Clasificación de Lenguaje de Señas con CNN

Jorge Martínez López
Tecnológico de Monterrey campus Querétaro
Querétaro, México
A01704518@tec.mx
jorgemartinez2555@hotmail.com

***Abstract*— This work explores the automatic classification of sign language using convolutional neural networks (CNN). CNN models, MobileNet, and an improved version of MobileNet were implemented to analyse their effectiveness on different datasets and test conditions.**

I. INTRODUCCIÓN

La comunicación a través del lenguaje de señas es esencial para millones de personas en el mundo. Sin embargo, dentro de sectores de la sociedad, existen limitantes en la comunicación e interacción afectiva con personas con discapacidad auditiva, limitando su acceso a servicios cotidianos.

En este contexto, el avance en técnicas de visión por computadora, específicamente las redes neuronales convolucionales (CNN), abre nuevas posibilidades para desarrollar sistemas de reconocimiento automático del lenguaje de señas que traduzcan señales manuales y gestos en tiempo real.

Esta tecnología tiene el potencial de mejorar significativamente la inclusión social y la accesibilidad en diversos sectores, facilitando la comunicación entre personas sordas y oyentes.

Este proyecto busca explorar cómo el diseño e implementación de modelos basados en CNN pueden contribuir a superar las barreras de comunicación y acercar las oportunidades de interacción con todas las personas.

Objetivo

Generar múltiples Red Neuronal Convolucional para detección de lenguaje de señas, con el fin de determinar su efectividad en su implementación y en su desenvolvimiento en la elaboración tarea.

II. HISTORIA DL

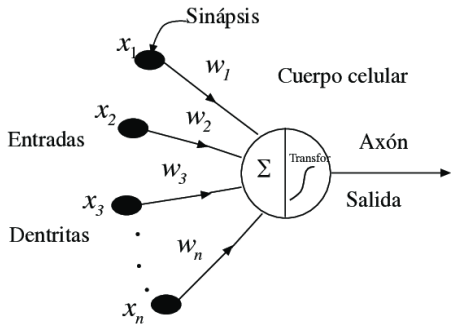
Los orígenes de la inteligencia artificial surgen entre 1943-1955, parte de tres fuentes de conocimiento de la época, las cuales radica en la filosofía básica y el funcionamiento de las neuronas, el análisis de la Lógica proporcional de Russell y la Teoría Computacional de Turing, estos trabajos fueron los cimientos de lo que se conoce hoy como redes neuronales. Sin embargo, hasta el 2012 fue que empezó a tener avances significativos las redes neuronales gracias a su desempeño en competencias, ya que la precisión para clasificar imágenes aumentaba a la medida de que se usaban diferentes arquitecturas y dataset para entrenar los modelos neuronales, de tal manera que su asertividad no bajaba del 80% de precisión, algo que con los algoritmos de Machine Learning difícilmente es alcanzable (*Artificial Intelligence: A Modern Approach, 4th Global Ed.*, 2022).

III. MODELOS REDES NEURONALES

Los modelos de redes neurales cuentan con una estructura base. Primeramente, debemos entender que una red está constituida con una red de perceptrones, un perceptrón consiste un modelo de regresión logística que puede ser 0 o 1, a su vez se dice que una red neuronal densa es una red de regresiones logísticas que se interconectan entre sí, solo que podemos cambiar las funciones de activación de cada perceptrón para mejorar su rendimiento.

A. Perceptrón

Como lo he mencionado anteriormente un perceptrón es la adaptación de un modelo de regresión logística con la posibilidad de cambiar las funciones de activación, esta es la única diferencia entre perceptrón y un modelo de regresión logística.



Img 1. Estructura de un perceptrón en diagrama.

Un perceptrón matemático se puede ver de la siguiente manera, siempre y cuando sea sigmoidal la función de activación.

$$Y = \frac{1}{1 + e^{-(X_n w_n + B)}}$$

"Y" es la salida de la neurona, "X" es la entrada de la neurona, "W" es el coeficiente o peso del modelo y "B" es el sesgo o el error del modelo.

La gran ventaja de un perceptrón es que es una regresión lineal por su función de activación y existen varias funciones de activación, y cada una tiene su propósito:

- Sigmoidal
- ReLU (Rectified Linear Unit)
- Tanh (Tangente hiperbólica)
- Leaky ReLU

- ELU (Exponential Linear Unit)
- Softmax (Normalización)
- Swish
- GELU (Gaussian Error Linear Unit)

Función	Rango de salida	No linealidad	Vanishing Gradient	Uso común
ReLU	[0, ∞)	Sí	No	Capas ocultas de CNNs, redes profundas
Sigmoide	(0, 1)	Sí	Sí	Clasificación binaria
Tanh	(-1, 1)	Sí	Sí	Capas ocultas de redes recurrentes
Leaky ReLU	(-∞, ∞)	Sí	No	Mitigar neuronas muertas
ELU	(-α, ∞)	Sí	No	Alternativa a ReLU
Softmax	(0, 1)	No (normaliza)	No	Clasificación multiclase
Swish	(-∞, ∞)	Sí	No	Redes profundas, variantes de ReLU
GELU	(-∞, ∞)	Sí	No	Transformers, redes profundas

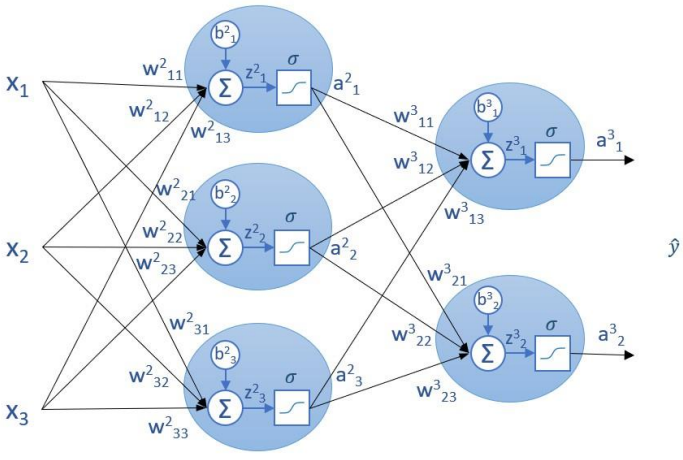
Img 2. Tabla de las funciones de activación.

Como se puede ver, existen diferentes variantes de funciones de activación y cada una funciona mejor que otra, ya que dependerá de la arquitectura que se esté manejando.

B. Red Neuronal Densa

Una Red Neuronal Densa está compuesta por capas de perceptrones, y un modelo neuronal puede ocupar desde N capas y N neuronas por capa, y la definición de la arquitectura dependerá de que tan complejo es el problema y que tan rápido quieres que un modelo se entrene.

Por otro lado, se puede hacer la analogía que una red neuronal es una matriz de regresiones logística con una salida, y lo mencionado anteriormente lo describe la siguiente imagen.



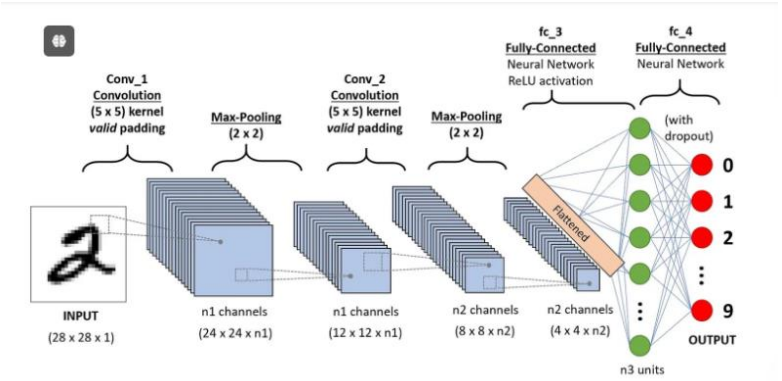
Img 3. Diagrama de una Red Neuronal, de dos capas, de tres neuronas de entrada y dos salidas.

Al tener un diagrama que muestre la interconectividad de las neuronas, es más simple interpretar su funcionamiento interno de las redes neuronales, y este es el principio que utilizan la mayoría de las arquitecturas actuales, en la cual lo que cambia es la función de activación, entonces tenemos en una red neuronal tenemos entradas y luego salidas de la primera capa, que a su vez se convierten entradas de la segunda capa neuronal, y la salida de esta capa, serían los valores esperados, y así para N capas de la red.

C. Red Neuronal Convolucional

Las Redes Neuronales Densa son comúnmente para variable numéricas, que se puede extrapolar para la solución de diferentes situaciones, siempre y cuando utilicen variables numéricas y estructura de datos tabulares.

Sin embargo, al trabajar con información espacial como imágenes o videos, ocupamos una diferente estructura de red neuronal que mejore el tiempo de entrenamiento y que aprenda lo suficiente para clasificar las imágenes.



Img 4. Diagrama de funcionamiento de una CNN.

Las Redes Neuronales Convolucionales o inglés Convolucional Neural Network (CNN), este de tipo de arquitectura consiste en un preprocesamiento para extraer la información importante de las imágenes con el objetivo que cada neurona aprenda eficientemente la información de las imágenes.

Cabe recalcar, una foto es un representación de en dos dimensiones de tres canales (R,G,B), no obstante, en la imagen anterior está en escala de grises, por lo que es de un canal (28,28,1), esto se representa que tenemos una imagen de 28px * 28px en un canal de escala de grises, el fin de esta anotación es poder pasar los parámetros adecuados a nuestro modelo para que aprenda, ejemplo: si estaríamos trabajando con redes densas, tendríamos una red de (28px*28px)784 neuronas de entrada y 10 neuronas de salida, representando cada neurona de salida una clase (0..9).

Por otra parte, queremos agregar mayor complejidad y poder descriptivo para nuestro modelo, por ende, primeramente, se le aplicará filtros para extraer la información esencial de la foto, así también reducir el consumo de recursos al momento de entrenar nuestro modelo.

1. Convolución

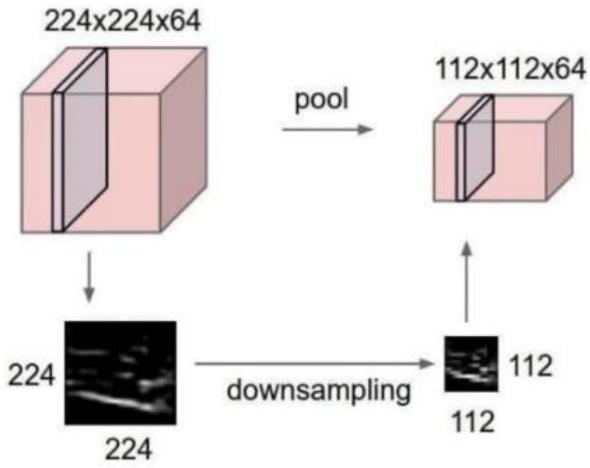
En procesamiento de imágenes y redes neuronales convolucionales (CNNs), un filtro o kernel es una pequeña matriz de pesos que se aplica sobre una imagen de entrada mediante una operación llamada convolución. Matemáticamente, esta operación se puede ver como un producto punto entre el filtro y una región específica de la imagen, creando un nuevo valor en la posición correspondiente del mapa de características.

El filtro "desliza" sobre la imagen en pasos, y cada aplicación del producto punto da lugar a un valor que representa una combinación de los píxeles bajo ese filtro. Este proceso ayuda a extraer características como bordes, texturas y patrones específicos de la imagen.

2. *Max-pooling*

Luego de aplicar filtros, se utiliza una operación llamada max-pooling para reducir la dimensionalidad del mapa de características generado por los filtros y, al mismo tiempo, preservar las características más importantes. En el max-pooling, una ventana de tamaño fijo (por ejemplo, 2x2 o 3x3) se desplaza a través del mapa de características con un paso definido, conocido como stride.

En cada posición de la ventana, se selecciona el valor máximo dentro de la región que cubre. Este valor se convierte en el nuevo valor de la salida en esa posición, lo que permite reducir la resolución espacial del mapa de características mientras se conservan las características más relevantes.



Img 5. Diagrama de funcionamiento Max-pooling.

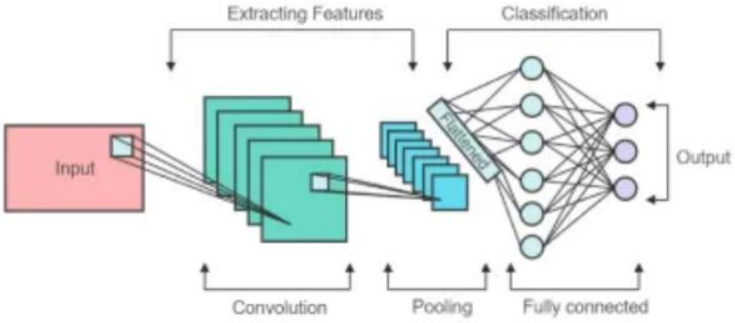
Esta combinación de convoluciones y max-pooling permite a las CNNs extraer patrones complejos y generalizar mejor al procesar imágenes de distintas escalas y posiciones.

3. Flatten

Al últimos, tenemos el flatten es una operación clave para las CNN, ya que convierte un tensor multidimensional en un vector unidimensional, con el objetivo de conectar capas de las características extraídas (convolucionales y de pooling) a las capas completamente conectadas.

Img 6. Diagrama de funcionamiento flatten, transformar de matriz a vector unidimensional.

Al entender que la esencia de una CNN es combinar pre-procesos de transformación para agregar información esencial para entrenar nuestro modelo.



Img 7. Diagrama de funcionamiento Max-pooling.

En el diagrama podemos observar que la convolución y el pooling son la extracción de los inputs o features y la unión entre el preprocesado y la red neuronal es la función flatten, al final tenemos nuestras salidas, que serán activadas por una función softmax.

D. Backpropagation

Cabe resaltar, que el método de cálculo que se utilizan para entrenar las redes neuronales lleva el nombre backpropagation o propagación hacia atrás de los errores. En una red neuronal de las capas anteriores depende directamente del error de las capas posteriores, es por ello por lo que podemos ir analizando y optimizando la propagación del error hacia atrás en la red, para hacer el cálculo se utiliza derivadas parciales para cálculo del error (Vive UNIR, 2023).

E. Hiperparámetros Redes Neuronales

Los hiperparámetros son variables que no se aprenden directamente del proceso de entrenamiento del modelo. Se establece antes del proceso del entrenamiento de la red y afectan en la forma en que se entrena el modelo.

Los parámetros de una red neuronal son los valores (pesos y sesgos) que aprende el algoritmo durante el proceso de entrenamiento para minimizar la función de coste y mejorar las predicciones, que en otras palabras son las conexiones ponderadas entre las neuronas en la red.

En los hiperparámetros lo podemos clasificar en dos grupos:

1. Nivel estructural; número de neuronas por capas, número de capas, inicialización de los pesos, funciones de activación, etc…
2. Nivel del algoritmo de aprendizaje; epochs, momentum, learning rate, batch size, etc…

Por otro lado, los hiperparámetros que tienen el mayor impacto en el rendimiento y capacidad de generalización

de la red neuronal son (Javier, 2024):

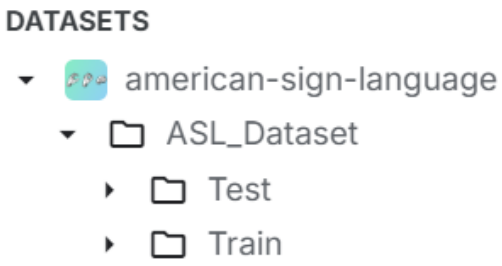
- Numero de capas y la capacidad de neuronas
- Tasa de aprendizaje
- Regularización
- Tamaño del lote
- Función de activación

IV. DATASET

En este proyecto abordaremos el lenguaje de señas, por lo que se ocupamos un dataset de Kaggle "American Sign Langage", el cual cuenta con 28 clases, que son las 26 letras del vocabulario americano, espacio y vacío, la información total son 4.98Gb entre la sección de entrenamiento y prueba, cada imagen es de 400px*400px de tres canales (RGB), en formato JPG. Dataset

V. ANÁLISIS DEL DATASET

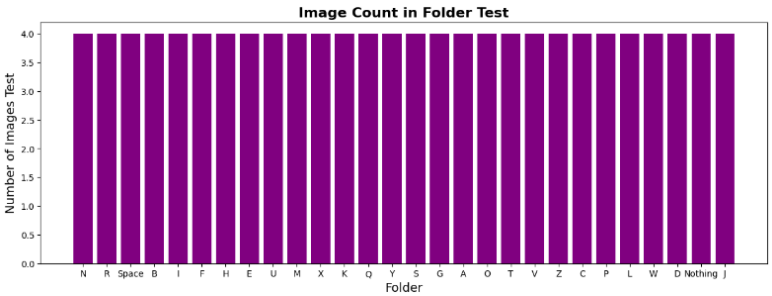
La estructura del dataset consiste en folder de entrenamiento y prueba, cada folder tiene sub-folder con las 28 clases, y cada sub-folder tiene guardado las fotos correspondientes a su clase.



Img 8. Estructura del dataset american-sign-language, la cual será usada para el entrenamiento de nuestros modelos.

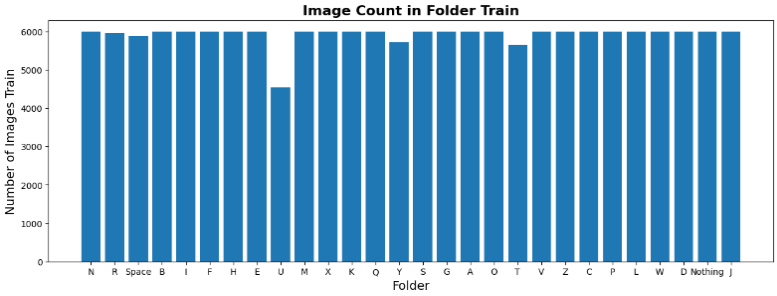
En el set de entrenamiento y prueba tienen las siguientes clases guardas: 'N', 'R', 'Space', 'B', 'I', 'F', 'H', 'E', 'U', 'M', 'X', 'K', 'Q', 'Y', 'S', 'G', 'A', 'O', 'T', 'V', 'Z', 'C', 'P', 'L', 'W', 'D', 'Nothing', 'J'.

La distribución del folder test y train:



Graf 1. Distribución de fotos por label del folder de prueba.

En el folder Test está conformado por las 28 clases mencionadas anteriormente, y cada clase cuenta con 4 imágenes para hacer predicciones.



Graf 2. Distribución de fotos por label del folder de entrenamiento.

En el folder Train está conformado por las 28 clases mencionadas anteriormente, y cada clase cuenta con más 4,000 imágenes, no obstante, la distribución no es homogénea.

A continuación, haremos la división de nuestro dataset para definir set de entrenamiento, validación y prueba, para ello utilizaremos la extensión ImageDataGenerator de Keras para hacer directo la división de los datos y transformaciones pertinentes a las imágenes.

El folder Train cuenta 165,670 imágenes totales, por lo que, se ocuparan 90% para entrenamiento y 10% validación, y el folder Test son 112 imágenes totales, estas imágenes las ocuparemos para predicciones de nuestro modelo. Además, para los tres sets, se distribuirán en batches de 32 imágenes, y un reajuste de tamaño de las imágenes de 224px * 224px.

Debo de resaltar, que, al set de entrenamiento y validación, aplicaremos normalización (0 a 1), cortes aleatorios de hasta 20% de la imagen, zoom de hasta 20% de la imagen, giramos aleatoriamente las imágenes hasta 5 grados, cambio de altura y ancho de las imágenes hasta un 3%, y reserva el 10% de los datos para su validación, rango de iluminación de 50% a 175%.

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    rotation_range=5,
    width_shift_range=0.03,
    height_shift_range=0.03,
    brightness_range=[0.5, 1.75],
    validation_split=0.1,
    fill_mode='nearest',

)
```

Img 9. Argumentación de datos para entrenamiento y validación.

Las transformaciones anteriores tienen el propósito de aumentar la cantidad de los datos y variedad de las imágenes con el fin de evitar el overfitting.

De otra manera, al set de prueba solo vamos a normalizar (0 a 1) las imágenes.

```
test_datagen=ImageDataGenerator(rescale=1./255)
```

Img 10. Normalización para el set de entrenamiento.

Hacemos nuestro Split de sets para generar validación, prueba y entrenamiento.

```
###train generator
train_generator=train_datagen.flow_from_directory(
    train_path,
    target_size=(224,224),
    batch_size=32,
    class_mode='categorical',
    subset='training', #90%

)

#### validation generator
validation_generator = train_datagen.flow_from_directory(
    train_path,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='validation', #10%

)


#### test generator
test_generator=test_datagen.flow_from_directory(
    test_path,
    target_size=(224,224),
    batch_size=112,
    class_mode='categorical',
)
```

Img 11. División de los sets, entrenamiento, prueba y validación.

VI. MODEL RED NEURONAL CONVOLUCIONAL

Al tener los datos que se utilizarán para entrenar, validar y prueba, podemos empezar con la construcción de nuestra red neuronal convolucional, anteriormente mencionamos que consiste un preprocesado en donde se extrae características esenciales de las imágenes. Código: programa

Para la construcción de nuestro modelo estaremos trabajando con Keras. Importamos las dependencias del framework que ocuparemos.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GlobalAveragePooling2D , Dropout , Dense, Conv2D, MaxPooling2D, Flatten, Input
```

Img 12. Librerías utilizadas para construir nuestra red neuronal.

```
cnn=Sequential()
cnn.add(Input(shape=(224, 224, 3)))
# Primera capa convolucional
cnn.add(Conv2D(32, (3,3), activation='relu', kernel_regularizer=l2(0.001)))
cnn.add(BatchNormalization())
cnn.add(MaxPooling2D(2,2))
cnn.add(Dropout(0.2))

# Segunda capa convolucional
cnn.add(Conv2D(64, (3,3), activation='relu', kernel_regularizer=l2(0.001)))
cnn.add(BatchNormalization())
cnn.add(MaxPooling2D(2,2))
cnn.add(Dropout(0.3))

# Tercera capa convolucional
cnn.add(Conv2D(128, (3,3), activation='relu', kernel_regularizer=l2(0.001)))
cnn.add(BatchNormalization())
cnn.add(MaxPooling2D(2,2))
cnn.add(Dropout(0.3))

# Cuarta capa convolucional
cnn.add(Conv2D(256, (3,3), activation='relu', kernel_regularizer=l2(0.001)))
cnn.add(BatchNormalization())
cnn.add(MaxPooling2D(2,2))
cnn.add(Dropout(0.3))

# Capa de Pooling global
cnn.add(GlobalAveragePooling2D())

# Capa densa con Dropout final
cnn.add(Dense(500, activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(units=train_generator.num_classes, activation='softmax'))
```

Img 13. Arquitectura de nuestro modelo CNN.

1. **Capa de entrada**: Recibe imágenes de tamaño 224x224 con 3 canales (RGB).
2. **Primera capa convolucional**:
   - Convolución con 32 filtros de tamaño 3x3, activación ReLU y regularización L2 de 0.01.
   - Normalización por lotes, seguido de MaxPooling 2x2 y Dropout de 0.2.
3. **Segunda capa convolucional**:
   - Convolución con 64 filtros de tamaño 3x3, activación ReLU y regularización L2 de 0.01.
   - Normalización por lotes, seguido de MaxPooling 2x2 y Dropout de 0.3.
4. **Tercera capa convolucional**:
   - Convolución con 128 filtros de tamaño 3x3, activación ReLU y regularización L2 de 0.01.
   - Normalización por lotes, seguido de MaxPooling 2x2 y Dropout de 0.3.
5. **Cuarta capa convolucional**:
   - Convolución con 256 filtros de tamaño 3x3, activación ReLU y regularización L2 de 0.01.
   - Normalización por lotes, seguido de MaxPooling 2x2 y Dropout de 0.3.

6. **Capa de Pooling global**:
    - GlobalAveragePooling para reducir la dimensionalidad antes de las capas densas.
7. **Capa densa con Dropout final**:
    - Capa densa con 500 unidades, activación ReLU y Dropout de 0.5.
    - Capa de salida con una unidad por clase y activación softmax para clasificación.

```
cnn.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=[
        Recall,
        Precision,
        'accuracy'
    ]
)
# Use ReduceLROnPlateau to reduce learning rate when the validation loss plateaus
lr_reduction = ReduceLROnPlateau(monitor='val_loss', patience=3, factor=0.5, min_lr=1e-6)
```

Img 14. Optimizadores de nuestro modelo CNN.

Utilizamos Adam con una tasa de aprendizaje de 0.01, el cual es usado por su eficiencia y capacidad de adaptación. En la función de pérdida, usamos crossentropy, la cual es adecuado para problema de clasificación múltiple. Además, vamos a ocupar las métricas recall y precisión, con el fin de identificar correctamente las clases y evitar falsos positivos, también ocupamos acurracy para medir la porción de predicciones correctas.

Cabe mencionar, que utilizaremos reducción de tasa de aprendizaje, cuando el valor de perdida si no mejora.

La función ReduceLROnPlateau:

- **Monitor**: val_loss (la pérdida de validación).
- **Paciencia**: 3 (reduce la tasa de aprendizaje si la pérdida de validación no mejora en 3 épocas consecutivas).
- **Factor**: 0.5 (la tasa de aprendizaje se reduce a la mitad cada vez que se activa).
- **min_lr**: 1e-6 (la tasa de aprendizaje mínima, para evitar que disminuya excesivamente).

Entrenamos nuestro modelo, una vez construido.

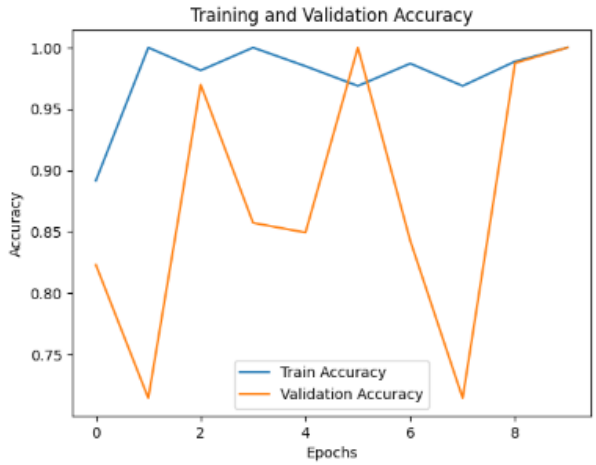
```
history=cnn.fit(train_generator,
                epochs=10,
                validation_data=validation_generator,
                steps_per_epoch=train_generator.samples // train_generator.batch_size,
                validation_steps=validation_generator.samples // validation_generator.batch_size,
               callbacks=[lr_reduction]
               )

results = cnn.evaluate(test_generator)
print("Test loss and accuracy:", results)
```

Img 15. Señalizar el proceso de entrenamiento.

El modelo, tardó en entrenarse 3.5hrs por 10 épocas.

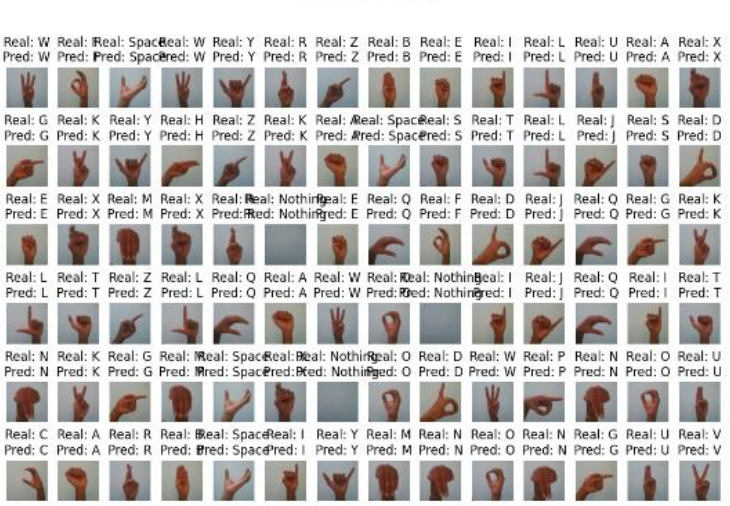
Graf 3. Gráfica de perdida entrenamiento contra validación en 10 épocas.

El modelo presenta en las últimas dos épocas una mejora significativa, podría indicar que el modelo pudiera haber aprendiendo bien sobre los datos de entrenamiento, no obstante, las fluctuaciones sugieren que el modelo tiene dificultades para generalizar los datos de validación, y de que el modelo no es lo suficiente robusto para la cantidad de datos.



Graf 4. Gráfica de precisión entrenamiento contra validación en 10 épocas.

En la gráfica anterior, se puede observar el accuracy del modelo, entrenamiento contra validación. El modelo presenta algunas fluctuaciones el de validación, esto puedo significar dos cosas que el modelo está overfitting o underfitting, debido a sus fluctuaciones de precisión, para saber si el modelo está overfitting o underfitting debemos evaluar el modelo con el set de prueba, y así podremos saber si predice correctamente.



Graf 5. Predict con el set prueba.

El modelo predijo con 100% las 112 fotos de las diferentes señales.

```
0.0010
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step - accuracy: 1.0000 - loss: 0.1221 - precision: 1.0000 - recall: 1.0000
Test loss and accuracy: [0.12214840948581696, 1.0, 1.0, 1.0]
```

Img 16. Evaluación del modelo con set de prueba.

Es muy probablemente que el modelo está overfitting, ya que la precisión del test fue de 100%, no obstante, para reafirmar o verificar que el modelo está overfitting los someteremos a dos pruebas, que más adelante detallamos.

VII. MODEL RED NEURONAL CONVOLUCIONAL CON MODELO PRE-ENTRENADO MOBILNET

Entrenamos otro modelo, utilizando un modelo pre-entrenado MobilNet. Código: programa

MobilNet es una arquitectura de redes neuronales convolucionales desarrolladas por Google, diseñadas especialmente para dispositivos móviles y otros entornos con recursos limitados.

Adaptamos esta arquitectura para nuestro objetivo de clasificación.

```
MobilNet=Sequential()
MobilNet.add(MobileNetV2(weights='imagenet',include_top=False,input_shape=(224,224,3)))
MobilNet.add(BatchNormalization())
MobilNet.add(Flatten())
MobilNet.add(Dropout(0.5))
MobilNet.add(Dense(256, activation='relu'))
MobilNet.add(Dropout(0.5))
MobilNet.add(Dense(units=train_generator.num_classes,activation='softmax',kernel_regularizer=l2(0.001)))
```

Img 17. Distribución de fotos por label del folder de prueba.

MobileNetV2:

- Se utiliza la arquitectura MobileNetV2 pre-entrenada en ImageNet, excluyendo la capa superior.
- Tamaño de entrada: 224x224 con 3 canales de color (RGB).

Batch Normalization:

- Normaliza la salida de MobileNetV2 para estabilizar y acelerar el entrenamiento.

Flatten:

- Convierte las características extraídas en un vector plano para pasarlas a las capas densas.

Dropout (0.5):

- Apaga aleatoriamente el 50% de las unidades para prevenir el sobreajuste.

Dense (256 unidades, ReLU):

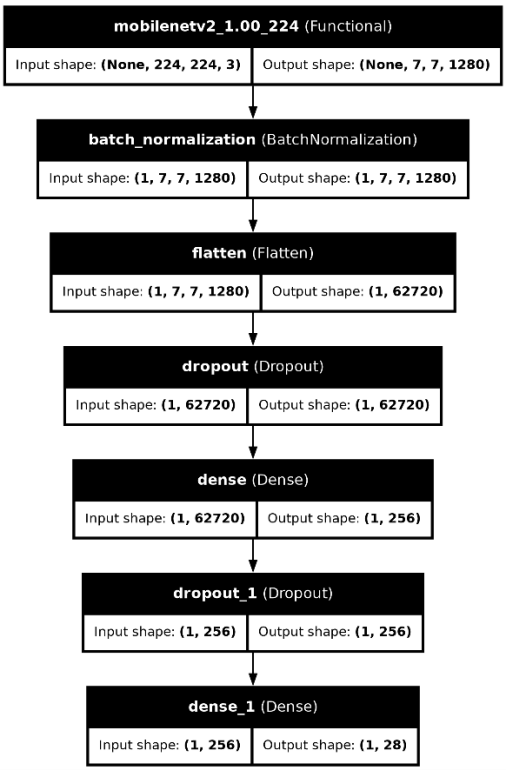
- Capa totalmente conectada con 256 neuronas y activación ReLU para aprender patrones complejos en las características extraídas.

Dropout (0.5):

- Dropout con una tasa del 50% para reducir el riesgo de sobreajuste.

Dense (Capa de salida):

- Capa densa final con una unidad por cada clase, activación softmax para la clasificación y regularización L2 de 0.01 para reducir el sobreajuste.



Diag 1. Diagrama de la arquitectura del modelo.

Ocuparemos los siguientes optimizadores

```
MobilNet.compile(
    optimizer=Adam(),
    loss='categorical_crossentropy',
    metrics=[
        Recall,
        Precision,
        'accuracy'
    ]
)
```

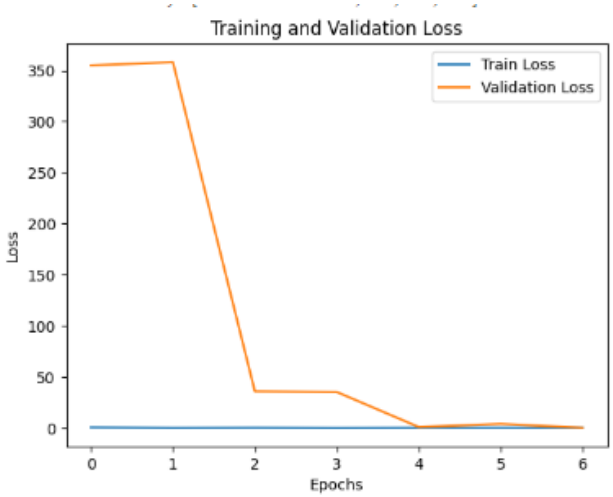
Img 18. Optimizadores de nuestro modelo MobilNet.

Utilizamos Adam con una tasa de aprendizaje por default. En la función de pérdida, usamos crossentropy, la cual es adecuado para problema de clasificación múltiple, mencionado anteriormente. Además, vamos a ocupar las métricas recall y precisión, con el fin de identificar correctamente las clases y evitar falsos positivos, también ocupamos acurracy para medir la porción de predicciones correctas.

```
# Fit the model
history2 = MobilNet.fit(
    train_generator,
    epochs=7,
    validation_data=validation_generator,
    steps_per_epoch=train_generator.samples// train_generator.batch_size,
    validation_steps=validation_generator.samples// validation_generator.batch_size,
    batch_size=32

)

# Evaluate the model
results2 = MobilNet.evaluate(test_generator)
print("Test loss and accuracy:", results2)
```

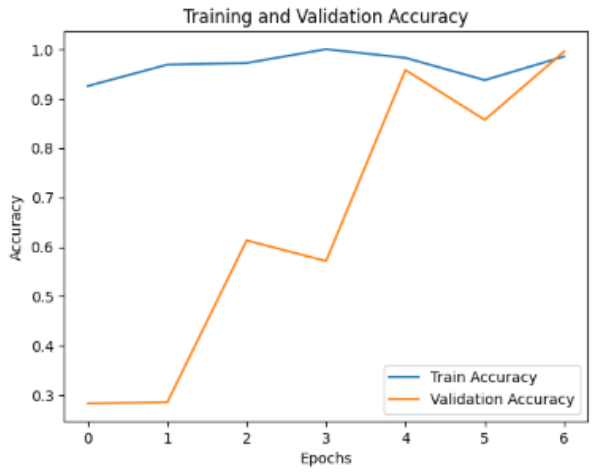
Img 19. Señalizar modelo MobilNet para el entrenamiento.

El modelo tardó entrenarse 3hrs en 7 épocas.



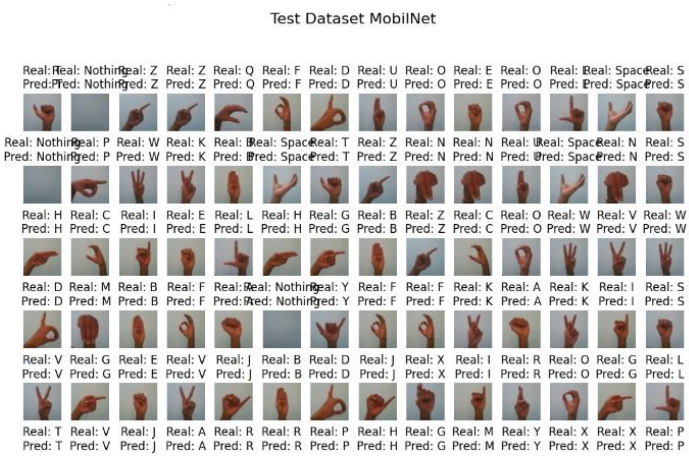
Graf 6. Gráfica de pérdida del entrenamiento vs validación.

El modelo presenta una reducción de la perdida significativa después de la época 2, puede ser indicios de que esté aprendiendo o se esté sobre aprendiendo los datos para generalizar, por lo que, se puede decir que está fitting o overfitting



Graf 7. Gráfica de precisión del entrenamiento vs validación.

El modelo aparenta estar aprendiendo debido el modelo aprende al paso de las épocas, logrando una precisión de validación que casi coincide con la precisión de entrenamiento final.



Graf 8. Predicciones del modelo MobilNet.

El modelo predijo 100% de las 112 imágenes de las diferentes clases.

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 6s 6s/step - accuracy: 1.0000 - loss: 0.0113 - precision: 1.0000 - recall: 1.0000
Test loss and accuracy: [0.011313770897686481, 1.0, 1.0, 1.0]
```

Img 20. Evaluando el modelo con el set de prueba.

Como lo mencionamos antes, es muy probablemente que el modelo se esté sobre ajustando, para reafirmar o verificar que el modelo haya aprendido, los someteremos a dos pruebas, que más adelante detallamos.

IX. MODEL RED NEURONAL CONVOLUCIONAL CON MODELO PRE-ENTRENADO MOBILNET MEJORA

Cabe recalcar que este modelo se construyó con la finalidad de mejorar la robustez del modelo MobilNet entrenado previamente para que sea capaz clasificar diferentes gestos en distintas condiciones del entorno como de las manos.

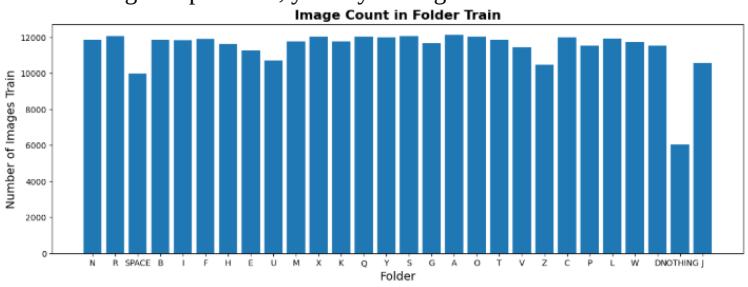
Dataset

En este tercer modelo se entrenó con un dataset combinado para aumenta la variedad de datos, que los modelos puedan generalizar los patrones en diferentes condiciones.

El dataset está constituido por los siguientes dataset de kaggle:

- American Sign Language
- American Sign Language 2
- ASL (American Sign Language) Alphabet Dataset
- American Sign Language Dataset

Conservamos los mismos label predefinidos en párrafos anteriores. Analizamos la distribución del dataset para ver la cantidad de imágenes por label, y si hay homogeneidad.



Graf 9. Gráfica que muestra la distribución del dataset por label.

El dataset cuenta con un total 319,545 imágenes, y cada label tiene más de 6,000 imágenes. Después de lo anterior, al dataset le aplicaremos argumentación de datos con el propósito de agregar más variedad a los datos, y evitar que el modelo se sobre aprenda los patrones de las imágenes el modelo.

Argumentación de datos

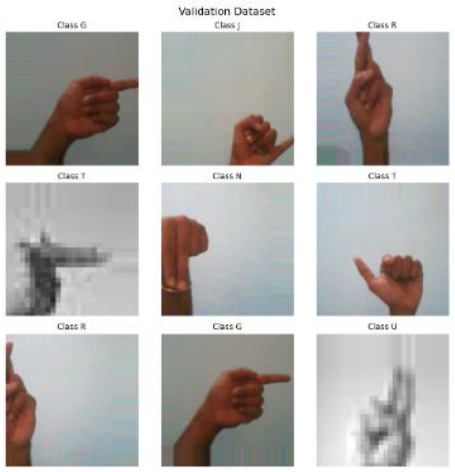
Debo de resaltar, que, al set de entrenamiento y validación, aplicaremos normalización (0 a 1), cortes aleatorios de hasta 20% de la imagen, zoom de hasta 10% de aumento y de reducción, cambio de posición a lo ancho y alto en un 20%, rango de iluminación de 30% a 130%, y reserva el 10% de los datos para su validación.

```
train = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=[0.9,1.1],
    width_shift_range=0.2,
    height_shift_range=0.2,
    brightness_range=[0.7, 1.3],
    validation_split=0.1


)
```

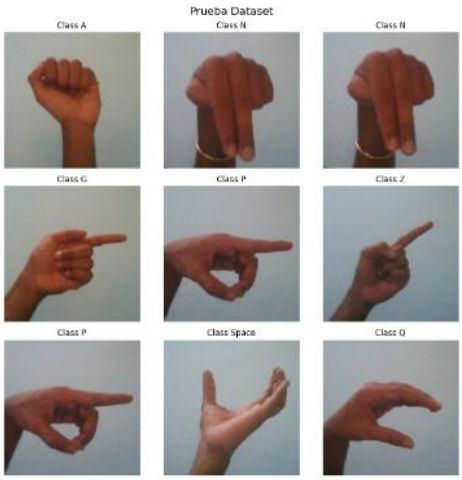
Img 21. Código con ImageDataGenerator para argumentación de datos.

Muestra de las modificaciones por set

Img 22. Código con ImageDataGenerator para argumentación de datos.



Img 23. Código con ImageDataGenerator para argumentación de datos.



Img 24. Código con ImageDataGenerator para argumentación de datos.

El dataset lo dividimos en 10% validación y 90% de entrenamiento, y evaluaremos el modelo con las 112 imágenes de prueba con las que evaluamos los dos modelos anteriores, para ello, solo normalizaremos. Además, para los tres sets, se distribuirán en batches de 32 imágenes, y un reajuste de tamaño de las imágenes de 224px * 224px, y revolvemos las imágenes del set de entrenamiento y validación

```
#### train generator
train_generator = train.flow_from_directory(
    train_path,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='training',
    shuffle=True
)
#### validation generator
val_generator = train.flow_from_directory(
    train_path,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
    subset='validation',
    shuffle=True
)

#### test generator
test_generator2=test_datagen.flow_from_directory
    test_path,
    target_size=(224,224),
    batch_size=112,
    class_mode='categorical'
)
```

Img 25. División de los sets de entrenamiento, prueba y validación.

Modelo MobilNet Mejorado

Nuevamente cabe mencionar que el objetivo de estas mejoras es tener un mejor rendimiento del modelo en términos de generalización y optimización.

1. Cambio de la capa Flatten por una capa de promedio global de pooling 2D.

   Cuando estamos trabajando con resoluciones grandes, esto hace que aumente la cantidad de datos, cuando se ocupa una capa de tipo flatten genera un vector unidimensional, generando aún más la cantidad de parámetros que condice a sobre ajustes y un aumento de complejidad innecesario. Sin embargo, al cambiar la capa flatten por global average pooling 2d, esto hace que, en lugar de aplanar la salida de las capas convolucionales, se calcule el promedio de cada mapa de características, y esto reduce el número de parámetros, que ayuda a la capacidad de generalizar.

2. Eliminación del Dropout de 50% antes de la capa densa.

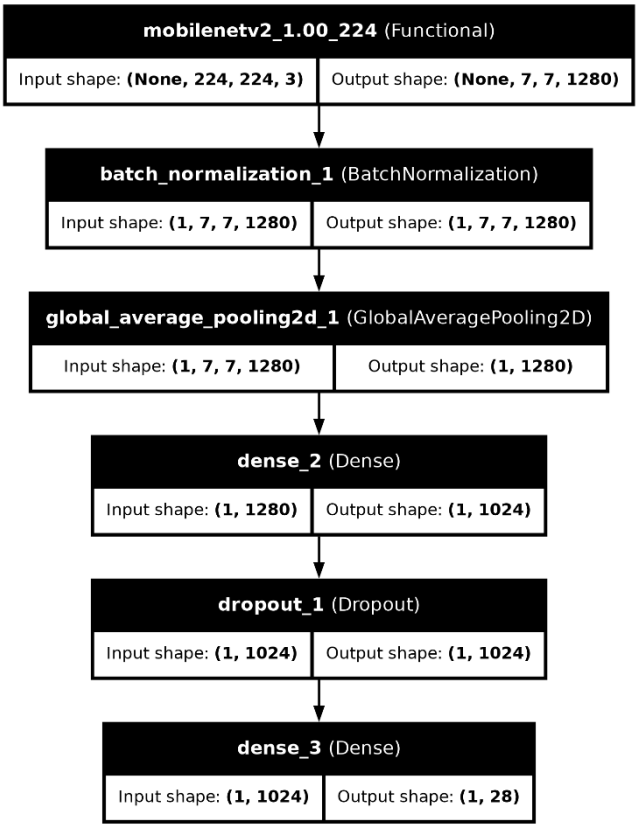
   Al agregar la capa Global Average Pooling antes de la capa densa, esto hace que el dropout se vuelva redudante, ya que con la capa anterior se reduce considerablemente los parámetros, y además el agregarlos disminuye la velocidad de entrenamiento.

3. Aumento de nueronas en la capa densa a 1024.

   El aumento de neuronas busca incrementar la capacidad del modelo para aprender representaciones más complejas, además útil está técnica debido la gran cantidad de datos.

4. Cambio en el tipo de regularización en la última capa a L1 con valor de 0.1

   L2 consta de penalizar los pesos grandes al sumar su cuadrado al término de pérdida, no obstante, con L1, penaliza la magnitud absoluta de los pesos, es decir que algunos pesos se vuelvan exactamente cero, con el fin de reducir la complejidad.



Diag 2. Arquitectura del modelo MobilNet Mejorado.

Parámetros

1. Utilizamos el optimizador Adam con un leraning rate 0.0001.

2. Función de perdida se utiliza crossentropy

3. Se usó la métrica de precisión o accuracy para evaluar el rendimiento

```
# Compilación del modelo
Net3.compile(
    optimizer=Adam(0.00001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

Img 26. Señalización de parámetros para evaluar el modelo.

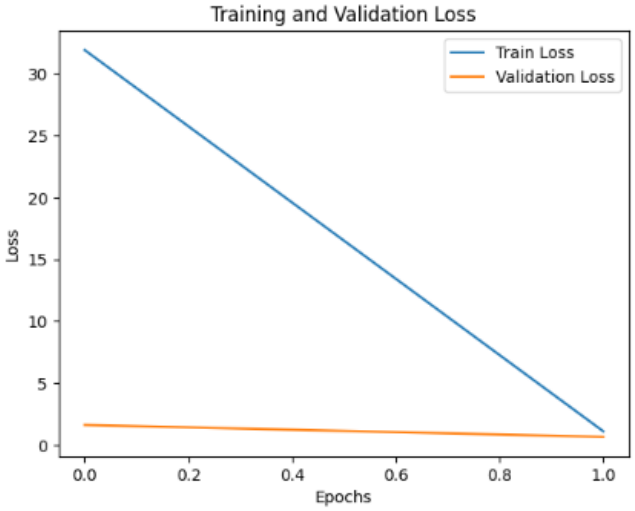
Entrenamiento

```
# Entrenamiento del modelo
history3 = Net3.fit(
    train_generator,
    validation_data=val_generator,
    epochs=2,
    #steps_per_epoch=(train_generator.samples // train_generator.batch_size),
    #validation_steps=(val_generator.samples // val_generator.batch_size),
    verbose=1,
    callbacks=[early_stop, lr_reduction,checkpoint_callback]
)
```

Img 27. Señalización de parámetros para entrenar el modelo.

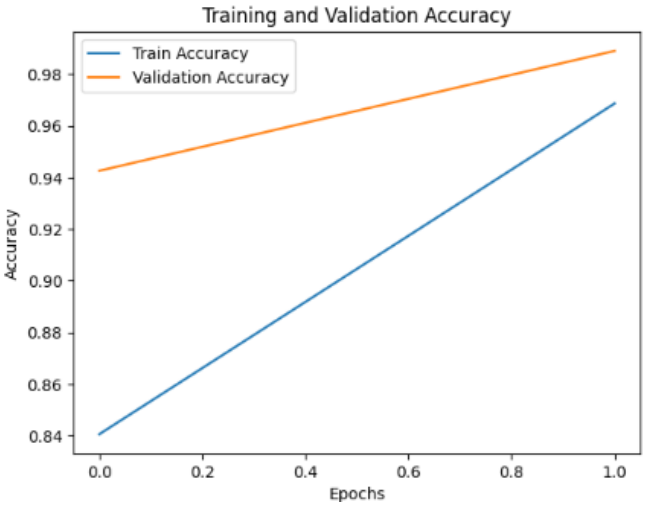
El modelo Net3 se entrena con los siguientes elementos:

1. **Datos de entrenamiento y validación**:
    - Train_generator: Generador de datos para el entrenamiento.
    - Validation_data=val_generator: Generador de datos para la validación.
2. **Número de épocas**:
    - Epochs=2: El modelo entrenará durante 2 épocas completas.
3. **Verbose**:
    - Verbose=1: Muestra el progreso del entrenamiento en la consola.
4. **Callbacks**:
    - **Early_stop**: Detiene el entrenamiento si no hay mejora en la pérdida de validación durante un número determinado de épocas.
    - **Lr_reduction**: Reduce la tasa de aprendizaje si la pérdida de validación no mejora.
    - **Checkpoint_callback**: (presumiblemente) guarda el modelo en checkpoints para poder recuperarlo después.
5. **Tiempo**
    - El modelo se tardó 3 horas para corres 2 épocas.



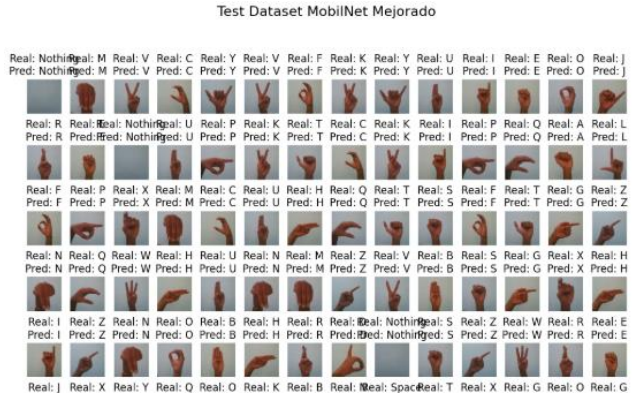
Graf 10. Gráfica que compara la perdida de validación con respecto entrenamiento.

A simple vista el modelo está sobre entrando extremo, y esto se debe a que asignamos el solo 10% para validación, y esto nos indica que posiblemente el conjunto de validación es fácil y que los datos están desbalanceados.



Graf 11. Gráfica que compara la precisión de validación con respecto entrenamiento

Nuevamente a simple vista se observa que el modelo está sobre entrando extremo.



Graf 12. Predicciones del modelo MobilNet Mejorado

El modelo predijo con 100% de asertividad todos los signos de la mano.

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 5s 5s/step - accuracy: 1.0000 - loss: 0.5908
Test loss and accuracy: [0.5907997488975525, 1.0]
```

Img 28. Evalaución del set de prueba con la métrica de accuracy y los para el modelo MobilNet Mejorado.

Lo interesante de este modelo, fue que su perdida fue de 0.59, algo que en los otros modelos fue muy baja, y comúnmente la perdida indica si está sobre ajustado o no.

Someteremos a los tres modelos a dos pruebas, para evaluar su precisión con diferentes gestos de manos y condiciones de la foto.

VIII. CNN VS MOBILNET VS MOBILNET MEJORADO

La mejor manera para comparar los modelos es probándolos con diferentes imágenes de diferentes dataset o simples imágenes del internet. Por ende, sometimos a dos pruebas los modelos para probar su rendimiento, y poder demostrar empíricamente las mejoras de los modelos.

En estas pruebas utilizaremos las métricas de precisión y F1-score.

Acurracy o precisión

$$Accuracy = \frac{Número\ de\ predciciones\ correctas}{Número\ total\ de\ predicciones}$$

Utilizaremos esta métrica para proporcionar una visión del rendimiento del modelo considerando todas las clases y algunas muestras.

F1-score

$$F1_{score} = 2 * (\frac{Precisión * Recall}{Precisión + Recall})$$

Precisión – Porcentaje de predicciones correctas sobre las que el modelo predijo como positivas.

$$Precisión = (\frac{Verdaderos\ positivos}{Verdaderos\ positivos + Falsos\ positivos})$$

Recall – Porcentaje de ejemplos de la clase positiva que fueron correctamente identificada.

$$Recall = \frac{Verdaderos\ positivos}{Verdadero\ positivos + Falsos\ positivos}$$

Utilizaremos F1_score, esta métrica que nos permite evaluar cómo el modelo gestiona las clases difíciles o desbalanceadas debido a que no es homogénea nuestro dataset.

Prueba 1

Esta primera prueba consiste en seleccionar un dataset de kaggle y evaluarlo con su set de prueba para ver su rendimiento con respecto a otro set con imágenes predefinidas y mejor resolución para probar el modelo.

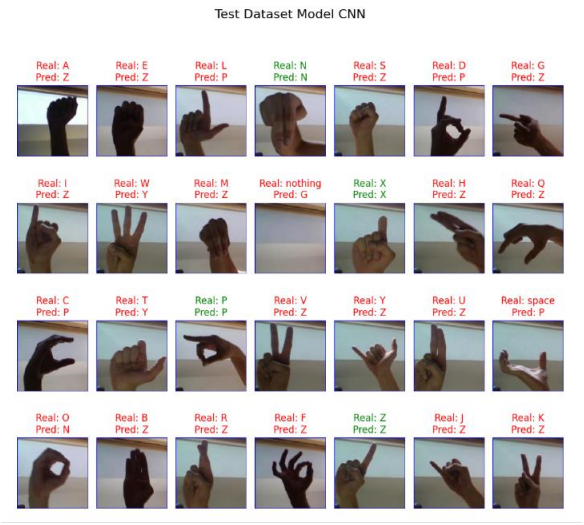
En esta prueba ocuparemos el set de prueba de este dataset originario de kaggle (Dataset), la cual evaluará cada modelo.

El objetivo de esta prueba es visualizar si los modelos están overfitting, ya que cada modelo presentó en su test de prueba un 100% de precisión con respecto al dataset original con la que se entrenó los modelos anteriormente.

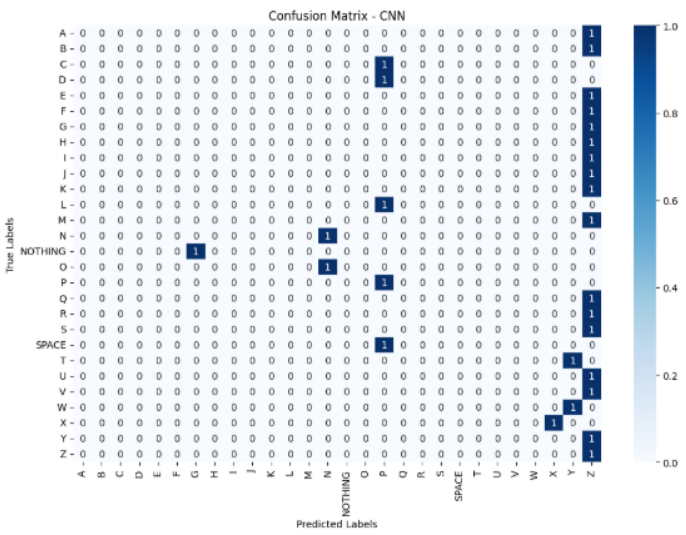
Entonces, la prueba consiste en tomar un dataset distinto al que se entrenó, con el fin de recolectar los samples de test de cada label de nuestros modelos , por lo que, tendremos 28 fotos, y cada foto es un label (A, B,C, D … Z,

NOTHING y SPACE), una vez de haber recolectado nuestro set de fotos, haremos la predicción de cada label, y medir la precisión de los modelos para clasificar el abecedario de lenguaje de señas, en otras palabras, es hacer nuevamente el mismo test de prueba que se le realizó a cada modelo, no obstante, con otro dataset.

Modelo CNN



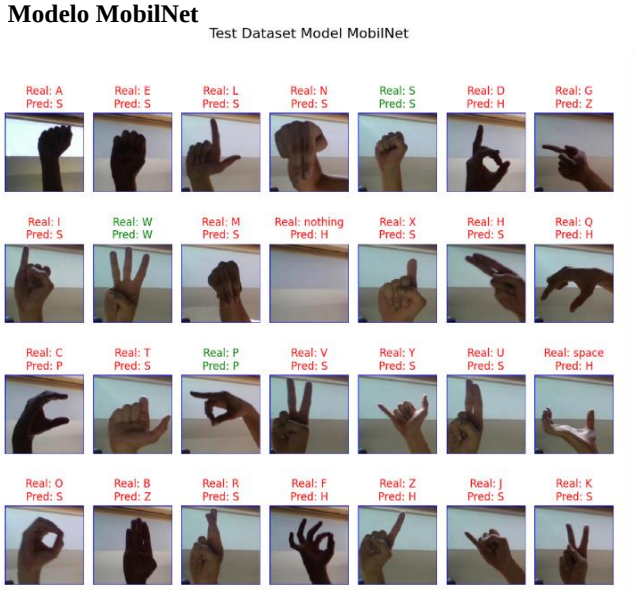
Graf 13. Predicciones del modelo CNN con un set de prueba distinto.



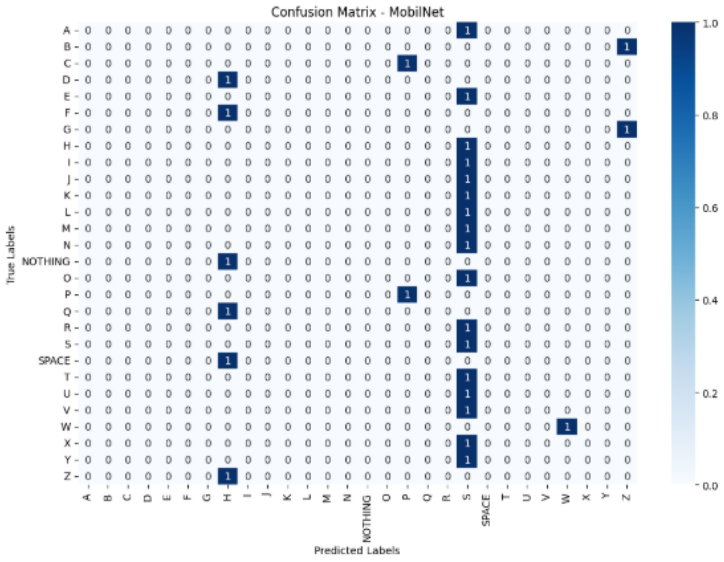
Graf 14. Matriz de confusión modelo CNN con un set de prueba distinto.

El modelo CNN al evaluar el rendimiento del modelo, se observó que fue capaz de identificar correctamente 4 de las 28 clases, lo que resulta en una precisión de aproximadamente 14%.

Modelo MobilNet



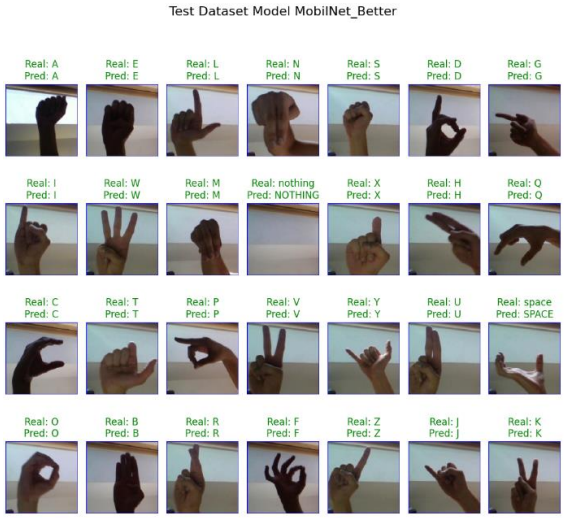
Graf 15. Predicciones del modelo MobilNet con un set de prueba distinto.



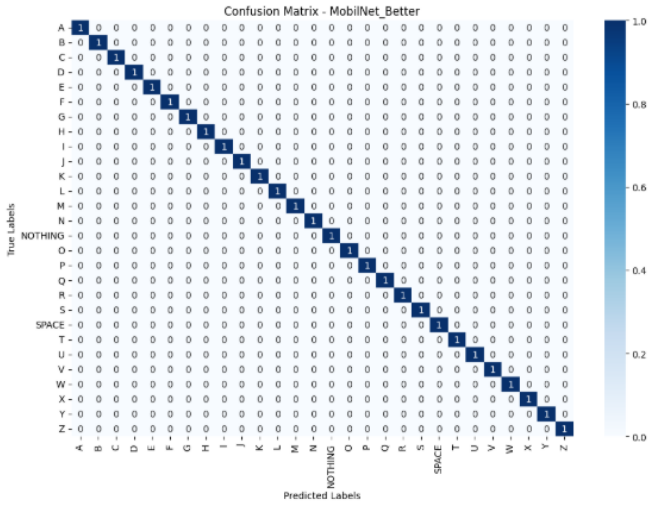
Graf 16. Matriz de confusión modelo MobilNet con un set de prueba distinto.

El modelo MobileNet al evaluar el rendimiento del modelo, se observó que fue capaz de identificar correctamente 3 de las 28 clases, lo que resulta en una precisión de aproximadamente 11%.

Modelo MobilNet Mejorado



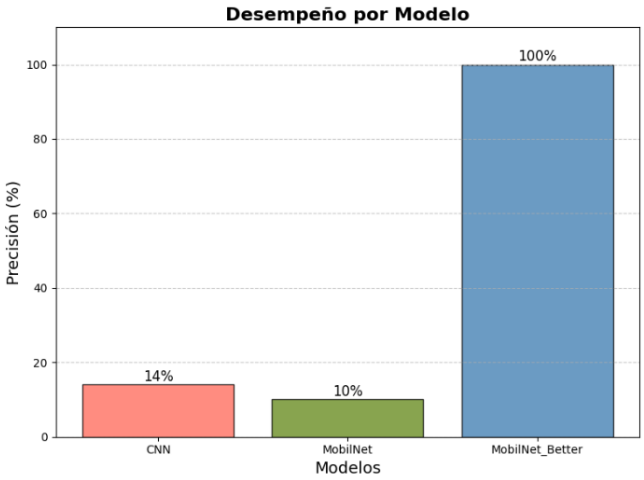
Graf 17. Predicciones del modelo MobilNet Mejorado con un set de prueba distinto.



Graf 18. Matriz de confusión modelo MobilNet Mejorado con un set de prueba distinto

El modelo MobileNet Mejorado al evaluar el rendimiento del modelo, se observó que fue capaz de identificar correctamente 28 de las 28 clases, lo que resulta en una precisión de aproximadamente 100%.

Evaluación de los modelos Prueba 1



Graf 19. Gráfica de comparación de los modelos CNN, MobilNet y MobilNEt Mejorado con otro set de prueba.

```
Summary of F1-Scores:
CNN: F1-Score = 0.0754
MobilNet: F1-Score = 0.0635
MobilNet_Better: F1-Score = 1.0000
```

Graf 20. Resumen del score de F1 por modelo.

Prueba 2

Esta segunda prueba consiste en seleccionar diferentes imágenes de internet, y generar un dataset para evaluar el rendimiento de los modelos. Dataset

El propósito de esta prueba es visualizar si los modelos están generalizando bien los patrones de las señales, con el fin de poder identificar distintas señales en diferentes ambientes.

Modelo CNN

Graf 21. Predicciones del modelo CNN con un set con datos random de internet.

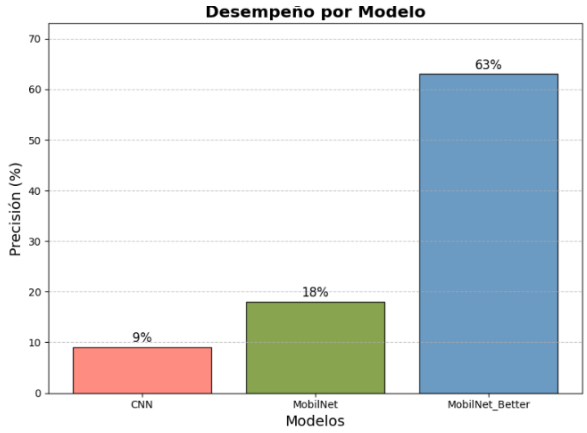
Modelo MobilNet

Graf 22. Predicciones del modelo MobilNet con un set con datos random de internet.

Modelo MobilNet Mejorada

Graf 23. Predicciones del modelo MobilNet Mejorado con un set con datos random de internet.

Evaluación de los modelos Prueba 2



Graf 24. Gráfica de comparación de los modelos CNN, MobilNet y MobilNEt Mejorado con samples de internet.

```
Summary of F1-Scores:
CNN: F1-Score = 0.1212
MobilNet: F1-Score = 0.1182
MobilNet_Better: F1-Score = 0.6455
```

Graf 25. Matriz de confusión modelo MobilNet Mejorado con un set de prueba distinto

X. RESULTADOS

	CNN	MOBILNET	MOBILNET MEJORADO
Set de Test			
Accuracy	100%	100%	100%
Loss	12%	1%	59%
Prueba 1			
Accuracy	14%	10%	100%
F1	7%	6%	100%
Prueba 2			
Accuracy	9%	18%	63%
F1	12%	11%	54%

Comparamos el rendimiento de tres modelos (CNN, MobileNet y MobileNet Mejorado) en tres escenarios. Aunque todos logran 100% de precisión en el conjunto de prueba, MobileNet destaca por su baja pérdida (1%), indicando que podría ser más confiable en sus predicciones, pero resultó ser el peor para hacer predicciones, mientras que MobileNet Mejorado presenta una pérdida alta (59%), probablemente se deba a la variedad de los datos, ya que construimos un dataset con múltiples dataset y utilizamos argumentación de datos para aumentar aún más la volatibilidad de los datos.

En las pruebas desafiantes:

- **Prueba 1:** MobileNet Mejorado supera ampliamente a los otros modelos con 100% de precisión y F1-score, frente a resultados muy bajos de la CNN (14% precisión) y MobileNet (10% precisión).
- **Prueba 2:** MobileNet Mejorado sigue liderando con 63% de precisión y 54% F1-score, aunque aún tiene margen de mejora.

MobileNet Mejorado es el más robusto para condiciones complejas, pero necesita ajustes para reducir la pérdida en el conjunto de prueba. La CNN y MobileNet básico tienen un desempeño limitado en escenarios desafiantes.

XI. CONCLUSIONES

Al finalizar este estudio, se concluye que los modelos evaluados de redes neuronales convolucionales, específicamente el modelo CNN básico, MobileNet y MobileNet Mejorado, muestran resultados prometedores, destacando el último como el más robusto y efectivo para clasificar el lenguaje de señas en condiciones variadas y complejas. Sin embargo, existe margen de mejora, especialmente en lo que respecta al último dataset utilizado. Se introdujeron imágenes de 28px * 28px en escala de grises, lo que, al ser redimensionadas a 224px * 224px, provocó que los gestos no se apreciaran con la suficiente claridad.

A pesar de esto, los modelos lograron predecir correctamente con un 100% de precisión en los datos de prueba. En la primera prueba con un set de datos externo, el modelo MobileNet Mejorado mantuvo un 100% de precisión, mientras que los modelos CNN y MobileNet alcanzaron solo un 14% y un 10%, respectivamente. En la segunda prueba, realizada con imágenes aleatorias de internet, el modelo MobileNet Mejorado alcanzó un 63% de precisión. No obstante, este desempeño podría haber sido aún mejor si no fuera por las similitudes visuales entre algunas señales, como la “R” y la “U”, que dificultaron la clasificación, aunque el modelo logró identificar correctamente las señales “E” y “A”, que también son muy parecidas, diferenciándose únicamente en la posición del pulgar.

En resumen, MobileNet Mejorado sobresale por su capacidad para generalizar patrones en diversas condiciones, evitando problemas de sobreajuste. Sin embargo, es evidente que hay oportunidades de mejora, particularmente en la parte de los datos. Este estudio resalta la importancia de las nuevas tecnologías y su potencial para ser implementadas en aplicaciones reales de reconocimiento de lenguaje de señas, contribuyendo así a una mayor inclusión social y

mejorando la comunicación entre los distintos sectores de la sociedad.

REFERENCIAS

Artificial Intelligence: A Modern Approach, 4th Global ed. (2022). Berkeley.edu. https://aima.cs.berkeley.edu/global-index.html

COCO Dataset Structure | Understanding Bounding Box Annotations for Object Detection
By Neuralception Container: YouTube Year: 2022 URL: https://www.youtube.com/watch?v=TLvdlDgZ3G0

GeeksforGeeks. (2017, August 21). *Introduction to Convolution Neural Network*. GeeksforGeeks. https://www.geeksforgeeks.org/introduction-convolution-neural-network/

Javier. (2024, May 7). *Hiperparametros de una red neuronal*. Javierheras.website; Inteligencia Artificial. https://blog.javierheras.website/hiperparametros-de-una-red-neuronal/

Vive UNIR. (2023, February 17). *¿Qué es el algoritmo backpropagation para el entrenamiento de redes neuronales?* UNIR. https://www.unir.net/revista/ingenieria/backpropagation/#:~:text=Backpropagation%20(o%20propagaci%C3%B3n%20hacia%20atr%C3%A1s,las%20redes%20de%20neuronas%20biol%C3%B3gicas.

https://www.youtube.com/watch?v=Y_WAXe4LprY&list=PLWzLQn_hxe6YL-9JiKwIZqIuB8jliA-Fs