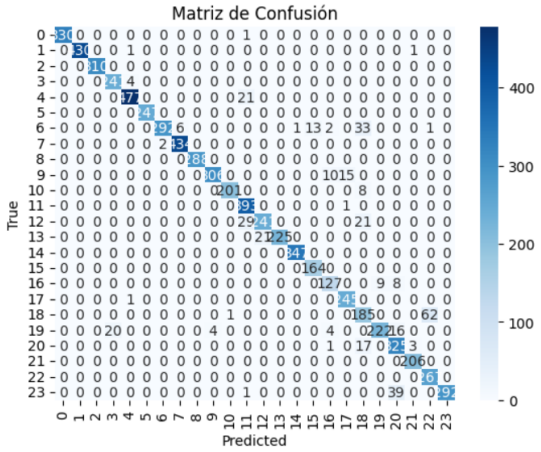
Figura 5.3: Prueba 3.

5.1.4. Prueba 4

Para esta cuarta prueba, se han mantenido las capas y el *data augmentation*. Sin embargo, se ha cambiado la cantidad de épocas, aumentándolas hasta 10, y se ha variado el *batch size* a 256. Se puede observar cómo el valor de F_1 ha aumentado hasta el 94%, lo que indica una clara mejora del modelo. Al observar la Figura 5.4a, se puede apreciar que los valores de entrenamiento, validación y test son más similares entre sí que en la prueba anterior. Además, en la matriz de confusión de la Figura 5.4b, se observan pocos errores de detección, lo que indica una clara mejora en el modelo.



(a) Evolución de la tasa de acierto para los conjuntos de entrenamiento, validación y test.



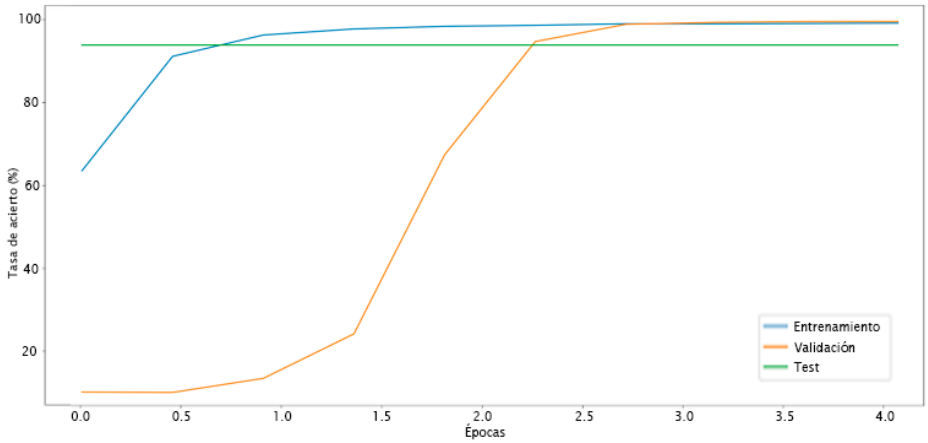
(b) Matriz de confusión para el conjunto de test.

Figura 5.4: Prueba 4.

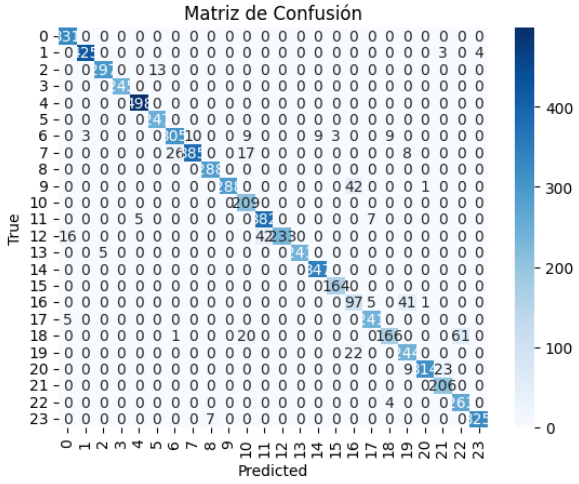
5.1.5. Prueba 5

Para esta quinta prueba se han mantenido los valores de épocas y *batch size* de la cuarta prueba, así como las capas del modelo, que tampoco se han variado. Sin embargo, se ha añadido una nueva capa de *data augmentation* llamada *random brightness*[1], la cual introduce cambios en el brillo de la imagen. Con este cambio, se obtuvo un valor de F_1 del 94%.

Al observar la Figura 5.5a, se pueden apreciar valores prácticamente idénticos en las últimas épocas para las gráficas de entrenamiento y validación. La matriz de confusión de la Figura 5.5b muestra una reducción en los errores sobre los datos de test con respecto a la prueba anterior.



(a) Evolución de la tasa de acierto para los conjuntos de entrenamiento, validación y test.



(b) Matriz de confusión para el conjunto de test.

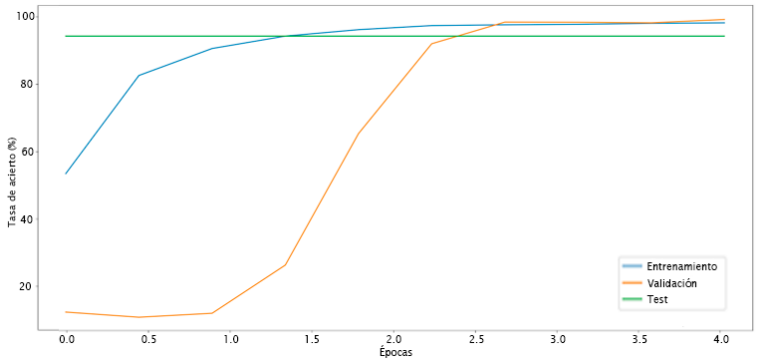
Figura 5.5: Prueba 5.

[1] Véase `https://www.tensorflow.org/api_docs/python/tf/keras/layers/RandomBrightness`

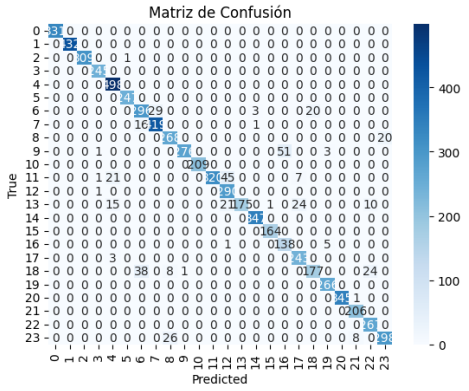
5.1.6. Prueba 6

Para esta sexta prueba se han mantenido los valores de épocas y *batch size* de la quinta prueba, así como las capas del modelo, que tampoco han sido modificadas. Se ha añadido una nueva capa de *data augmentation*: en este caso se ha vuelto a introducir el efecto espejo que previamente se eliminó en la Prueba 3 con el fin de obtener más ejemplos para la mano derecha. Tras la Prueba 5, se hicieron unas primeras pruebas preliminares introduciendo al modelo imágenes capturas por una *webcam*. Ahí se pudo comprobar que cuando los signos eran representados con la mano derecha, el modelo fallaba. Esto se debe a que el *dataset* está sesgado hacia la mano izquierda.

Se ha logrado mantener un valor de F_1 del 94%. Al observar la Figura 5.6a, se puede apreciar que se reducen las diferencias entre la tasa de acierto de test y la de entrenamiento, lo que indica que el efecto espejo es positivo y ayuda al modelo a generalizar mejor. En la matriz de confusión de la Figura 5.6b, se observan tonos de azul más intensos a lo largo de la diagonal principal, lo cual indica que el número de verdaderos positivos ha aumentado.



(a) Evolución de la tasa de acierto para los conjuntos de entrenamiento, validación y test.



(b) Matriz de confusión para el conjunto de test.

Figura 5.6: Prueba 6.

5.2. Demostración

5.2.1. Comparación entre las manos

Para las pruebas finales con la cámara, se ha utilizado el modelo mencionado en la Sección 5.1.6. Durante el proceso de prueba, se decidió comparar los resultados obtenidos al utilizar la mano derecha y la mano izquierda, con el fin de evaluar cualitativamente la mejora introducida por el *data augmentation* de tipo efecto espejo. Por esta razón, se ha preparado la Tabla 5.1 para observar de manera más clara la diferencia entre ambas manos a la hora de ser ser detectadas por el modelo.

Letra	**Mano izquierda**	**Mano derecha**
A	Nada	Nada
B	100%	90%
C	100%	95%
D	70%	90%
E	Nada	Nada
F	100%	100%
G	100%	90%
H	100%	100%
I	85%	Nada
K	70%	Nada
L	100%	90%
M	Nada	Nada
N	Nada	Nada
O	100%	90%
P	100%	70%
Q	100%	95%
R	75%	Nada
S	Nada	Nada
T	100%	90%
U	Nada	Nada
V	90%	95%
W	65%	85%
X	100%	100%
Y	100%	100%

Tabla 5.1: Comparación entre mano derecha y mano izquierda.

Como se puede observar en la Tabla 5.1, el *dataset* está diseñado para ser utilizado con la mano izquierda, ya que en general los valores son mejores. Sin embargo, al haberle aplicado el *data augmentation* previamente mencionado, también se obtiene un buen resultado con la mano derecha. Además, se puede apreciar que las letras A, E, M, N y S son muy similares entre sí, como se muestra en la Figura 4.1, y el modelo no logra diferenciarlas con tanta claridad.

5.2.2. Pruebas con la cámara

A continuación, se va a observar el resultado final de la implementación de la cámara, de manera que se pueda apreciar la solución real y final del modelo. Para estos casos, se van a realizar diferentes pruebas teniendo en cuenta el fondo, la iluminación del entorno y la tonalidad de la piel.

5.2.2.1. Comparación de fondos

(a) Prueba sin fondo. **(b)** Prueba con un fondo de color oscuro.

Figura 5.7: Pruebas sobre el fondo con la mano derecha.

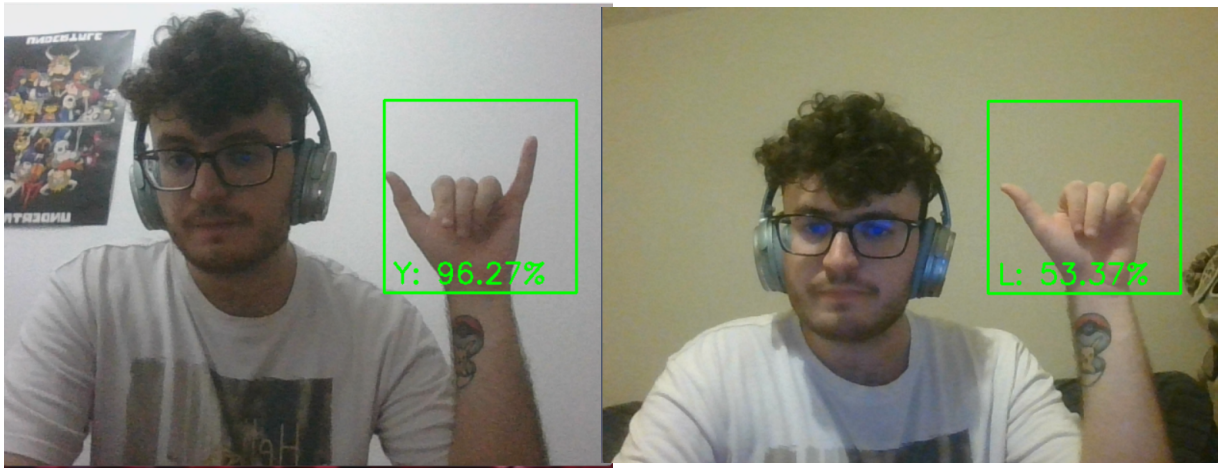
(a) Prueba sin Luz. **(b)** Prueba con un fondo de color oscuro.

Figura 5.8: Pruebas sobre el fondo con la mano izquierda.

5.2.2.2. Comparación de luminosidad

(a) Prueba con buena luz. **(b)** Prueba con peor luz.

Figura 5.9: Pruebas sobre la luz con la mano derecha.



(a) Prueba con buena luz. **(b)** Prueba con peor luz.

Figura 5.10: Pruebas sobre la luz con la mano izquierda.

5.2.2.3. Comparación de tonalidad de la piel

(a) Prueba con piel clara. **(b)** Prueba con piel oscura.

Figura 5.11: Pruebas sobre la tonalidad de la piel con la mano derecha.

(a) Prueba con piel clara. **(b)** Prueba con piel oscura.

Figura 5.12: Pruebas sobre la tonalidad de la piel con la mano izquierda.

5.2.2.4. Discusión de los resultados

Antes de comenzar con la explicación de los resultados obtenidos con la cámara, hay que tener en cuenta que todas las imágenes del *dataset* tienen las mismas condiciones: manos claras, sin fondo y con mucha luz. Sin embargo, en todas las pruebas realizadas se utilizaron ambas manos para poder comprobar el efecto introducido por el *data augmentation* con el fin de clasificar adecuadamente los signos realizados con la mano derecha.

Los resultados observables en la Sección 5.2.2.1 dejan claro cómo el fondo afecta significativamente al resultado final, ya que las imágenes del *dataset* no tienen fondo (o mejor dicho, cuentan con un fondo uniforme de tonos grisáceos).

En la Sección 5.2.2.2, se pueden observar una cosa clara. El *dataset* está formado por imágenes con una iluminación muy buena, por lo que al realizar pruebas con una iluminación diferente, nos encontramos con dos situaciones: por un lado, el modelo detecta correctamente el signo pero con un menor porcentaje de exactitud; por otro lado, confunde signos que puedan ser mínimamente parecidos.

En los resultados de la Sección 5.2.2.3, se encuentra una grata sorpresa. Normalmente, al detectar manos o partes del cuerpo de personas de piel oscura, suelen surgir problemas debido al desequilibrio de los datos disponibles en *datasets* públicos. En este caso concreto, se esperaba obtener resultados sensiblemente peores con las manos de piel oscura que con las manos de piel clara pero, como se puede comprobar, los resultados son muy parecidos.

6. Conclusiones

En esta sección se presentan las conclusiones finales de proyecto, partiendo desde un resumen del trabajo hasta un comentario sobre las posibles futuras mejoras.

6.1. Resumen del trabajo

Se ha conseguido obtener un modelo basado en Redes Neuronales Convolucionales (CNN) capaz de detectar las letras estáticas del alfabeto de la Lengua de Signos Americana (LSA), alcanzando una tasa de acierto del 94%. Además, se ha incorporado este modelo en una interfaz que permite su interacción con la cámara de un ordenador, lo que posibilita la detección en tiempo real de las letras. Los resultados obtenidos en situaciones reales son satisfactorios, ya que con la luz y el ángulo adecuados, se obtiene hasta un 100% de acierto en el reconocimiento de algunos signos.

El modelo consta de dos capas convolucionales encargadas de extraer las características apropiadas de las imágenes de los signos, para que dos capas densas puedan clasificarlos correctamente. Para asegurar una cantidad adecuada de variedades de imágenes, se han aplicado técnicas de *data augmentation* como rotación, cambio de brillo, *zoom* e incluso añadir un efecto espejo a las imágenes de entrenamiento originales.

Se puede confirmar que se han cumplido satisfactoriamente los objetivos propuestos en la Sección 1.1. Al obtener un modelo capaz de reconocer las letras estáticas de la LSA, se han cumplido los objetivos de aprender sobre tecnología de vanguardia en IA, ser capaz de definir y entrenar una red neuronal, y construir un modelo que reconozca la lengua de signos en imágenes. Del mismo modo, al implementar este modelo en un programa capaz de utilizar una *webcam* para hacer pruebas en tiempo real con signos estáticos, se ha logrado cumplir el objetivo de llevar el modelo implementado a una situación real.

6.1.1. Competencias personales adquiridas

Realizar este TFG me ha aportado distintas competencias personales. En primer lugar, el aprendizaje y la familiarización con la lengua de signos me han permitido adquirir una mayor rapidez para trabajar con los signos de la LSA. También he logrado una mayor fluidez de programación al programar en un lenguaje nuevo, como Python, e incluso al trabajar con librerías que me eran poco conocidas hasta que empecé a utilizarlas. En conjunto, todo esto me ha proporcionado habilidades autónomas de investigación, búsqueda e indagación.

6.1.2. Problemas encontrados

A la hora de realizar diversas pruebas con el modelo, se han detectado varios problemas, principalmente al trabajar con la cámara.

Debido a la falta de diversidad en el *dataset*, es necesario que el fondo sea blanco y haya una buena iluminación para que los signos se detecten correctamente. Cualquier movimiento del fondo, cambio en la iluminación o incluso cambios en la tonalidad de piel del usuario afectan al modelo y hace que disminuya su rendimiento, pudiendo, en el peor de los casos, dejar de reconocer la letra signada.

Además, se encontró otro problema que se discutirá en detalle en la Sección 6.2. Se intentó trabajar con un *dataset* de la Lengua de Signos Española (LSE) para poder abarcar ambos idiomas de signos. Sin embargo, esto resultó imposible debido a diversos problemas a la hora de obtener las imágenes. A diferencia de la lengua de signos americana, no existen conjuntos de datos públicos de gran tamaño para la LSE, lo que impide verificar la validez del enfoque basado en CNN. Se puede encontrar más información sobre el *dataset* de la LSE en el Apéndice A.

6.2. Mejoras futuras

Se sabe de la existencia de *datasets* de lengua de signos en distintos idiomas, por lo tanto, con las modificaciones adecuadas, este modelo podría utilizarse también para otros idiomas.

Otro punto importante sería implementar un modelo para identificar signos que requieran movimiento.

Bibliografía

Albawi, S., Mohammed, T. A., y Al-Zawi, S. (2017). Understanding of a convolutional neural network. En *2017 International Conference on Engineering and Technology (ICET)* (pp. 1–6). IEEE.

Benítez-Andrades, J. A. (s.f.). *Inteligencia artificial para aprender lengua de signos.* `https://aprendeconeli.com/inteligencia-artificial-para-aprender-lengua-de-signos/`.

Bradski, G. (2000). The OpenCV library. *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, *25*(11), 120–123.

Challenger-Pérez, I., Díaz-Ricardo, Y., y Becerra-García, R. A. (2014). El lenguaje de programación Python. *Ciencias Holguín*, *20*(2), 1–13.

Cheok, M. J., Omar, Z., y Jaward, M. H. (2019). A review of hand gesture and sign language recognition techniques. *International Journal of Machine Learning and Cybernetics*, *10*, 131–153.

Chollet, F. (2021). *Deep learning with Python*. Simon and Schuster.

Ferstl, Y., Neff, M., y McDonnell, R. (2020). Adversarial gesture generation with realistic gesture phasing. *Computers & Graphics*, *89*, 117–130.

Géron, A. (2022). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, Inc.

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., … others (2018). Recent Advances in Convolutional Neural Networks. *Pattern Recognition*, *77*, 354–377.

Kingma, D. P., y Ba, J. (2015). Adam: A Method for Stochastic Optimization. En Y. Bengio y Y. LeCun (Eds.), *3rd International Conference on Learning Representations, ICLR*. San Diego, USA.

Krizhevsky, A., Sutskever, I., y Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in neural information processing systems*, *25*.

LeCun, Y., Bengio, Y., y Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436–444.

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., y Jackel, L. (1989). Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, *2*.

Newport, E. L., y Meier, R. P. (1985). The Acquisition of American Sign Language. En *The crosslinguistic study of language acquisition* (pp. 881–938). Psychology Press.

Ortiz, I. d. l. R. R. (2005). *Comunicar a través del silencio: las posibilidades de la lengua de signos española* (Vol. 5). Universidad de Sevilla.

Pigou, L., Dieleman, S., Kindermans, P.-J., y Schrauwen, B. (2015). Sign language recognition using convolutional neural networks. En *Computer vision-eccv 2014 workshops: Zurich, switzerland, september 6-7 and 12, 2014, proceedings, part i 13* (pp. 572–578). Springer.

Pérez, J. (s.f.). *Reconocimiento de estados emocionales de personas mediante la voz utilizando algoritmos de aprendizaje de máquina.* `https://www.researchgate.net/figure/Figura-6-Matriz-de-Confusion-Resultados-del-Modelo-SVM-en-la-Prueba-1-Utilizando-el_fig2_329629520`.

Rouhiainen, L. (2018). Inteligencia artificial. *Madrid: Alienta Editorial.*

Lista de Acrónimos y Abreviaturas

AI	Artificial Intelligence.
CNN	Redes Neuronales Convolucionales.
FDP	Final Degree Project.
GPU	Unidad de Procesamiento Gráfico.
IA	Inteligencia Artificial.
LSA	Lengua de Signos Americana.
LSE	Lengua de Signos Española.
OpenCV	Open Source Computer Vision Libreary.
TF	TensorFlow.
TFG	Trabajo Final de Grado.

A. Anexo I

A.1. Lengua de Signos Española

La Lengua de Signos Española (LSE) es una lengua visual-gestual utilizada por la comunidad sorda en España. Esta lengua, al igual que otras lenguas de signos, tiene su propia gramática, sintaxis y léxico. A pesar de que la LSE no tiene una larga historia documentada, se sabe que esta lengua ha existido en España desde la Edad Media (Ortiz, 2005).

Aunque se desconoce mucho sobre los orígenes de la LSE, se sabe que los sordos y las personas que vivían cerca de ellos desarrollaron un sistema de signos rudimentario que les permitía comunicarse. La LSE evolucionó a lo largo de los siglos y se convirtió en una lengua rica y compleja. Sin embargo, durante mucho tiempo, la LSE fue considerada una forma inferior de comunicación y los sordos fueron marginados y discriminados.

Fue en el siglo XVI cuando se produjo un gran avance en la educación de los sordos en España. El fraile benedictino Pedro Ponce de León, natural de la provincia de Burgos, fue el primer educador de sordos conocido en la historia. Ponce de León desarrolló un método de enseñanza basado en el uso de la LSE y en la enseñanza del habla y la lectura labial. Este método fue muy innovador para la época y permitió que los sordos aprendieran a comunicarse y a integrarse en la sociedad.

En el siglo XVIII, el jesuita español Lorenzo Hervás y Panduro, nacido en la provincia de Cuenca, comenzó a estudiar las lenguas de todo el mundo. Hervás y Panduro estaba particularmente interesado en las lenguas indígenas de América Latina y en las lenguas de signos utilizadas por los sordos. Hervás y Panduro se dio cuenta de que las lenguas de signos eran lenguas completas y complejas, con su propia gramática y sintaxis. Además, descubrió que la LSE era muy diferente de otras lenguas de signos utilizadas en Europa y América.

A pesar de estos avances en la educación y el estudio de la LSE, la lengua de signos española no fue reconocida oficialmente como una lengua hasta la década de 1970. Durante mucho tiempo, se consideró que la única forma de comunicarse con los sordos era mediante la oralidad y la lectura labial. Sin embargo, gracias a la lucha de la comunidad sorda y de los educadores de sordos, se logró el reconocimiento de la LSE como una lengua con sus propias características y peculiaridades.

Hoy en día, la LSE es utilizada por una gran parte de la comunidad sorda en España, así como por personas que trabajan con sordos, como intérpretes, educadores y profesionales de la salud. Además, la LSE se ha convertido en un objeto de estudio para lingüistas y antropólogos, que han podido analizar y comparar las lenguas de signos de todo el mundo.

La LSE es una lengua rica y compleja, con una gramática y sintaxis propia. Aunque la LSE tiene algunas similitudes con la lengua española, también tiene muchas diferencias (Ortiz, 2005). Por ejemplo, la LSE utiliza la expresión facial, la dirección y la velocidad del movimiento.[1]

A.1.1. Dataset Lengua de Signos Española

El *dataset* de la LSE se llama *Spanish Sign Language Alphabet (Static)*.[2] Se trata de un *dataset* formado por un total de 1.998 imágenes, con aproximadamente 100 imágenes por letra. A diferencia de su versión americana, la LSE posee un total de 19 letras estáticas y 8 en movimiento. Por lo que, para este *dataset* sería ideal contar con un modelo que detectará signos en movimiento.

A continuación, en la Figura A.1, se puede observar una muestra de los distintos signos que se consideran en el *dataset* de la LSE.

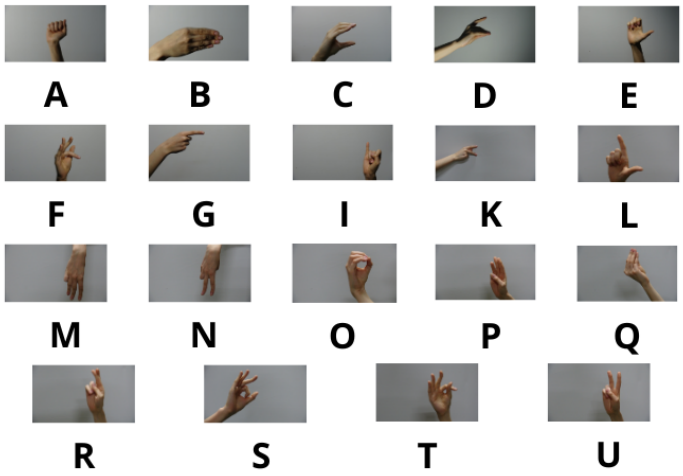


Figura A.1: *Dataset* de LSE.

La Tabla A.1 sintetiza la información que caracteriza a este *dataset*.

	Número de muestras	**Número de clases**	**Tamaño de las imágenes**
LSE	1.998	19	64x64

Tabla A.1: Datos del *dataset* de LSE.

[1] Se recomienda, como lectura complementaria, la entrada del *blog* aportada por la Federación de Personas Sordas de la Comunidad de Madrid: `https://www.fesorcam.org/historia-de-la-lse/`

[2] Disponible en: `https://www.kaggle.com/datasets/kirlelea/spanish-sign-language-alphabet-static`