

CNN para reconocimiento del alfabeto ASL en imágenes

Daniel Queijeiro Albo A01710441

ABSTRACT. El propósito de este documento es presentar el proceso de desarrollo y optimización de redes neuronales convolucionales (CNN) para clasificar señas del alfabeto ASL. Se desarrollaron dos modelos: un modelo base y un modelo optimizado. El objetivo final fue superar las limitaciones del primer enfoque mediante la mejora de la calidad de los datos, técnicas de aumentación robustas y una arquitectura más capaz, logrando una mejora significativa en las predicciones.

1. INTRODUCCIÓN

El lenguaje de señas americano (ASL, por sus siglas en inglés) es un sistema de comunicación visual fundamental para la comunidad sorda. El desarrollo de sistemas automáticos de reconocimiento de señas puede facilitar la comunicación y reducir las barreras de accesibilidad.

Se utilizó el dataset [ASL Alphabet](#) disponible en Kaggle. Este conjunto de datos contiene imágenes de 200 x 200 píxeles en formato RGB, representando diferentes configuraciones de manos correspondientes a cada símbolo del alfabeto ASL.

2. PRIMER MODELO (BASE)

2a. Descripción del Dataset

El dataset está conformado por 87,000 imágenes correspondientes a 29 clases, de las cuales 26 de ellas son para las letras A - Z y las otras 3 para “Space”, “Delete” y “Nothing”.

La inclusión de las 3 clases extra es un aspecto crítico del dataset. En una aplicación en tiempo real (ej. un traductor

de video en vivo), el modelo no solo debe reconocer las letras, sino también:

- Nothing: Identificar cuándo el usuario no está haciendo ninguna señal (ej. la mano está en reposo). Esto es vital para evitar una avalancha de predicciones incorrectas (falsos positivos).
- Space: Reconocer la señal de espacio, permitiendo al sistema delimitar palabras y frases de forma coherente.
- Delete: Entender la señal para borrar o corregir, lo cual es crucial para construir una interfaz de usuario interactiva y funcional.

2b. ETL

Debido al origen del dataset, no fue necesario hacer una limpieza exhaustiva al dataset. Lo principal que se alteró del dataset fue crear una separación de los datos de entrenamiento para crear datos de validación, siguiendo el patrón de 80% para entrenamiento y 20% para validación. El dataset igualmente ya incluye datos para prueba, pero para este estudio se promueve el uso de imágenes propias para las pruebas.

El data augmentation es una técnica que crea variaciones artificiales de las imágenes de entrenamiento mediante transformaciones geométricas y fotométricas. Esto aumenta la diversidad del dataset y mejora la capacidad de generalización del modelo.

En nuestro caso queremos que el modelo aprenda a identificar las señas aún con variaciones en pose, centrado o iluminación, por lo que usando un pipeline de data augmentation realizamos los siguientes cambios (únicamente al conjunto de entrenamiento)

1. RandomFlip("horizontal")
 2. RandomRotation(0.1)
 3. RandomZoom(0.2)
 4. RandomTranslation(0.2, 0.2)
-

2c. Construcción del modelo

Una CNN es un tipo de red neuronal artificial que imita la forma en que el humano procesa la información visual. Son el estándar de la industria para tareas de visión por computadora. Sus componentes clave incluyen:

- Capas Convolucionales (Conv2D): Actúan como detectores de características. Aplican filtros a la imagen de entrada para identificar patrones, como bordes, texturas o formas. En las capas más profundas, estos filtros aprenden a reconocer características más complejas.
- Capas de Agrupación (MaxPooling2D): Reducen las dimensiones espaciales (ancho y alto) de la imagen. Esto hace que el

modelo sea más eficiente y ayuda a que la red sea robusta a pequeñas variaciones en la posición de la señal dentro de la imagen.

- Capas Densas (Dense): Son las capas de una red neuronal tradicional. Se colocan al final de la CNN para tomar las características de alto nivel detectadas por las capas convolucionales y realizar la clasificación final.

Arquitectura del modelo

Se diseñó una arquitectura CNN secuencial que acepta imágenes de tamaño 200 x 200 x 3 (ancho, alto y canales de color). Dicha arquitectura se compone de cuatro bloques convolucionales principales, seguidos de un bloque de clasificación.

En los bloques convolucionales se van aumentando los filtros progresivamente (32, 64, 128 y 256) para que el modelo pueda aprender características desde muy simples hasta muy complejas.

Y en el bloque de clasificación obtenemos al final nuestra probabilidad de que la imagen pertenezca a cada una de las 29 imágenes o mejor dicho, su predicción.

Además, se implementaron 3 configuraciones para mejorar la gestión del entrenamiento. Se agregó un "Early stopping" para que se detenga el entrenamiento automáticamente si la pérdida no mejora después de 10 épocas. También se reduce el learning rate a la mitad si es que la pérdida se estanca durante 5 épocas. Y finalmente, se guarda la mejor versión del modelo (basado en la pérdida) por si el modelo fluctúa en su accuracy.

2d. Resultados

El modelo se entrenó utilizando la configuración descrita anteriormente. El rendimiento se evaluó en los conjuntos de datos en entrenamiento, validación y prueba.

El modelo mostró una curva de aprendizaje estable, con el accuracy aumentando y el loss disminuyendo consistentemente

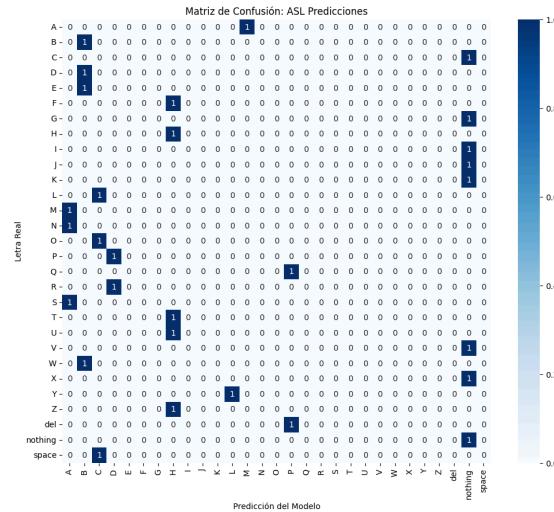
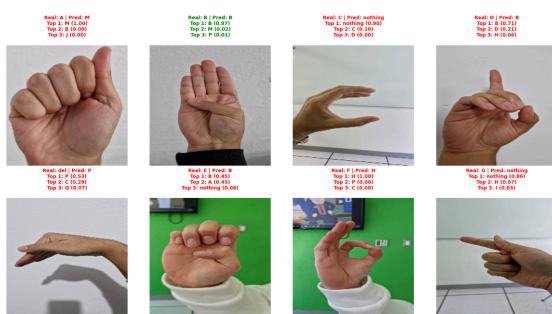
```
accuracy: 0.9475 - loss: 0.1600 - val_accuracy: 0.9502
```

Accuracy en entrenamiento: ~94%

Accuracy en validación: ~95%

Estos resultados iniciales parecían prometedores, sugiriendo que el modelo había aprendido correctamente a clasificar las 29 clases del alfabeto ASL.

Sin embargo, al probar el modelo con imágenes reales tomadas fuera del entorno controlado del dataset, los resultados fueron drásticamente diferentes.



A pesar del excelente rendimiento en validación (~95%), el modelo fallaba constantemente con imágenes del mundo real. Este es un caso clásico de overfitting: el modelo memorizó características específicas del dataset de entrenamiento en lugar de aprender las características generales de las señas.

El dataset original, aunque era grande (87,000 imágenes), proviene de un video donde cada frame se convirtió en una foto del dataset. Esto significa que para cada letra, hay miles de imágenes que son prácticamente idénticas entre sí, con diferencias mínimas de un frame al siguiente:



Como se observa en la imagen, los 8 frames mostrados corresponden a la misma seña "A" pero son casi indistinguibles. El modelo aprende a reconocer estos patrones específicos de píxeles en lugar de la

geometría general de la seña, resultando en un sobreajuste extremo.

2e. Conclusiones del primer modelo

El primer modelo nos demuestra que un alto accuracy en validación no garantiza un buen rendimiento en producción. El modelo estaba severamente sobreajustado a las características superficiales del dataset de entrenamiento, no a la geometría real de las señas ASL.

3. SEGUNDO MODELO OPTIMIZADO

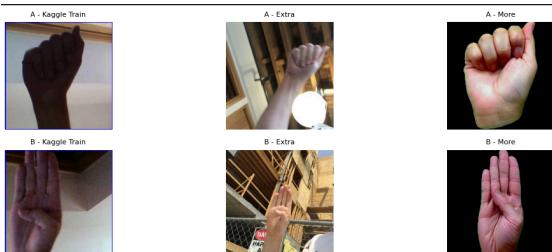
Para superar las limitaciones críticas del primer modelo (sobreajuste y falta de generalización), se desarrolló una segunda iteración con un enfoque completamente diferente: en lugar de mejorar sólo la arquitectura, se atacó el problema desde tres frentes simultáneos: datos, aumentación e inferencia inteligente.

3a. Estrategia de mejora de los datos

Fusión de múltiples datasets

El problema principal del primer modelo era que las imágenes eran muy similares como para permitir al modelo generalizar correctamente. Para solucionarlo, se combinaron 3 datasets diferentes.

1. [ASL Alphabet](#)
2. [ASL Alphabet Test](#)
3. [American Sign Language Dataset](#)

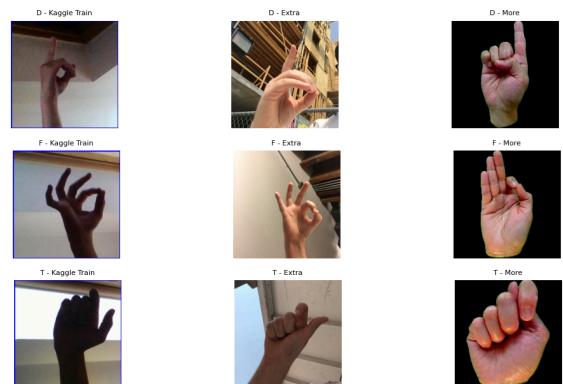


Esta combinación forzó al modelo a aprender las características reales de las señas, ya que no podía solo memorizar el fondo o luz de las fotos.

Reducción y filtrado

Se tomó la decisión de eliminar las clases “del”, “space”, y “nothing”, ya que el tercer dataset no incluía estas 3 clases.

También se identificó que ciertas letras en el tercer dataset venían con inconsistencias visuales en la seña. Específicamente las letras D, F y T.



3b. Mejoras en el data augmentation

Para el segundo modelo se añadieron transformaciones para mejorar el rendimiento del modelo ante cambios de luz.

En el primer modelo ya teníamos las siguientes transformaciones:

1. RandomFlip(“horizontal”)
2. RandomRotation(0.1)
3. RandomZoom(0.2)
4. RandomTranslation(0.2, 0.2)

Y ahora también usamos:

5. RandomBrightness(0.2)
6. RandomContrast(0.2)

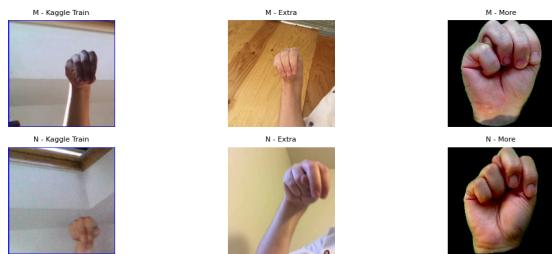
Gracias a estas transformaciones el modelo debe generalizar de mejor manera las imágenes tomadas en condiciones de luz diferentes a las del entrenamiento.

3c. Mejoras en la arquitectura

Basándonos en los resultados que obtuvimos en el primer modelo, realizamos cambios a la arquitectura para aumentar la capacidad del modelo.

Mayor capacidad en capas densas

El primer modelo utilizaba una capa densa de 256 neuronas antes de la capa de salida. Para el segundo modelo, se aumentó a 512 neuronas. Esta decisión se tomó considerando que ahora el modelo debe aprender de 3 datasets diferentes con mayor variabilidad. Una red con mayor capacidad puede capturar patrones más complejos y sutiles, como las diferencias finas entre señas visualmente similares (por ejemplo, distinguir entre M y N, que difieren solo en la posición de los dedos).



Cambio a Flatten

El primer modelo utilizaba GlobalAveragePooling2D, una técnica que toma los mapas de características de la última capa convolucional y calcula el promedio de cada uno. Aunque esto reduce drásticamente el número de parámetros y ayuda a prevenir el sobreajuste, tiene una desventaja crítica al perder información espacial.

En el reconocimiento de señas ASL, la posición exacta de cada dedo es fundamental. Por ejemplo, las letras M y N se distinguen únicamente por cuántos dedos están extendidos sobre el pulgar. Al promediar los mapas de características, el modelo pierde la noción de dónde está cada característica, sólo sabe qué características están presentes.

El segundo modelo utiliza Flatten, que simplemente “aplana” todos los valores de los mapas de características en un vector largo. Esto preserva la información posicional completa, permitiendo que las capas densas posteriores aprendan relaciones espaciales entre diferentes partes de la mano a cambio de un mayor número de parámetros.

3d. Primeros resultados

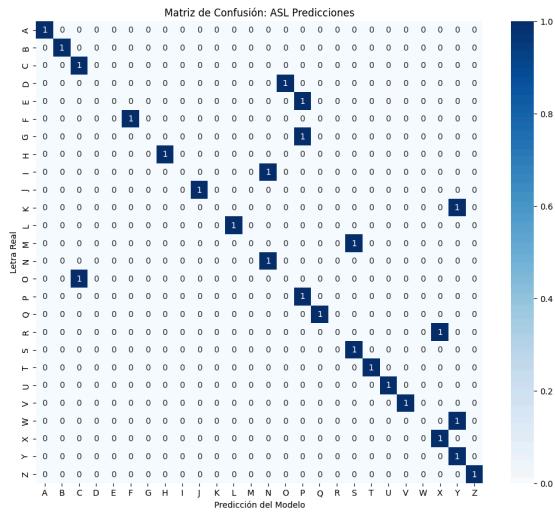
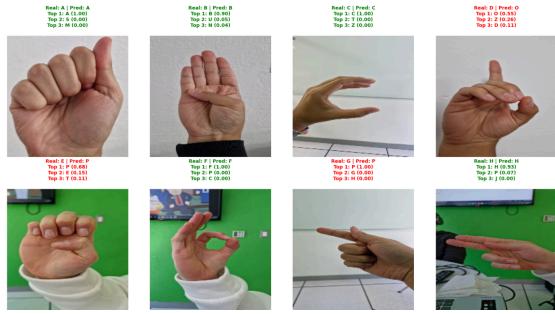
Después de entrenar el segundo modelo con las mejoras descritas (fusión de datasets, aumento de iluminación, arquitectura mejorada), se procedió a evaluar su rendimiento con imágenes reales.

Métricas de Evaluación

Para evaluar el rendimiento del modelo se utilizan tres métricas principales:

- Precisión: Cuando el modelo dice "A", ¿qué tan seguido tiene razón?
- Recall: De todas las "A" reales, ¿cuántas encontró el modelo?
- F1-Score: Promedio balanceado entre Precision y Recall

Si un modelo se confunde y predice letras incorrectas (falsos positivos) se puede decir que tiene una precisión baja. En cambio, si al modelo se le “escapan” letras que debería detectar (falsos negativos) tiene un recall bajo.



La precisión del modelo de 65% es una mejora, pero las métricas de recall (0.65) y F1-Score (0.56) nos muestra que el modelo sigue siendo inconsistente y propenso a errores en un entorno real. Al analizar las imágenes de prueba, se identificó otro problema, el ruido del fondo.

Las imágenes del dataset de entrenamiento tienen la mano centrada y ocupando la mayor parte del frame. En contraste, las imágenes reales tienen fondos complejos (muebles, paredes con texturas), la mano en diferentes posiciones del frame, y diferentes proporciones mano/fondo. El modelo aprendió a reconocer señas, pero se confunde cuando hay información irrelevante alrededor de la mano.

A pesar de las mejoras implementadas, el modelo aún comete varios errores con imágenes reales:

- D confundida con O
- E confundida con P

- G confundida con P
- I confundida con N
- K confundida con Y
- M confundida con S
- O confundida con C
- R confundida con X
- W confundida con Y

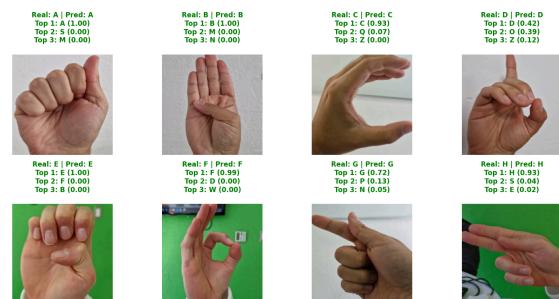
3e. Eliminación de ruido

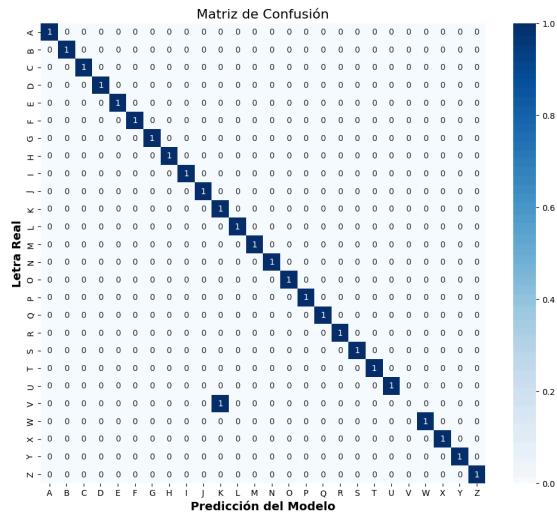
Para resolver el problema del ruido del fondo sin necesidad de reentrenar el modelo, se implementó una estrategia de preprocessamiento inteligente a las imágenes antes de pasarlas por el modelo.

Se implementó la función “predict_smart” que utiliza la librería “MediaPipe” para aislar la mano antes de pasarla al modelo. Primero se calcula el bounding box de la mano y un poco de espacio extra por seguridad y se recorta a 200x200 la imagen y se le da al modelo.

3f. Resultados con preprocessamiento

Gracias al preprocessamiento y eliminación de ruido nuestro modelo logra mejorar bastante. Obtenemos una precisión de 96.15%, acertando a 25 de 26 letras.





El incremento en precisión a 96% nos indica que el modelo con la función extra redujo significativamente los falsos positivos. También, con la mejora al recall (0.65 a 0.96) nos demuestra que el modelo ha reducido los falsos negativos. Finalmente, el F1-Score (0.56 a 0.95), confirma que el modelo y la función han logrado obtener un buen resultado con imágenes del mundo real.

Únicamente la letra “V” es confundida con “K”, debido a que ambas señales son muy similares.



4. CONCLUSIONES

Este proyecto me ha hecho aprender que el éxito de un modelo de Deep Learning no se obtiene únicamente por la complejidad de su arquitectura, sino en la calidad de los datos con los que se entrena. Uno de los

puntos más importantes fue comprobar que un alto accuracy durante la etapa de validación puede ser engañoso si el dataset de entrenamiento no refleja el mundo real. El primer modelo, a pesar de sus métricas casi perfectas, falló al enfrentarse a imágenes con fondos y condiciones de luz no controladas, mostrando la necesidad de evaluar los modelos en escenarios que simulen su uso final.

La estrategia de fusionar múltiples datasets permitió que el modelo aprendiera características generalizables en lugar de memorizar patrones específicos de un solo entorno.

Finalmente, la implementación de MediaPipe transformó los resultados intermedios del modelo a resultados precisos, demostrando que normalizar la entrada puede ser la clave para tener un gran modelo.

Trabajo a futuro

El siguiente paso es volver a agregar las clases “Space”, “Del”, “Nothing” mediante la recolección de un dataset balanceado que mantenga los estándares de calidad establecidos.

También, existe una gran oportunidad para explorar técnicas de Transfer Learning utilizando arquitecturas preentrenadas más profundas, como ResNet o EfficientNet. Estas redes, al haber sido entrenadas con millones de imágenes, podrían ofrecer una capacidad de extracción de características aún mayor, potencialmente mejorando la robustez del modelo.

El objetivo final de estas mejoras sería el desarrollo de una aplicación de traducción

en tiempo real que integre el modelo optimizado. Una aplicación de este tipo no solo serviría como una demostración técnica, sino que tendría un impacto social al facilitar la comunicación y la accesibilidad para la comunidad de usuarios del lenguaje de señas.

5. REFERENCIAS

IBM. (2025, 17 noviembre). Convolutional

Neural Networks. *IBM*.

<https://www.ibm.com/think/topics/convolutional-neural-networks>

Convolutional Neural Network (CNN). (s.

f.). TensorFlow.

<https://www.tensorflow.org/tutorials/images/cnn>

Murel, J., PhD, & Kavlakoglu, E. (2025,

29 enero). Aumento de datos. *IBM*.

<https://www.ibm.com/es-es/think/topics/data-augmentation>

Mucci, T. (2025, 17 noviembre).

Overfitting vs. Underfitting. *IBM*.

<https://www.ibm.com/think/topics/overfitting-vs-underfitting>

Murel, J., PhD, & Kavlakoglu, E. (2025b,

noviembre 27). Matriz de

confusión. *IBM*.

<https://www.ibm.com/mx-es/think/topics/confusion-matrix>

Wilber, J. (s. f.). *Train, Test, and*

Validation Sets. MLU-Explain.

<https://mlu-explain.github.io/train-test-validation/>