Descargar el dataset desde kaggle

Para poder descargar el dataset usado en el modelo ejecuta el siguiente comando, ten en cuenta que estamos usando kaggle para esto y debes instalarlo y configurar la apiKey.

`pip install kaggle`

`kaggle datasets download -d grassknoted/asl-alphabet`

`kaggle datasets download datamunge/sign-language-mnist`

Mover Imagenes de test a train para hacer la división personalizada, asi cumplir con un 80 train / 20 test

```
import os
import shutil
import random
import math

def move_test_images():
    """
    Moves test images to their corresponding letter folder in the
train directory.
    """
    source_dir =
'../data/raw/asl-alphabet/asl_alphabet_test/asl_alphabet_test'
    train_dir =
'../data/raw/asl-alphabet/asl_alphabet_train/asl_alphabet_train'

    if not os.path.exists(source_dir):
        print(f"Error: Source directory not found at {source_dir}")
        return

    try:
        image_files = [f for f in os.listdir(source_dir) if
f.endswith('.jpg')]
    except FileNotFoundError:
        print(f"Error: Could not list files in {source_dir}. It might
not be a directory.")
        return


    for filename in image_files:
        letter = filename.split('_')[0]

        destination_folder = os.path.join(train_dir, letter)

        source_path = os.path.join(source_dir, filename)
        destination_path = os.path.join(destination_folder, filename)
```

```
        try:
            shutil.move(source_path, destination_path)
            print(f"Moved {filename} to {destination_folder}")
        except FileNotFoundError:
            print(f"Error: Could not find {filename} to move.")
        except Exception as e:
            print(f"An error occurred while moving {filename}: {e}")


move_test_images()
print("Script finished.")

def split_data(source_dir, processed_dir, split_ratio=0.8):
    """
    Splits the data from source_dir into training and testing sets
    and saves them in processed_dir.

    Args:
        source_dir (str): The path to the directory containing the raw
data,
                          with subdirectories for each class.
        processed_dir (str): The path to the directory where the
processed
                             (split) data will be saved.
        split_ratio (float): The ratio of training data to the total
data.
    """
    train_dir = os.path.join(processed_dir, 'train')
    test_dir = os.path.join(processed_dir, 'test')

    os.makedirs(train_dir, exist_ok=True)
    os.makedirs(test_dir, exist_ok=True)

    if not os.path.exists(source_dir):
        print(f"Error: Source directory not found at {source_dir}")
        return

    for letter_folder in os.listdir(source_dir):
        letter_path = os.path.join(source_dir, letter_folder)
        if os.path.isdir(letter_path):
            train_letter_dir = os.path.join(train_dir, letter_folder)
            test_letter_dir = os.path.join(test_dir, letter_folder)
            os.makedirs(train_letter_dir, exist_ok=True)
            os.makedirs(test_letter_dir, exist_ok=True)

            images = [f for f in os.listdir(letter_path) if
os.path.isfile(os.path.join(letter_path, f))]
            random.shuffle(images)
```

```
            split_point = math.ceil(len(images) * split_ratio)
            train_images = images[:split_point]
            test_images = images[split_point:]

            for image in train_images:
                source_image_path = os.path.join(letter_path, image)
                dest_image_path = os.path.join(train_letter_dir,
image)
                shutil.copyfile(source_image_path, dest_image_path)

            print(f"Copied {len(train_images)} images to
{train_letter_dir}")

            for image in test_images:
                source_image_path = os.path.join(letter_path, image)
                dest_image_path = os.path.join(test_letter_dir, image)
                shutil.copyfile(source_image_path, dest_image_path)

            print(f"Copied {len(test_images)} images to
{test_letter_dir}")



SOURCE_DATA_DIR =
'../data/raw/asl-alphabet/asl_alphabet_train/asl_alphabet_train'
PROCESSED_DATA_DIR = '../data/processed'

split_data(SOURCE_DATA_DIR, PROCESSED_DATA_DIR)
print("Data splitting finished.")
```

```
Copied 2401 images to ../data/processed/train/R
Copied 600 images to ../data/processed/test/R
Copied 2401 images to ../data/processed/train/U
Copied 600 images to ../data/processed/test/U
Copied 2401 images to ../data/processed/train/I
Copied 600 images to ../data/processed/test/I
Copied 2401 images to ../data/processed/train/N
Copied 600 images to ../data/processed/test/N
Copied 2401 images to ../data/processed/train/G
Copied 600 images to ../data/processed/test/G
Copied 2401 images to ../data/processed/train/Z
Copied 600 images to ../data/processed/test/Z
Copied 2401 images to ../data/processed/train/T
Copied 600 images to ../data/processed/test/T
Copied 2401 images to ../data/processed/train/S
Copied 600 images to ../data/processed/test/S
Copied 2401 images to ../data/processed/train/A
Copied 600 images to ../data/processed/test/A
Copied 2401 images to ../data/processed/train/F
Copied 600 images to ../data/processed/test/F
```

```
Copied 2401 images to ../data/processed/train/O
Copied 600 images to ../data/processed/test/O
Copied 2401 images to ../data/processed/train/H
Copied 600 images to ../data/processed/test/H
Copied 2400 images to ../data/processed/train/del
Copied 600 images to ../data/processed/test/del
Copied 2401 images to ../data/processed/train/nothing
Copied 600 images to ../data/processed/test/nothing
Copied 2401 images to ../data/processed/train/space
Copied 600 images to ../data/processed/test/space
Copied 2401 images to ../data/processed/train/M
Copied 600 images to ../data/processed/test/M
Copied 2401 images to ../data/processed/train/J
Copied 600 images to ../data/processed/test/J
Copied 2401 images to ../data/processed/train/C
Copied 600 images to ../data/processed/test/C
Copied 2401 images to ../data/processed/train/D
Copied 600 images to ../data/processed/test/D
Copied 2401 images to ../data/processed/train/V
Copied 600 images to ../data/processed/test/V
Copied 2401 images to ../data/processed/train/Q
Copied 600 images to ../data/processed/test/Q
Copied 2401 images to ../data/processed/train/X
Copied 600 images to ../data/processed/test/X
Copied 2401 images to ../data/processed/train/E
Copied 600 images to ../data/processed/test/E
Copied 2401 images to ../data/processed/train/B
Copied 600 images to ../data/processed/test/B
Copied 2401 images to ../data/processed/train/K
Copied 600 images to ../data/processed/test/K
Copied 2401 images to ../data/processed/train/L
Copied 600 images to ../data/processed/test/L
Copied 2401 images to ../data/processed/train/Y
Copied 600 images to ../data/processed/test/Y
Copied 2401 images to ../data/processed/train/P
Copied 600 images to ../data/processed/test/P
Copied 2401 images to ../data/processed/train/W
Copied 600 images to ../data/processed/test/W
Data splitting finished.
```

Normalización y preprocesamiento de datos

1. Normalizar
2. Convertir a escala de grises

```
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
```

```
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, InputLayer
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

TRAIN_DIR = '../data/processed/train'
TEST_DIR = '../data/processed/test'

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

test_datagen = ImageDataGenerator(rescale=1./255)

BATCH_SIZE = 32
EPOCHS = 30
IMG_HEIGHT = 200
IMG_WIDTH = 200

train_generator = train_datagen.flow_from_directory(
    TRAIN_DIR,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)

validation_generator = test_datagen.flow_from_directory(
    TEST_DIR,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)

print("Class indices:", train_generator.class_indices)

Found 69628 images belonging to 29 classes.
Found 17400 images belonging to 29 classes.
Class indices: {'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G':
6, 'H': 7, 'I': 8, 'J': 9, 'K': 10, 'L': 11, 'M': 12, 'N': 13, 'O':
14, 'P': 15, 'Q': 16, 'R': 17, 'S': 18, 'T': 19, 'U': 20, 'V': 21,
'W': 22, 'X': 23, 'Y': 24, 'Z': 25, 'del': 26, 'nothing': 27, 'space':
```

```
28}
Found 17400 images belonging to 29 classes.
Class indices: {'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G':
6, 'H': 7, 'I': 8, 'J': 9, 'K': 10, 'L': 11, 'M': 12, 'N': 13, 'O':
14, 'P': 15, 'Q': 16, 'R': 17, 'S': 18, 'T': 19, 'U': 20, 'V': 21,
'W': 22, 'X': 23, 'Y': 24, 'Z': 25, 'del': 26, 'nothing': 27, 'space':
28}
```

Arquitectura de la red neuronal

```
model = keras.Sequential(
    [
        layers.Input(shape=(120000,)),
        layers.Dense(512, activation="relu"),
        layers.Dense(29, activation="softmax"),
    ]
)

model.summary()
```

```
Model: "sequential"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━
┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━
┩
│ dense (Dense)                        │ (None, 512)                 │
61,440,512 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ dense_1 (Dense)                      │ (None, 29)                  │
14,877 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────
┘
```

```
 Total params: 61,455,389 (234.43 MB)

 Trainable params: 61,455,389 (234.43 MB)

 Non-trainable params: 0 (0.00 B)
```

```
num_classes = len(train_generator.class_indices)

model = Sequential([
    InputLayer(shape=(IMG_HEIGHT, IMG_WIDTH, 3)),

    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
```

```
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

    Flatten(),

    Dense(512, activation='relu'),
    Dense(num_classes, activation='softmax')
])

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_2"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━
┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━
┩
│ conv2d_3 (Conv2D)                    │ (None, 198, 198, 32)        │
896 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ max_pooling2d_3 (MaxPooling2D)       │ (None, 99, 99, 32)          │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ conv2d_4 (Conv2D)                    │ (None, 97, 97, 64)          │
18,496 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ max_pooling2d_4 (MaxPooling2D)       │ (None, 48, 48, 64)          │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ conv2d_5 (Conv2D)                    │ (None, 46, 46, 128)         │
73,856 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ max_pooling2d_5 (MaxPooling2D)       │ (None, 23, 23, 128)         │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
```

```
──────┤
│ flatten_1 (Flatten)                 │ (None, 67712)             │
0 │
├─────────────────────────────────────┼───────────────────────────┼──────────
──────┤
│ dense_4 (Dense)                     │ (None, 512)               │
34,669,056 │
├─────────────────────────────────────┼───────────────────────────┼──────────
──────┤
│ dense_5 (Dense)                     │ (None, 29)                │
14,877 │
└─────────────────────────────────────┴───────────────────────────┴──────────
──────┘

 Total params: 34,777,181 (132.66 MB)

 Trainable params: 34,777,181 (132.66 MB)

 Non-trainable params: 0 (0.00 B)
```

Entrenamiento y guardado del Modelo

```
!pip install scipy

FILE_PATH = '../models/best_asl_model.keras'
checkpoint = ModelCheckpoint(
    filepath=FILE_PATH,
    monitor='val_accuracy',
    save_best_only=True,
    verbose=1
)

early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True,
    verbose=1
)

history = model.fit(
    train_generator,
    epochs=EPOCHS,
    validation_data=validation_generator,
    callbacks=[checkpoint]
)
print("Trainning finished, best model saved at:", FILE_PATH)

Epoch 1/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 509ms/step - accuracy: 0.2305 -
loss: 2.6487
```

```
/Users/jepolancos/projects/hands-unmuted/.venv/lib/python3.11/site-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call
`super().__init__(**kwargs)` in its constructor. `**kwargs` can
include `workers`, `use_multiprocessing`, `max_queue_size`. Do not
pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()


Epoch 1: val_accuracy improved from -inf to 0.78787, saving model
to ../models/best_asl_model.keras
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1172s 538ms/step - accuracy: 0.2306 -
loss: 2.6484 - val_accuracy: 0.7879 - val_loss: 0.6132
Epoch 2/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 505ms/step - accuracy: 0.7045 -
loss: 0.8791
Epoch 2: val_accuracy improved from 0.78787 to 0.90603, saving model
to ../models/best_asl_model.keras
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1160s 533ms/step - accuracy: 0.7045 -
loss: 0.8790 - val_accuracy: 0.9060 - val_loss: 0.2632
Epoch 3/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 493ms/step - accuracy: 0.8164 -
loss: 0.5421
Epoch 3: val_accuracy improved from 0.90603 to 0.94598, saving model
to ../models/best_asl_model.keras
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1136s 522ms/step - accuracy: 0.8164 -
loss: 0.5421 - val_accuracy: 0.9460 - val_loss: 0.1558
Epoch 4/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 500ms/step - accuracy: 0.8597 -
loss: 0.4074
Epoch 4: val_accuracy did not improve from 0.94598
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1153s 530ms/step - accuracy: 0.8597 -
loss: 0.4074 - val_accuracy: 0.9043 - val_loss: 0.2686
Epoch 5/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 498ms/step - accuracy: 0.8802 -
loss: 0.3546
Epoch 5: val_accuracy improved from 0.94598 to 0.96655, saving model
to ../models/best_asl_model.keras
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1149s 528ms/step - accuracy: 0.8802 -
loss: 0.3546 - val_accuracy: 0.9666 - val_loss: 0.1001
Epoch 6/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 507ms/step - accuracy: 0.8970 -
loss: 0.3106
Epoch 6: val_accuracy improved from 0.96655 to 0.96920, saving model
to ../models/best_asl_model.keras
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1166s 536ms/step - accuracy: 0.8970 -
loss: 0.3106 - val_accuracy: 0.9692 - val_loss: 0.0867
Epoch 7/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 506ms/step - accuracy: 0.9088 -
loss: 0.2707
```

```
Epoch 7: val_accuracy improved from 0.96920 to 0.97414, saving model
to ../models/best_asl_model.keras
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1165s 535ms/step - accuracy: 0.9088 -
loss: 0.2707 - val_accuracy: 0.9741 - val_loss: 0.0765
Epoch 8/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 508ms/step - accuracy: 0.9165 -
loss: 0.2493
Epoch 8: val_accuracy improved from 0.97414 to 0.97782, saving model
to ../models/best_asl_model.keras
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1170s 537ms/step - accuracy: 0.9165 -
loss: 0.2493 - val_accuracy: 0.9778 - val_loss: 0.0664
Epoch 9/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 521ms/step - accuracy: 0.9232 -
loss: 0.2331
Epoch 9: val_accuracy improved from 0.97782 to 0.98385, saving model
to ../models/best_asl_model.keras
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1199s 551ms/step - accuracy: 0.9232 -
loss: 0.2331 - val_accuracy: 0.9839 - val_loss: 0.0484
Epoch 10/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 510ms/step - accuracy: 0.9315 -
loss: 0.2051
Epoch 10: val_accuracy improved from 0.98385 to 0.98833, saving model
to ../models/best_asl_model.keras
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1172s 538ms/step - accuracy: 0.9315 -
loss: 0.2051 - val_accuracy: 0.9883 - val_loss: 0.0364
Epoch 11/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 499ms/step - accuracy: 0.9336 -
loss: 0.2006
Epoch 11: val_accuracy did not improve from 0.98833
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1150s 528ms/step - accuracy: 0.9336 -
loss: 0.2006 - val_accuracy: 0.9811 - val_loss: 0.0555
Epoch 12/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 500ms/step - accuracy: 0.9387 -
loss: 0.1835
Epoch 12: val_accuracy did not improve from 0.98833
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1151s 529ms/step - accuracy: 0.9387 -
loss: 0.1835 - val_accuracy: 0.9876 - val_loss: 0.0388
Epoch 13/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 506ms/step - accuracy: 0.9415 -
loss: 0.1769
Epoch 13: val_accuracy did not improve from 0.98833
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1164s 535ms/step - accuracy: 0.9415 -
loss: 0.1769 - val_accuracy: 0.9840 - val_loss: 0.0441
Epoch 14/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 503ms/step - accuracy: 0.9448 -
loss: 0.1690
Epoch 14: val_accuracy did not improve from 0.98833
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1159s 533ms/step - accuracy: 0.9448 -
loss: 0.1690 - val_accuracy: 0.9824 - val_loss: 0.0494
```

```
Epoch 15/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 502ms/step - accuracy: 0.9473 -
loss: 0.1653
Epoch 15: val_accuracy improved from 0.98833 to 0.99103, saving model
to ../models/best_asl_model.keras
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1156s 531ms/step - accuracy: 0.9473 -
loss: 0.1653 - val_accuracy: 0.9910 - val_loss: 0.0265
Epoch 16/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 503ms/step - accuracy: 0.9476 -
loss: 0.1604
Epoch 16: val_accuracy did not improve from 0.99103
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1158s 532ms/step - accuracy: 0.9476 -
loss: 0.1604 - val_accuracy: 0.9853 - val_loss: 0.0481
Epoch 17/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 500ms/step - accuracy: 0.9489 -
loss: 0.1607
Epoch 17: val_accuracy did not improve from 0.99103
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1150s 529ms/step - accuracy: 0.9489 -
loss: 0.1607 - val_accuracy: 0.9753 - val_loss: 0.0732
Epoch 18/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 502ms/step - accuracy: 0.9516 -
loss: 0.1495
Epoch 18: val_accuracy improved from 0.99103 to 0.99195, saving model
to ../models/best_asl_model.keras
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1156s 531ms/step - accuracy: 0.9516 -
loss: 0.1495 - val_accuracy: 0.9920 - val_loss: 0.0244
Epoch 19/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 501ms/step - accuracy: 0.9516 -
loss: 0.1509
Epoch 19: val_accuracy did not improve from 0.99195
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1153s 530ms/step - accuracy: 0.9516 -
loss: 0.1509 - val_accuracy: 0.9863 - val_loss: 0.0395
Epoch 20/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 500ms/step - accuracy: 0.9541 -
loss: 0.1425
Epoch 20: val_accuracy did not improve from 0.99195
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1149s 528ms/step - accuracy: 0.9541 -
loss: 0.1425 - val_accuracy: 0.9916 - val_loss: 0.0249
Epoch 21/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 503ms/step - accuracy: 0.9575 -
loss: 0.1342
Epoch 21: val_accuracy did not improve from 0.99195
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1159s 532ms/step - accuracy: 0.9575 -
loss: 0.1342 - val_accuracy: 0.9896 - val_loss: 0.0309
Epoch 22/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 503ms/step - accuracy: 0.9555 -
loss: 0.1399
Epoch 22: val_accuracy did not improve from 0.99195
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1157s 532ms/step - accuracy: 0.9555 -
```

```
loss: 0.1399 - val_accuracy: 0.9914 - val_loss: 0.0290
Epoch 23/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 502ms/step - accuracy: 0.9597 -
loss: 0.1276
Epoch 23: val_accuracy did not improve from 0.99195
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1155s 531ms/step - accuracy: 0.9597 -
loss: 0.1276 - val_accuracy: 0.9910 - val_loss: 0.0287
Epoch 24/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 501ms/step - accuracy: 0.9594 -
loss: 0.1320
Epoch 24: val_accuracy did not improve from 0.99195
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1152s 529ms/step - accuracy: 0.9594 -
loss: 0.1320 - val_accuracy: 0.9906 - val_loss: 0.0270
Epoch 25/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 499ms/step - accuracy: 0.9593 -
loss: 0.1289
Epoch 25: val_accuracy did not improve from 0.99195
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1147s 527ms/step - accuracy: 0.9593 -
loss: 0.1289 - val_accuracy: 0.9885 - val_loss: 0.0322
Epoch 26/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 500ms/step - accuracy: 0.9609 -
loss: 0.1279
Epoch 26: val_accuracy improved from 0.99195 to 0.99557, saving model
to ../models/best_asl_model.keras
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1150s 528ms/step - accuracy: 0.9609 -
loss: 0.1279 - val_accuracy: 0.9956 - val_loss: 0.0125
Epoch 27/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 499ms/step - accuracy: 0.9579 -
loss: 0.1361
Epoch 27: val_accuracy did not improve from 0.99557
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1150s 528ms/step - accuracy: 0.9579 -
loss: 0.1361 - val_accuracy: 0.9668 - val_loss: 0.0990
Epoch 28/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 497ms/step - accuracy: 0.9615 -
loss: 0.1235
Epoch 28: val_accuracy did not improve from 0.99557
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1145s 526ms/step - accuracy: 0.9615 -
loss: 0.1235 - val_accuracy: 0.9906 - val_loss: 0.0281
Epoch 29/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 495ms/step - accuracy: 0.9619 -
loss: 0.1223
Epoch 29: val_accuracy did not improve from 0.99557
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1140s 524ms/step - accuracy: 0.9619 -
loss: 0.1223 - val_accuracy: 0.9949 - val_loss: 0.0161
Epoch 30/30
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 0s 494ms/step - accuracy: 0.9635 -
loss: 0.1174
Epoch 30: val_accuracy did not improve from 0.99557
2176/2176 ━━━━━━━━━━━━━━━━━━━━ 1138s 523ms/step - accuracy: 0.9635 -
```

```
loss: 0.1174 - val_accuracy: 0.9922 - val_loss: 0.0244
Trainning finished, best model saved at:
../models/best_asl_model.keras
```

Evaluación del modelo

```
loaded_model = keras.models.load_model(FILE_PATH)
test_loss, test_accuracy = loaded_model.evaluate(validation_generator)
print(f"Test accuracy: {test_accuracy:.4f}")
```

```
544/544 ━━━━━━━━━━━━━━━━━━━━ 64s 117ms/step - accuracy: 0.9954 - loss:
0.0132
Test accuracy: 0.9956
544/544 ━━━━━━━━━━━━━━━━━━━━ 64s 117ms/step - accuracy: 0.9954 - loss:
0.0132
Test accuracy: 0.9956
```

Función de predicción del modelo

```
try:
    model = keras.models.load_model(FILE_PATH)
    print("Model loaded successfully.")
except Exception as e:
    print(f"Error loading model: {e}")
    exit()


def predict_image(model, img_path, class_indices):
    """Carga una imagen, la preprocesa y predice su clase."""

    img = image.load_img(img_path, target_size=(200, 200))


    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)


    img_array /= 255.0


    predictions = model.predict(img_array)


    predicted_class_index = np.argmax(predictions[0])
    confidence = np.max(predictions[0])
```

```
    index_to_class = {v: k for k, v in class_indices.items()}

    predicted_class_name = index_to_class[predicted_class_index]

    print(f"Índice de clase predicho: {predicted_class_index},
Confianza: {confidence:.4f}")
    print(f"Nombre de clase predicho: {predicted_class_name}")

    return predicted_class_name

try:
    image_path = '../data/test_images/2_test.jpg'
    predicted_class = predict_image(model, image_path,
train_generator.class_indices)
    print(f"\nResultado final para {image_path}: {predicted_class}")
except NameError:
    print("Asegúrate de haber ejecutado la celda que define
'train_generator' primero.")
except Exception as e:
    print(f"Ocurrió un error durante la predicción: {e}")


Model loaded successfully.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
Índice de clase predicho: 9, Confianza: 0.9999
Nombre de clase predicho: J

Resultado final para ../data/test_images/2_test.jpg: J
Índice de clase predicho: 9, Confianza: 0.9999
Nombre de clase predicho: J

Resultado final para ../data/test_images/2_test.jpg: J
```