AMERICAN SIGN LANGUAGE CLASSIFIER:

ABSTRACT:

The project relates to ASL image-based recognition using Convolutional Neural Networks. Being one of the most widely used sign languages, especially by the deaf and hard-of-hearing communities, automatic recognition of ASL is, therefore, important in bridging the communication gap between common people. The project benefited from a labelled dataset of images for static hand gestures representing alphabets from ASL. After preprocessing, the images were augmented such that the model could generalize well and not overfit on the dataset. A deep-learning CNN-based model was designed, trained, and evaluated with respect to key performance metrics for the task of American Sign Language recognition. Hence, one would want a robust and scalable system that can perform image recognition of American Sign Language signs with good accuracy for real-time applications. Potential uses for this system can include educational tools, assistive technologies, and accessibility applications. The resultant model showed very promising results, thus attesting that CNN is an effective solution for gesture recognition tasks.

Introduction:

ASL is a visually oriented language mainly serving the deaf and hard-of-hearing community. It consists of a series of hand gestures and movements that represent words and letters. In as much as ASL is pertinent in communication, its use poses a multitude of challenges when trying to facilitate interaction between those who perform sign language and those who can't. Thus technological advancements forged in computer vision and artificial intelligence provide just such an opportunity to close the present gap.

This system creates the potential for designing a machine-learning model that recognizes static ASL alphabets in images by employing a convolutional neural network architecture. The recognized hand gestures have been compiled into a dataset to train the model so that it can learn to associate picture patterns with specific ASL letters. It simultaneously serves as an academic platform to implement deep learning techniques in concrete problems and become a cornerstone for building inclusive communication platforms. Evaluation of the model based on efficiency, scalability, and accuracy will give an insight into its feasibility for deployment in assistive applications.

Dataset Description:

- This dataset contains a total of 3,132 images across 29 classes, including the 26 English letters (A–Z plus 'del', 'space' and 'nothing), with exactly 108 RGB images in each class, making it a balanced dataset for classification tasks.
- All images are of the same size and show static hand poses representing the associated ASL signs for that class. The images are organized by class folders, allowing easy loading in Tensorflow's ImageDataGenerator, thus applying automatic labelling and streamlining image preprocessing workflows.
- Due to the limited size, the dataset is well suited for small to midscale deep learning experiments and all image processing can be done on a standard CPU without the need for GPU acceleration. However, data augmentation during the image data processing pipelines has been used to improve model performance and generalization including example horizontal flipping, zooming, brightness changes, and small rotations to provide variability for gesture appearance in the real world.
- The dataset may be small, but we think it is an effective way to show how to build a robust prototype for ASL/Fingerspelling recognition. Its small size also makes it great for novice students and researchers interested in computer vision and the emerging area of gesture-based language recognition.

Technologies and Libraries Used:

This design leverages Python as the core programming language due to its simplicity and wide support for machine knowledge fabrics. The model development and training were done using TensorFlow and Keras, which give high- position APIs for structure and training deep knowledge models efficiently. Pivotal libraries used include:

1. TensorFlow/ Keras for erecting the CNN architecture, training the model, and handling data channels.
2. NumPy for numerical computations and array operations.
3. Matplotlib and Seaborn for imaging training performance criteria like delicacy and loss angles.
4. Pandas for running and assaying any fresh data structures (if demanded).

5. Scikit- learn for performance evaluation criteria analogous as confusion matrix and type report.

Data addition and preprocessing were also managed using TensorFlow's ImageDataGenerator. The entire design was executed in a Jupyter Tablet terrain, which offered an interactive interface for testing, imaging, and repeating snappily. These tools together handed a robust and flexible development terrain for erecting the ASL image type system.

Data Preprocessing:

Preprocessing was required to improve the performance and generalisability of the model, particularly as the dataset was small in size! All images were resized to one fixed sized to avoid inconsistent scales during training.

The data set was loaded via TensorFlow's ImageDataGenerator which not only provides a batched loading and labelling, but also allows for real time data augmentation. The following transformations were applied:

- Normalizing pixel values between 0 and 1.
- Horizontal flip to approximate for various hand orientations.
- Zoom factor for size variance.
- Adjust brightness of screen in different environments.
- Shear modification and small rotations to increase variance of the gestures.

The data was separated into training (80%) and validation (20%) sets based on directory, with equal class representation in the two sets.

No manual annotations were needed; the class labels were deduced from the folder names. This architecture facilitated preprocessing ease while maintaining the dataset structured and model-ready.

These pre-processing stages were important to minimize the overfitting and enhance the robustness as well as enable the CNN to learn more generalized features from the hand gesture images.

Model Architecture:

The model was built using a powerful and accurate image recognition architecture Convolutional Neural Network (CNN). Complex and performance, of the architecture design with variations of the dataset scale.

The model includes following layers:

- Input Layer: It is an input layer which takes RGB image of same sizes.
- Convolutional Layers: 3 sets of convolution blocks with increasing filters size (32, 64 & 128) followed by ReLU activation after each set to add a non-linearity.
- MaxPooling Layers: This is to be followed after every 2 Conv layers so as to decrease the spatial dimensions of the input, and to generalize better.
- Dropout Layers: Approximately 25–50% are added after a part of the pooling layers in order not to overfit.
- Flatten Layer: It will convert the 2D layer into 1D.
- Wide Layers: One or two fully connected layers to learn coarse features, with ReLU.
- Output Layer: Final dense layer with 29 output neurons (29 Classes) with SoftMax activation for multi-class classification.

Features from edges to textures to higher-level shapes, which are known to yield a more accurate hand configurations are among the features that are taken into account in this design.

Training Configuration:

A widely used Adam optimizer was used in the model as the optimizer that converges fast in deep learning problems. Categorical cross-entropy was employed as the loss function, as the task is multi-class classification across 29 output classes.

Key training configurations:

- Optimizer Adam (learning rate = 0.001)
- Loss Function: Categorical Crossentropy
- Metrics: Accuracy
- Batch Size: 32
- Epochs: 50 (modified depending on the performance of the validation)
- Validation Split: 20% of train directory < () >
- Callbacks: Model Checkpoint to save the best weight model and Early Stopping to stop training if validation loss didn't improve

The model was trained with TensorFlow. fit () (with real-time data augmentation by ImageDataGenerator). This allowed every epoch to train the model on different transformations of the original data, leading to better generalization.

The training was performed on a CPU without GPU acceleration as the model was an average sized model and the dataset was comparatively small. Loss and accuracy values were displayed for both training set and validation set to observe for overfitting and adjusting hyperparameters accordingly.

Results and Evaluation:

This shows good generalization performance of the model for ASL gestures. The final best validation accuracy achieved after 50 epochs, with early stopping, was about 97.6%, and even the training accuracy was just a tad higher, suggesting little or no over-fitting, thanks to the powerful augmentation and dropping layers.

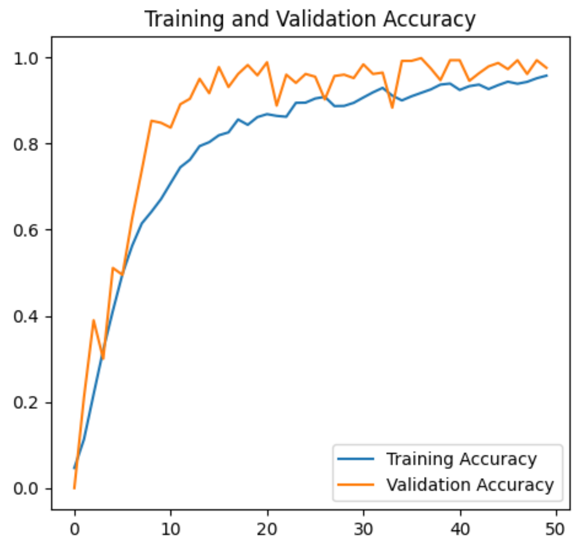
Key evaluation metrics included:

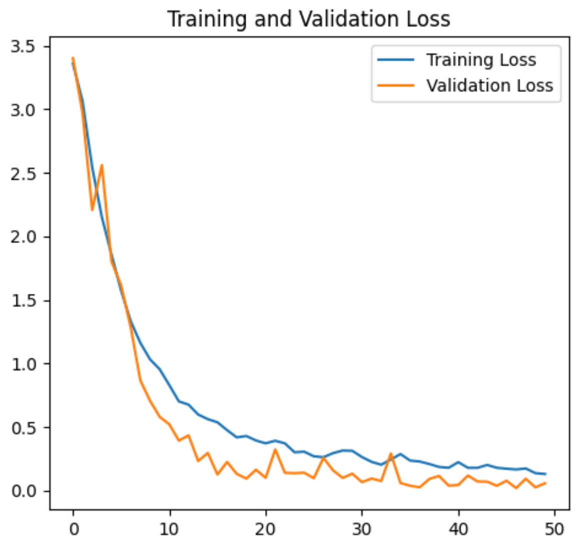
- Accuracy: Main measure of how well said classification does.
- Confusion Matrix: Offered a finer look in to the class-wise predictions and in turn helped in discerning which gesture(s) were most misclassified.

Such misclassifications can be observed, where most of the gestures were classified correctly, although the similar looking gestures (‘M’ and ‘N’ or ‘D’ and ‘R’) had a number of confusions, which is understandable due to hand-shape similarities.

Training/validation accuracy and loss plots across epochs validated smooth convergence and no apparent overfitting. Finally, despite the small size of the dataset, the model showed strong generalisation, apparently due to careful architecture design and to data augmentation.

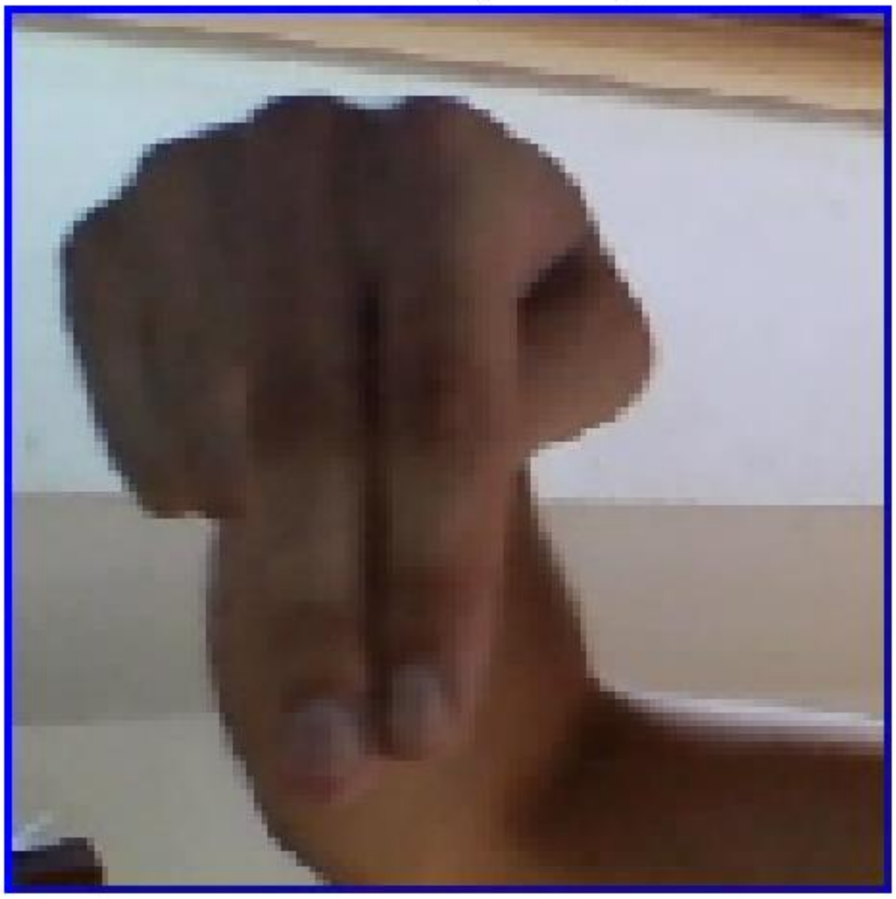
Results Graph:



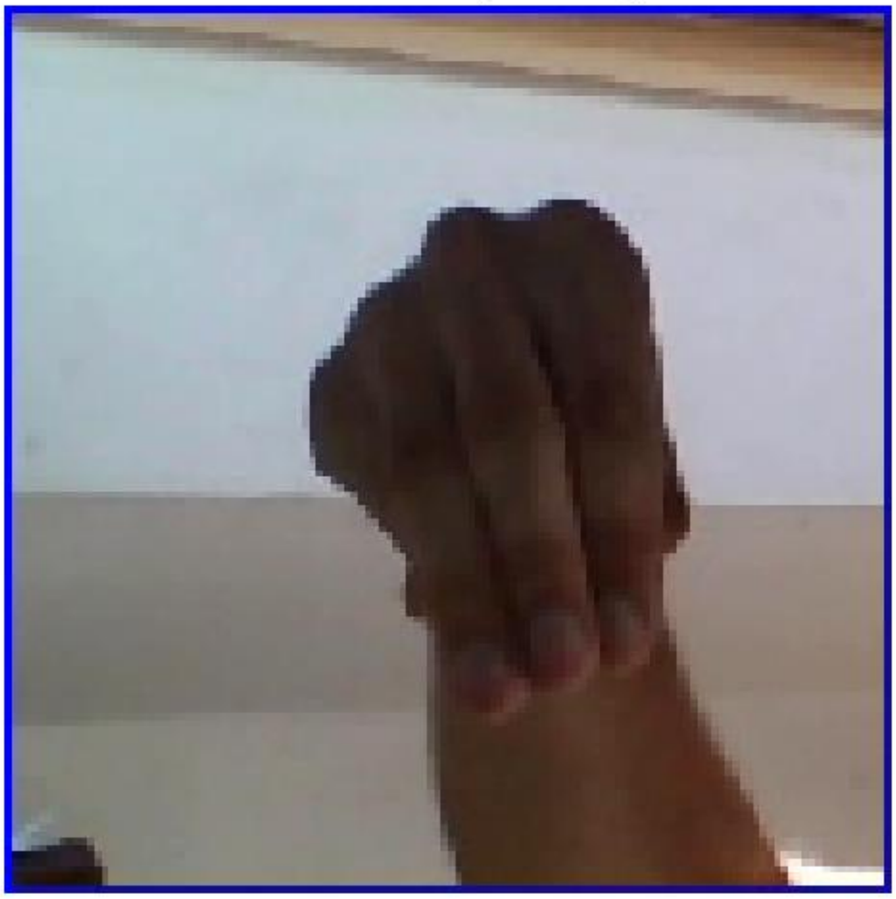


Sample Output:

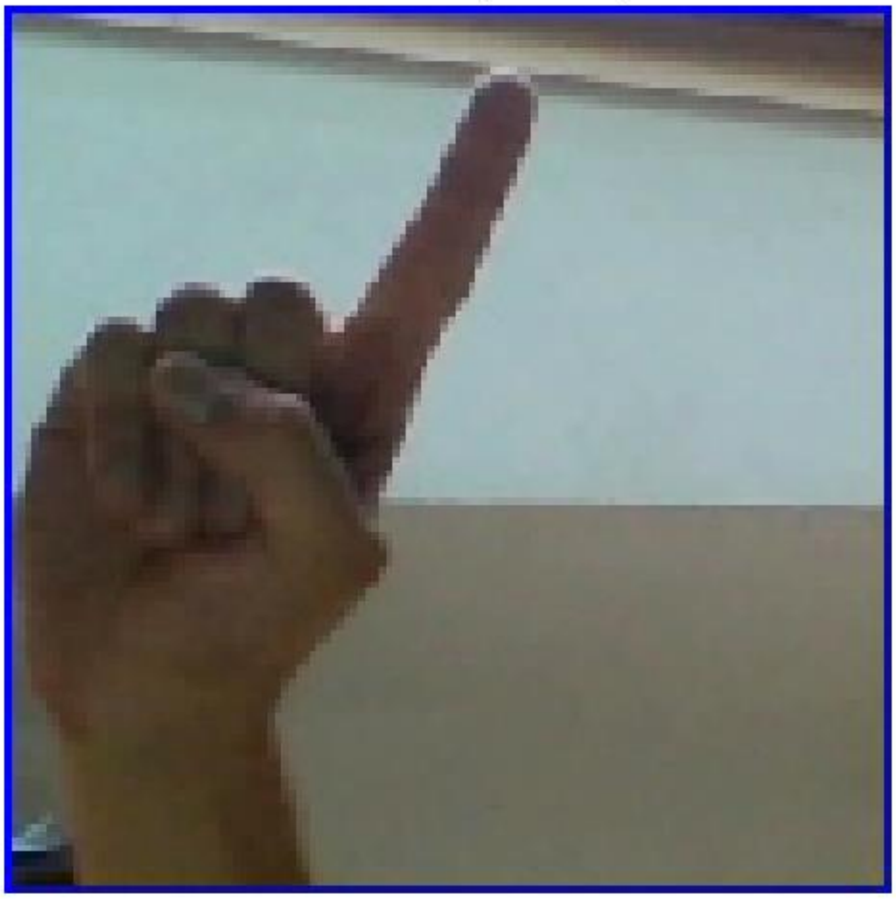
Prediction: N (100.00%)



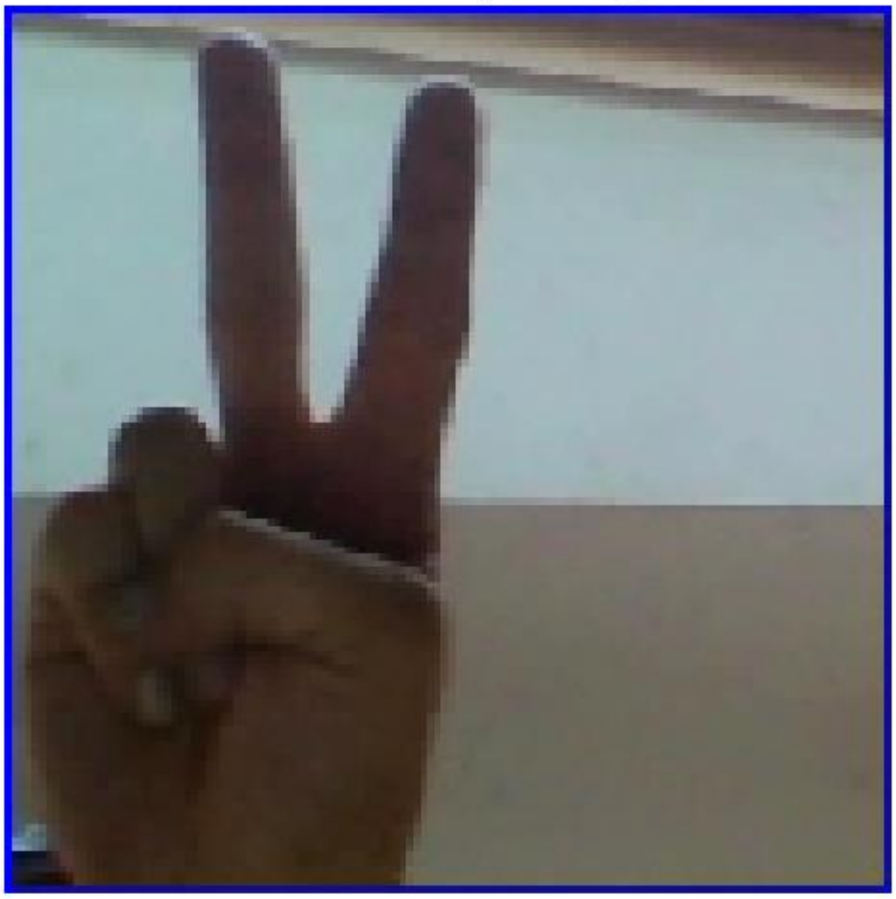
Prediction: M (100.00%)



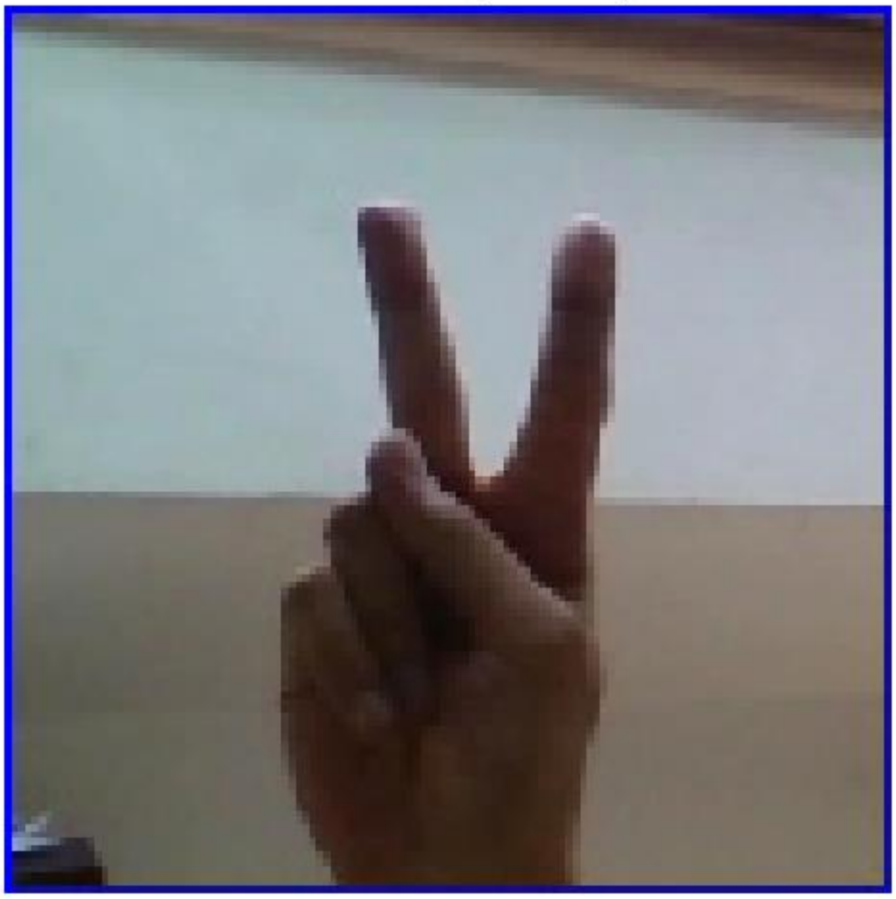
Prediction: Z (98.81%)



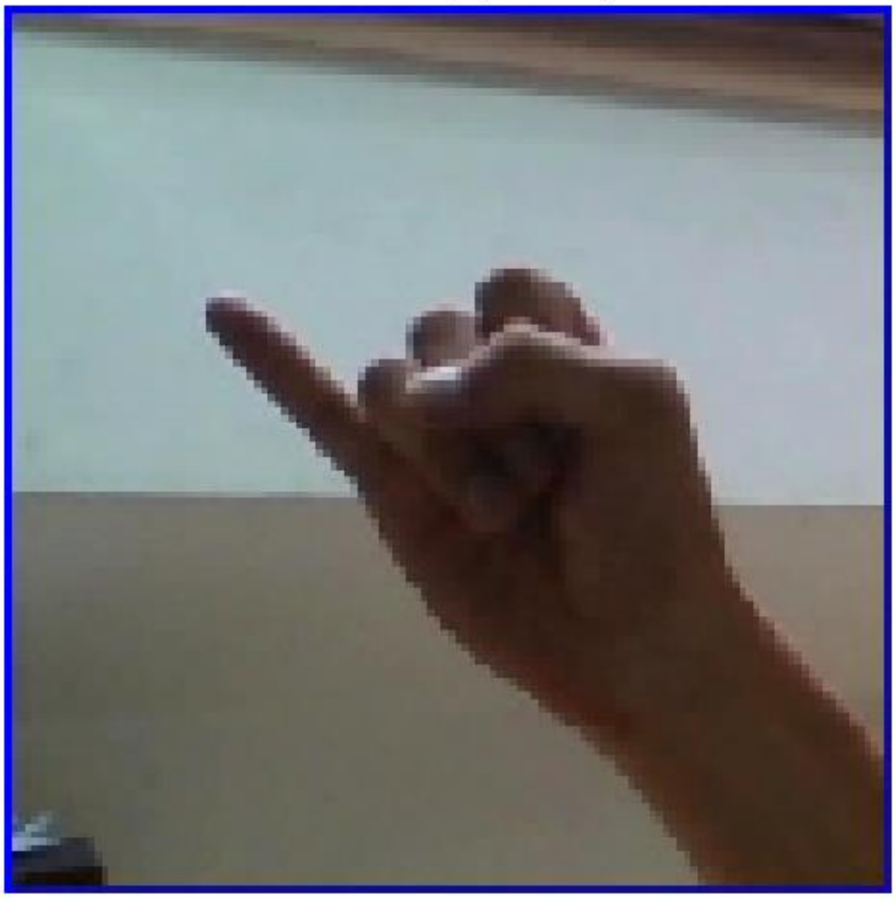
Prediction: V (100.00%)



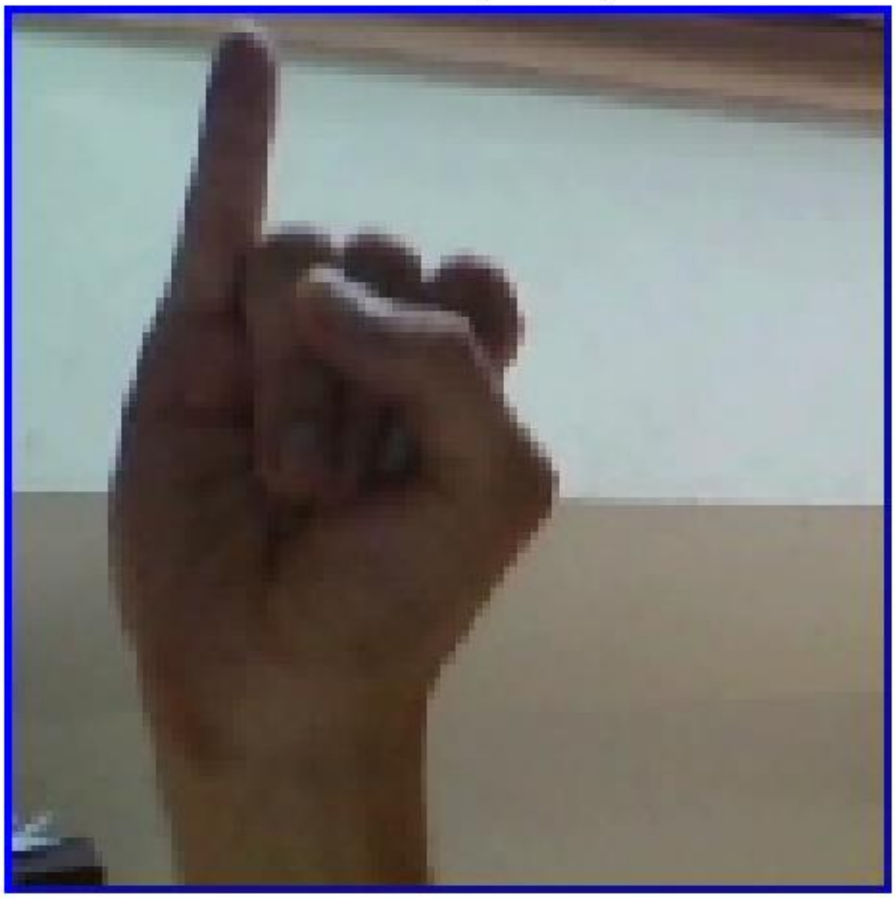
Prediction: K (100.00%)



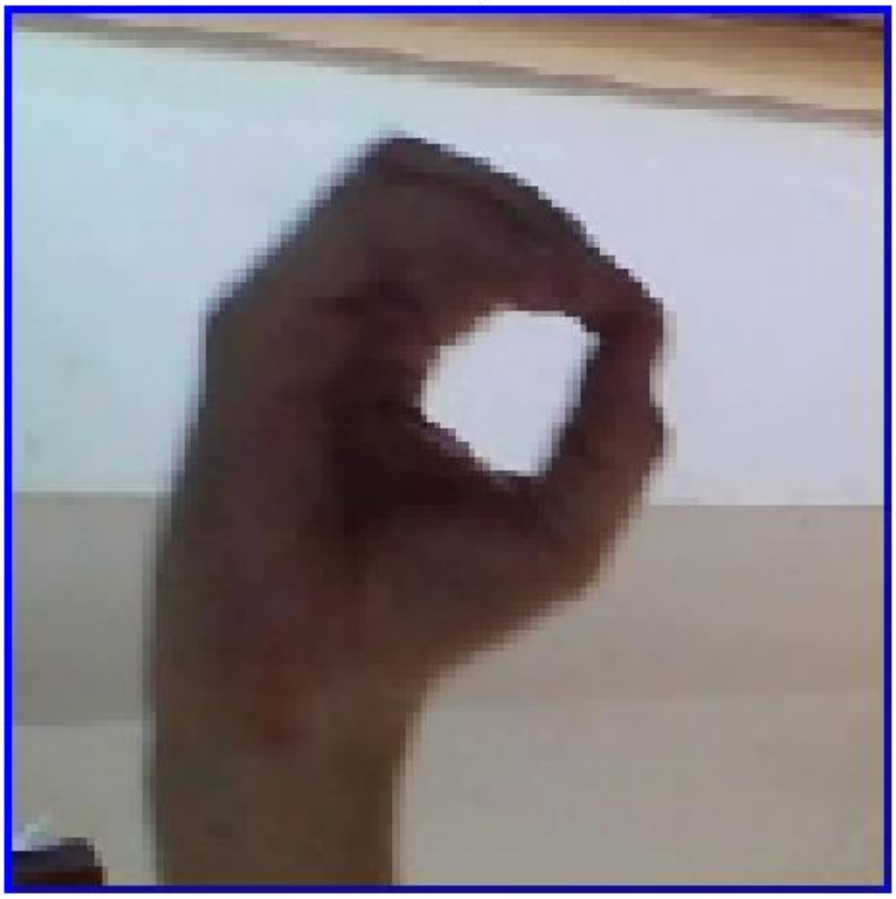
Prediction: J (81.29%)



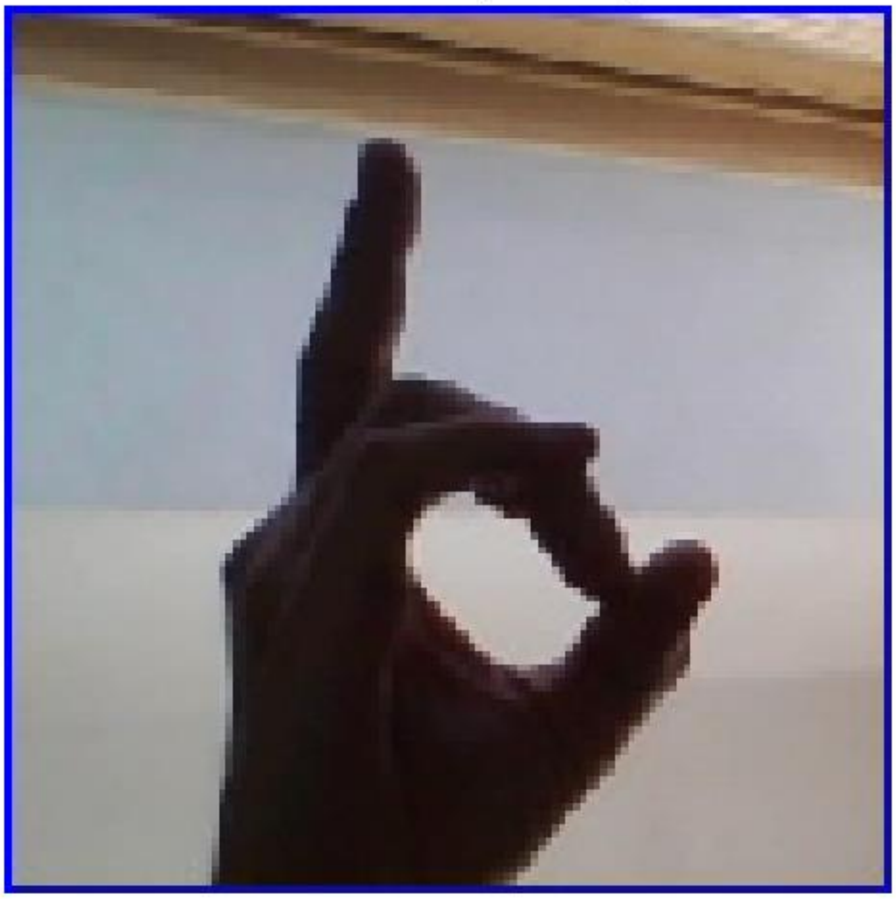
Prediction: I (88.23%)



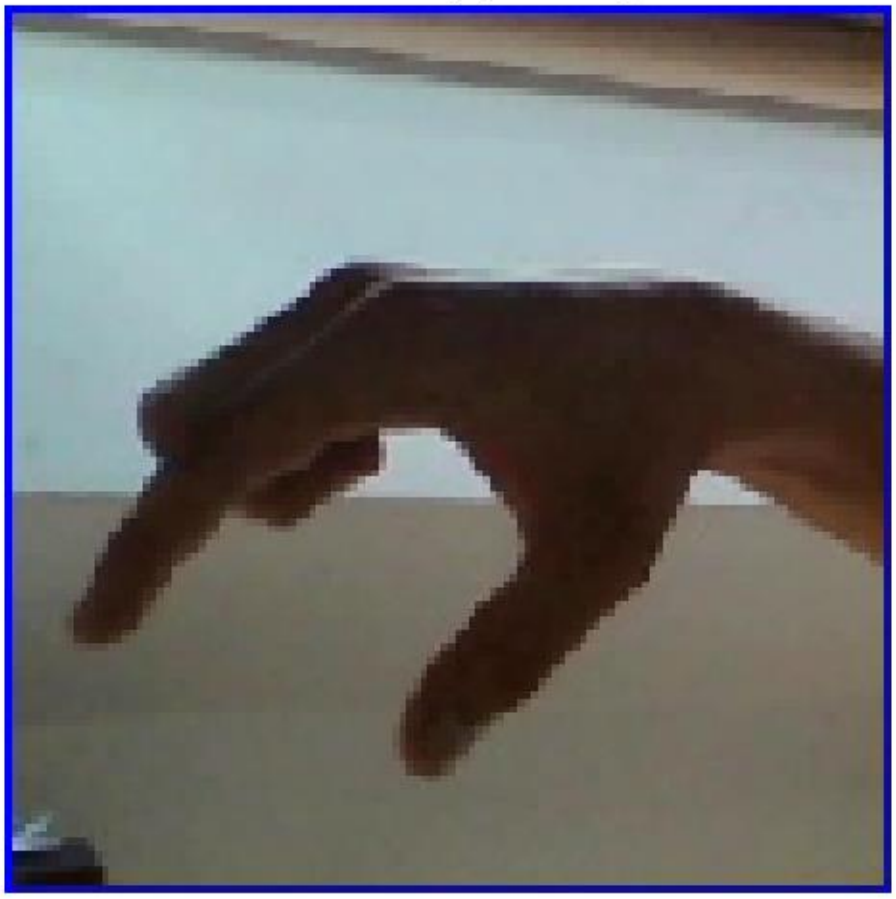
Prediction: O (99.91%)

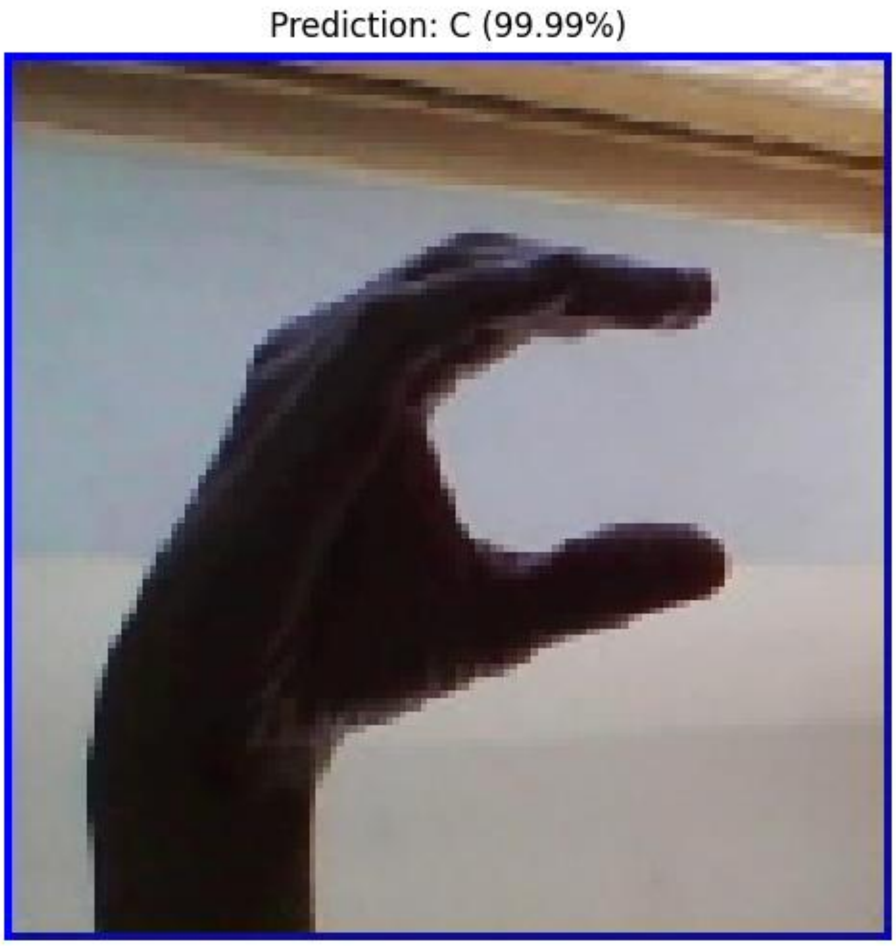


Prediction: D (100.00%)



Prediction: Q (100.00%)





Challenges Faced:

- A number of challenges are encountered in designing the ASL recognition model. The first one was the small size of the dataset – only 108 images per class -, which led the model to overfit. To overcome this limitation, aggressive data augmentation was used to mimic a larger variety during training.
- Another difficulty was visual similarity of some gestures, e.g. ‘M’ - ‘N’ and ‘D’ - ‘R’. These parallels lead to occasional misclassifications as the model was unable to disambiguate fine-grained hand placements.
- Second, images in the dataset were taken against a relatively unstructured background, and the subjects were not wearing glasses therefore the data did not represent average visual variability in natural scenes. The model may thus work less optimally in dynamic or poorly illuminated environments, unless it has been further trained on more diverse data.

- A CPU-based training also made experimentation cycle less efficient. Although CPU training was possible due to the moderate size of the dataset, exploration of deeper architectures and lengthier training times would have benefited substantially from GPU acceleration.
- Lastly, as for class imbalance, it wasn't a problem, but there was a need to be careful identifying folder names with class labels to avoid accidentally mislabelling files in silence.
- But overcoming the odds, the model emerged quite well-trained thanks to careful model design, regularization strategies, and a bit of luck while tuning.

Future Scope:

- While the current model performs well on static images, several improvements can be made to enhance its practical applicability. A key future direction is expanding the dataset with more diverse images, including different backgrounds, lighting conditions, and hand shapes to improve real-world generalization.
- Incorporating dynamic gesture recognition (e.g., using video sequences for signs that involve motion) would significantly broaden the model's capabilities, bringing it closer to full ASL interpretation rather than alphabet-only classification.
- The current model only covers alphabets and basic commands. Future versions could include ASL words or phrases, enabling real-time sentence construction. This would require integrating sequence models like LSTMs or Transformers for temporal learning.
- Deployment on mobile devices or edge systems using tools like TensorFlow Lite could help make the solution accessible and usable offline in educational or assistive environments.
- Additionally, integrating the model into an interactive user interface—such as a camera-based app that translates sign language into text or speech—would make it more usable for non-signers and help bridge communication gaps in daily life.

Conclusion:

The project achieved recognition system in American Sign Language (ASL) based on images with Convolutional Neural Network obtaining a validation accuracy of around 97.6% after 50 epochs. This shows the applicability of CNNs to hand gesture recognition problem. The model was trained on a balanced dataset with 3132 RGB images, over 29 classes of static ASL alphabets and some commands.

The performance of the model relied on strong data preprocessing and augmentation, including resizing, pixel normalization, horizontal flipping, zooming, brightness modification, shear transformation, and slight rotation, which reduced the risk of overfitting, thanks to small size of the dataset. Although the model had good generalization, the difficulty to distinguish certain gestures with visual similarities in between (e.g., 'M' and 'N', 'D' and 'R'), and unstructured background of images were two other challenges that caused mis-classifications. In the future, we will enlarge the dataset for diverse images and add dynamic gesture (video sequences) recognition; deploy the model on portable devices, in order to make the method more practical and to communicate real-time between disability person and computer.

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout, Rescaling
from tensorflow.keras.layers import RandomFlip, RandomRotation,
RandomZoom, RandomContrast
import numpy as np
import matplotlib.pyplot as plt
import os

DATA_DIR = 'E:\Projects\ASL'

IMAGE_SIZE = (128, 128)
BATCH_SIZE = 32
VALIDATION_SPLIT = 0.2

train_ds = tf.keras.utils.image_dataset_from_directory(
    DATA_DIR,
    labels='inferred',
    label_mode='categorical',
    image_size=IMAGE_SIZE,
    interpolation='nearest',
    batch_size=BATCH_SIZE,
    shuffle=True,
    validation_split=VALIDATION_SPLIT,
    subset='training',
    seed=123
)

val_ds = tf.keras.utils.image_dataset_from_directory(
    DATA_DIR,
    labels='inferred',
    label_mode='categorical',
    image_size=IMAGE_SIZE,
    interpolation='nearest',
    batch_size=BATCH_SIZE,
    shuffle=False,
    validation_split=VALIDATION_SPLIT,
    subset='validation',
    seed=123
)
```

```
Found 3132 files belonging to 29 classes.
Using 2506 files for training.
Found 3132 files belonging to 29 classes.
Using 626 files for validation.
```

```python
class_names = train_ds.class_names
NUM_CLASSES = len(class_names)
print(f"Detected {NUM_CLASSES} classes: {class_names}")
```

```
AUTOTUNE = tf.data.AUTOTUNE
train_ds =
train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)

data_augmentation = Sequential([
    RandomFlip("horizontal_and_vertical"),
    RandomRotation(0.2),
    RandomZoom(0.2),
    RandomContrast(0.2),
], name="data_augmentation")

Detected 29 classes: ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
'X', 'Y', 'Z', 'del', 'nothing', 'space']

model = Sequential([
    data_augmentation,
    Rescaling(1./255),
    Conv2D(32, (3, 3), activation='relu', input_shape=(IMAGE_SIZE[0],
IMAGE_SIZE[1], 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(NUM_CLASSES, activation='softmax')
])

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

model.summary()

EPOCHS = 50

history = model.fit(
    train_ds,
    epochs=EPOCHS,
    validation_data=val_ds,
)

acc = history.history['accuracy']
```

```
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(EPOCHS)

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

model.save('ASL.keras')
print("Model saved as 'ASL.keras'")
```

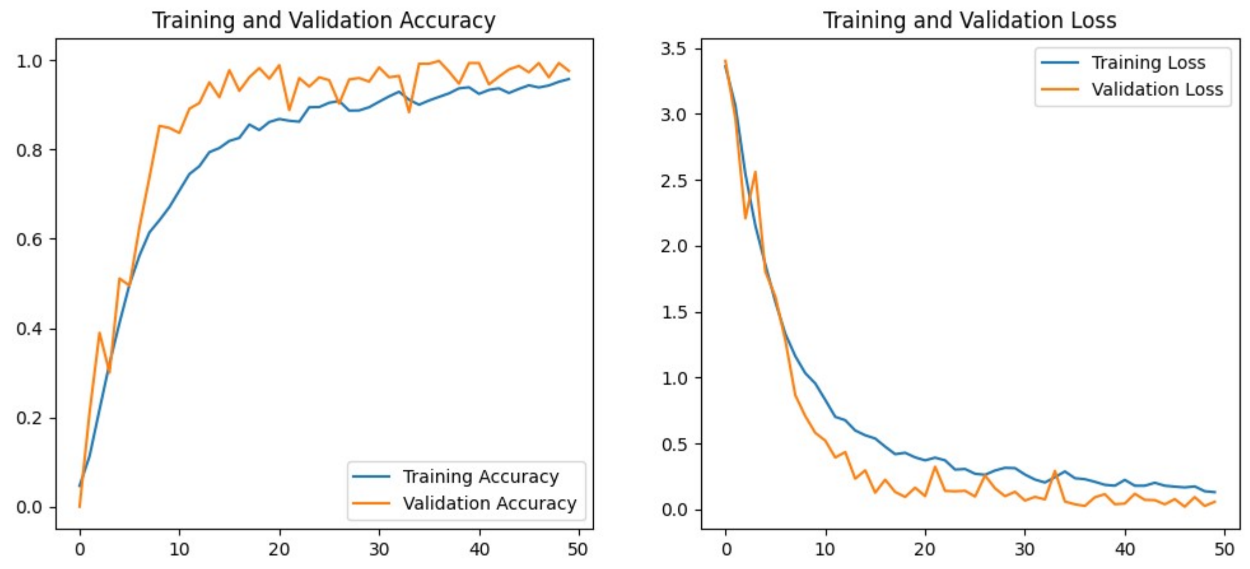
```
C:\Users\tanis\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\keras\src\layers\convolutional\
base_conv.py:99: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(

Model: "sequential_1"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
━━━━━┓
┃ Layer (type)                     ┃ Output Shape            ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
━━━━━┩
│ data_augmentation (Sequential)   │ ?                       │   0
(unbuilt) │
├──────────────────────────────────┼─────────────────────────┼──────────
─────┤
│ rescaling_1 (Rescaling)          │ ?                       │   0
(unbuilt) │
├──────────────────────────────────┼─────────────────────────┼──────────
─────┤
│ conv2d_3 (Conv2D)                │ ?                       │   0
(unbuilt) │
├──────────────────────────────────┼─────────────────────────┼──────────
```

```
──────┤
│ max_pooling2d_3 (MaxPooling2D)  │ ?                      │   0
(unbuilt) │
├─────────────────────────────────┼────────────────────────┼──────
──────┤
│ conv2d_4 (Conv2D)               │ ?                      │   0
(unbuilt) │
├─────────────────────────────────┼────────────────────────┼──────
──────┤
│ max_pooling2d_4 (MaxPooling2D)  │ ?                      │   0
(unbuilt) │
├─────────────────────────────────┼────────────────────────┼──────
──────┤
│ conv2d_5 (Conv2D)               │ ?                      │   0
(unbuilt) │
├─────────────────────────────────┼────────────────────────┼──────
──────┤
│ max_pooling2d_5 (MaxPooling2D)  │ ?                      │   0
(unbuilt) │
├─────────────────────────────────┼────────────────────────┼──────
──────┤
│ flatten_1 (Flatten)             │ ?                      │   0
(unbuilt) │
├─────────────────────────────────┼────────────────────────┼──────
──────┤
│ dense_2 (Dense)                 │ ?                      │   0
(unbuilt) │
├─────────────────────────────────┼────────────────────────┼──────
──────┤
│ dropout_1 (Dropout)             │ ?                      │
0 │
├─────────────────────────────────┼────────────────────────┼──────
──────┤
│ dense_3 (Dense)                 │ ?                      │   0
(unbuilt) │
└─────────────────────────────────┴────────────────────────┴──────
──────┘

 Total params: 0 (0.00 B)

 Trainable params: 0 (0.00 B)

 Non-trainable params: 0 (0.00 B)

Epoch 1/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 38s 366ms/step - accuracy: 0.0444 - loss:
3.4101 - val_accuracy: 0.0000e+00 - val_loss: 3.4021
Epoch 2/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 27s 343ms/step - accuracy: 0.0872 - loss:
3.1579 - val_accuracy: 0.2109 - val_loss: 2.9677
```

```
Epoch 3/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 349ms/step - accuracy: 0.1959 - loss:
2.6365 - val_accuracy: 0.3898 - val_loss: 2.2066
Epoch 4/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 352ms/step - accuracy: 0.3080 - loss:
2.2226 - val_accuracy: 0.3003 - val_loss: 2.5601
Epoch 5/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 350ms/step - accuracy: 0.4015 - loss:
1.9076 - val_accuracy: 0.5112 - val_loss: 1.8026
Epoch 6/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 27s 341ms/step - accuracy: 0.4737 - loss:
1.6371 - val_accuracy: 0.4952 - val_loss: 1.6131
Epoch 7/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 29s 361ms/step - accuracy: 0.5558 - loss:
1.3424 - val_accuracy: 0.6262 - val_loss: 1.2818
Epoch 8/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 29s 371ms/step - accuracy: 0.6089 - loss:
1.1770 - val_accuracy: 0.7380 - val_loss: 0.8646
Epoch 9/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 27s 342ms/step - accuracy: 0.6380 - loss:
1.0585 - val_accuracy: 0.8530 - val_loss: 0.7057
Epoch 10/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 29s 363ms/step - accuracy: 0.6538 - loss:
1.0140 - val_accuracy: 0.8482 - val_loss: 0.5791
Epoch 11/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 29s 372ms/step - accuracy: 0.6949 - loss:
0.8728 - val_accuracy: 0.8371 - val_loss: 0.5203
Epoch 12/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 27s 346ms/step - accuracy: 0.7309 - loss:
0.7128 - val_accuracy: 0.8914 - val_loss: 0.3917
Epoch 13/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 30s 373ms/step - accuracy: 0.7577 - loss:
0.6819 - val_accuracy: 0.9042 - val_loss: 0.4332
Epoch 14/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 29s 371ms/step - accuracy: 0.7863 - loss:
0.5974 - val_accuracy: 0.9505 - val_loss: 0.2311
Epoch 15/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 357ms/step - accuracy: 0.8010 - loss:
0.5432 - val_accuracy: 0.9169 - val_loss: 0.2947
Epoch 16/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 355ms/step - accuracy: 0.7984 - loss:
0.6012 - val_accuracy: 0.9776 - val_loss: 0.1247
Epoch 17/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 353ms/step - accuracy: 0.8121 - loss:
0.4968 - val_accuracy: 0.9313 - val_loss: 0.2237
Epoch 18/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 354ms/step - accuracy: 0.8558 - loss:
0.4202 - val_accuracy: 0.9617 - val_loss: 0.1302
Epoch 19/50
```

```
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 358ms/step - accuracy: 0.8409 - loss:
0.4432 - val_accuracy: 0.9824 - val_loss: 0.0928
Epoch 20/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 352ms/step - accuracy: 0.8591 - loss:
0.3899 - val_accuracy: 0.9585 - val_loss: 0.1629
Epoch 21/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 29s 371ms/step - accuracy: 0.8778 - loss:
0.3521 - val_accuracy: 0.9888 - val_loss: 0.0988
Epoch 22/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 33s 413ms/step - accuracy: 0.8677 - loss:
0.3804 - val_accuracy: 0.8882 - val_loss: 0.3221
Epoch 23/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 31s 393ms/step - accuracy: 0.8644 - loss:
0.3706 - val_accuracy: 0.9601 - val_loss: 0.1385
Epoch 24/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 32s 402ms/step - accuracy: 0.8929 - loss:
0.3205 - val_accuracy: 0.9409 - val_loss: 0.1352
Epoch 25/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 31s 391ms/step - accuracy: 0.8883 - loss:
0.3207 - val_accuracy: 0.9617 - val_loss: 0.1393
Epoch 26/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 361ms/step - accuracy: 0.9003 - loss:
0.2812 - val_accuracy: 0.9553 - val_loss: 0.0957
Epoch 27/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 360ms/step - accuracy: 0.9173 - loss:
0.2329 - val_accuracy: 0.9026 - val_loss: 0.2559
Epoch 28/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 355ms/step - accuracy: 0.8874 - loss:
0.2937 - val_accuracy: 0.9569 - val_loss: 0.1579
Epoch 29/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 354ms/step - accuracy: 0.8768 - loss:
0.3399 - val_accuracy: 0.9601 - val_loss: 0.0983
Epoch 30/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 353ms/step - accuracy: 0.9124 - loss:
0.2520 - val_accuracy: 0.9521 - val_loss: 0.1315
Epoch 31/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 360ms/step - accuracy: 0.8898 - loss:
0.3046 - val_accuracy: 0.9840 - val_loss: 0.0652
Epoch 32/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 353ms/step - accuracy: 0.9140 - loss:
0.2520 - val_accuracy: 0.9617 - val_loss: 0.0931
Epoch 33/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 353ms/step - accuracy: 0.9378 - loss:
0.1822 - val_accuracy: 0.9649 - val_loss: 0.0739
Epoch 34/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 354ms/step - accuracy: 0.9106 - loss:
0.2503 - val_accuracy: 0.8834 - val_loss: 0.2902
Epoch 35/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 358ms/step - accuracy: 0.9057 - loss:
```

```
0.2792 - val_accuracy: 0.9920 - val_loss: 0.0578
Epoch 36/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 358ms/step - accuracy: 0.9022 - loss:
0.2375 - val_accuracy: 0.9920 - val_loss: 0.0368
Epoch 37/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 29s 363ms/step - accuracy: 0.9165 - loss:
0.2284 - val_accuracy: 0.9984 - val_loss: 0.0243
Epoch 38/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 361ms/step - accuracy: 0.9246 - loss:
0.2032 - val_accuracy: 0.9744 - val_loss: 0.0902
Epoch 39/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 357ms/step - accuracy: 0.9355 - loss:
0.1884 - val_accuracy: 0.9473 - val_loss: 0.1137
Epoch 40/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 353ms/step - accuracy: 0.9387 - loss:
0.1757 - val_accuracy: 0.9936 - val_loss: 0.0374
Epoch 41/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 356ms/step - accuracy: 0.9259 - loss:
0.2146 - val_accuracy: 0.9936 - val_loss: 0.0424
Epoch 42/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 352ms/step - accuracy: 0.9300 - loss:
0.1977 - val_accuracy: 0.9457 - val_loss: 0.1168
Epoch 43/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 356ms/step - accuracy: 0.9453 - loss:
0.1499 - val_accuracy: 0.9633 - val_loss: 0.0702
Epoch 44/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 354ms/step - accuracy: 0.9361 - loss:
0.1749 - val_accuracy: 0.9792 - val_loss: 0.0685
Epoch 45/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 354ms/step - accuracy: 0.9312 - loss:
0.1967 - val_accuracy: 0.9872 - val_loss: 0.0364
Epoch 46/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 354ms/step - accuracy: 0.9536 - loss:
0.1398 - val_accuracy: 0.9728 - val_loss: 0.0762
Epoch 47/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 29s 369ms/step - accuracy: 0.9314 - loss:
0.1799 - val_accuracy: 0.9936 - val_loss: 0.0182
Epoch 48/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 359ms/step - accuracy: 0.9473 - loss:
0.1628 - val_accuracy: 0.9617 - val_loss: 0.0917
Epoch 49/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 355ms/step - accuracy: 0.9533 - loss:
0.1329 - val_accuracy: 0.9936 - val_loss: 0.0240
Epoch 50/50
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 353ms/step - accuracy: 0.9595 - loss:
0.1247 - val_accuracy: 0.9760 - val_loss: 0.0561
```



```
Model saved as 'ASL.keras'
```

```
#Tester

tf.config.set_visible_devices([], 'GPU')
print("Configured to use CPU only.")

MODEL_PATH = 'ASL.keras'
IMAGE_PATH = "E:\\Projects\\asl_alphabet_test\\asl_alphabet_test\\
C_test.jpg"

IMAGE_SIZE = (128, 128)

CLASS_NAMES = sorted(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
'X', 'Y', 'Z', 'del', 'nothing', 'space'])

model = tf.keras.models.load_model(MODEL_PATH)

def preprocess_image(image_path, target_size):
    img = tf.keras.preprocessing.image.load_img(image_path,
target_size=target_size)
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    return img_array

test_image_array = preprocess_image(IMAGE_PATH, IMAGE_SIZE)

predictions = model.predict(test_image_array)

predicted_probabilities = predictions[0]
predicted_class_index = np.argmax(predicted_probabilities)
predicted_class_name = CLASS_NAMES[predicted_class_index]
```

```
confidence = predicted_probabilities[predicted_class_index] * 100

print(f"\nPredicted class: {predicted_class_name}")
print(f"Confidence: {confidence:.2f}%")

plt.figure(figsize=(6, 6))
display_img = tf.keras.preprocessing.image.load_img(IMAGE_PATH,
target_size=IMAGE_SIZE)
plt.imshow(display_img)
plt.title(f"Prediction: {predicted_class_name} ({confidence:.2f}%)")
plt.axis('off')
plt.show()
```

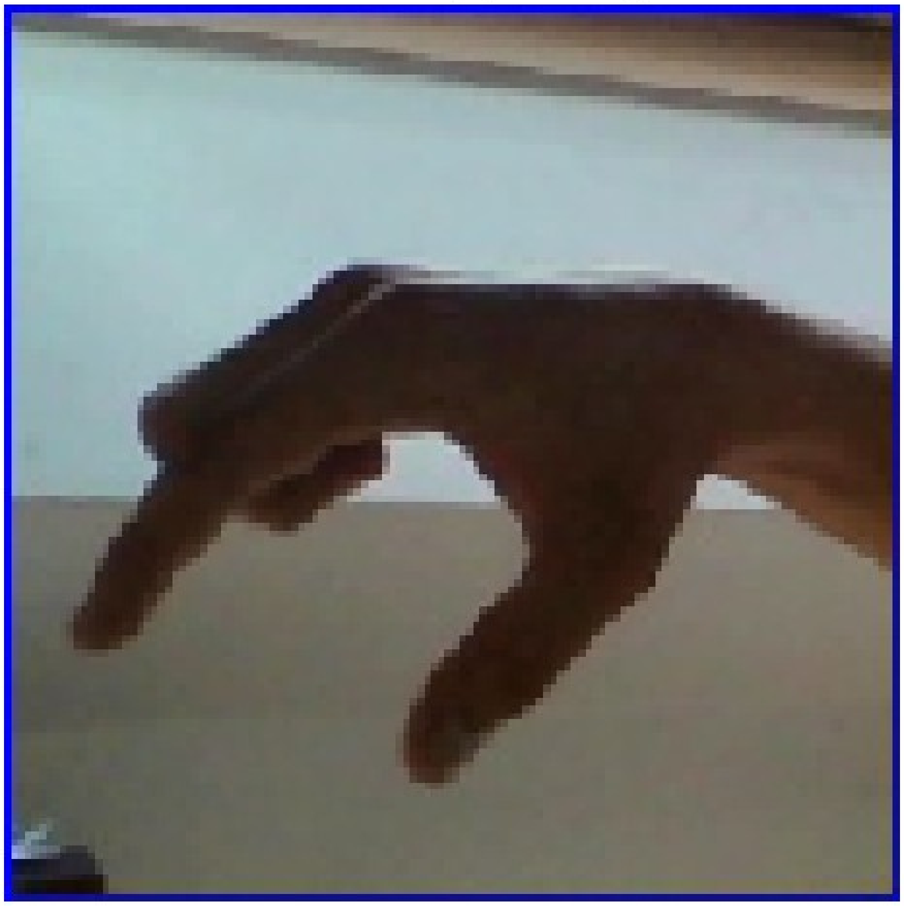
```
Configured to use CPU only.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step

Predicted class: C
Confidence: 99.99%
```

Prediction: C (99.99%)

```
#Tester

tf.config.set_visible_devices([], 'GPU')
print("Configured to use CPU only.")

MODEL_PATH = 'ASL.keras'
IMAGE_PATH = "E:\\Projects\\asl_alphabet_test\\asl_alphabet_test\\
Q_test.jpg"

IMAGE_SIZE = (128, 128)

CLASS_NAMES = sorted(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
'X', 'Y', 'Z', 'del', 'nothing', 'space'])

model = tf.keras.models.load_model(MODEL_PATH)

def preprocess_image(image_path, target_size):
    img = tf.keras.preprocessing.image.load_img(image_path,
target_size=target_size)
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    return img_array

test_image_array = preprocess_image(IMAGE_PATH, IMAGE_SIZE)

predictions = model.predict(test_image_array)

predicted_probabilities = predictions[0]
predicted_class_index = np.argmax(predicted_probabilities)
predicted_class_name = CLASS_NAMES[predicted_class_index]
confidence = predicted_probabilities[predicted_class_index] * 100

print(f"\nPredicted class: {predicted_class_name}")
print(f"Confidence: {confidence:.2f}%")

plt.figure(figsize=(6, 6))
display_img = tf.keras.preprocessing.image.load_img(IMAGE_PATH,
target_size=IMAGE_SIZE)
plt.imshow(display_img)
plt.title(f"Prediction: {predicted_class_name} ({confidence:.2f}%)")
plt.axis('off')
plt.show()
```

```
Configured to use CPU only.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 135ms/step

Predicted class: Q
Confidence: 100.00%
```

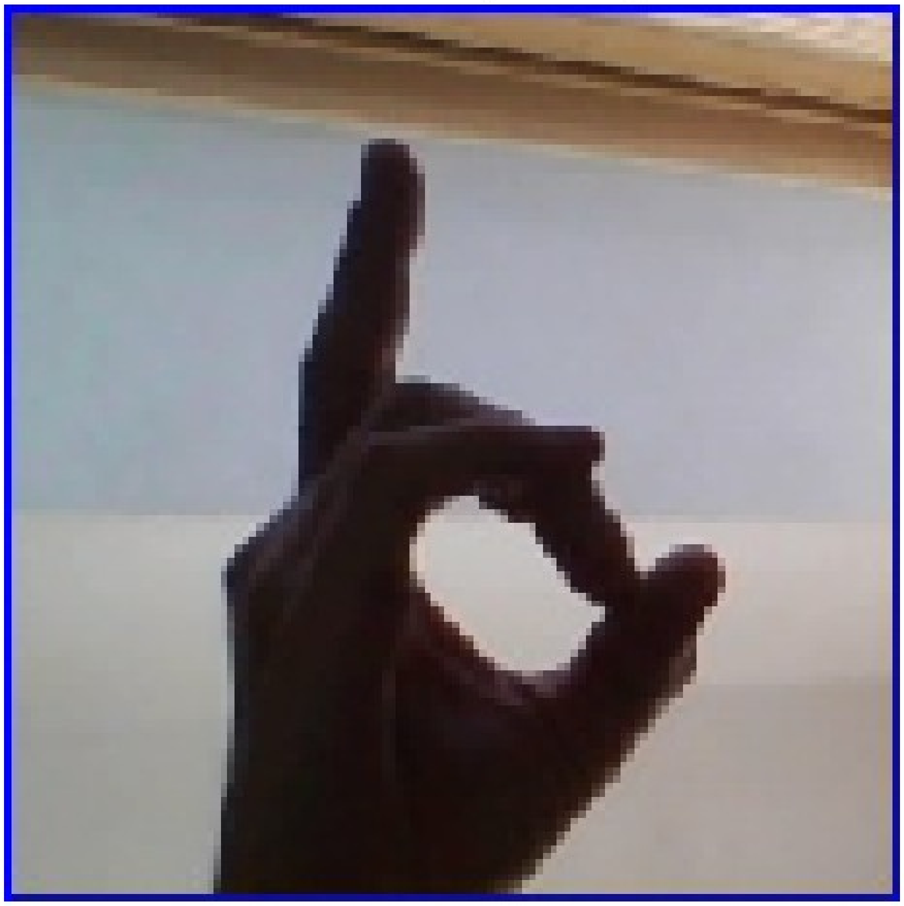
Prediction: Q (100.00%)



```
#Tester

tf.config.set_visible_devices([], 'GPU')
print("Configured to use CPU only.")

MODEL_PATH = 'ASL.keras'
IMAGE_PATH = "E:\\Projects\\asl_alphabet_test\\asl_alphabet_test\\
D_test.jpg"

IMAGE_SIZE = (128, 128)

CLASS_NAMES = sorted(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
'X', 'Y', 'Z', 'del', 'nothing', 'space'])

model = tf.keras.models.load_model(MODEL_PATH)

def preprocess_image(image_path, target_size):
    img = tf.keras.preprocessing.image.load_img(image_path,
target_size=target_size)
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    return img_array

test_image_array = preprocess_image(IMAGE_PATH, IMAGE_SIZE)

predictions = model.predict(test_image_array)

predicted_probabilities = predictions[0]
predicted_class_index = np.argmax(predicted_probabilities)
predicted_class_name = CLASS_NAMES[predicted_class_index]
confidence = predicted_probabilities[predicted_class_index] * 100

print(f"\nPredicted class: {predicted_class_name}")
print(f"Confidence: {confidence:.2f}%")

plt.figure(figsize=(6, 6))
display_img = tf.keras.preprocessing.image.load_img(IMAGE_PATH,
target_size=IMAGE_SIZE)
plt.imshow(display_img)
plt.title(f"Prediction: {predicted_class_name} ({confidence:.2f}%)")
plt.axis('off')
plt.show()
```

```
Configured to use CPU only.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 149ms/step

Predicted class: D
Confidence: 100.00%
```

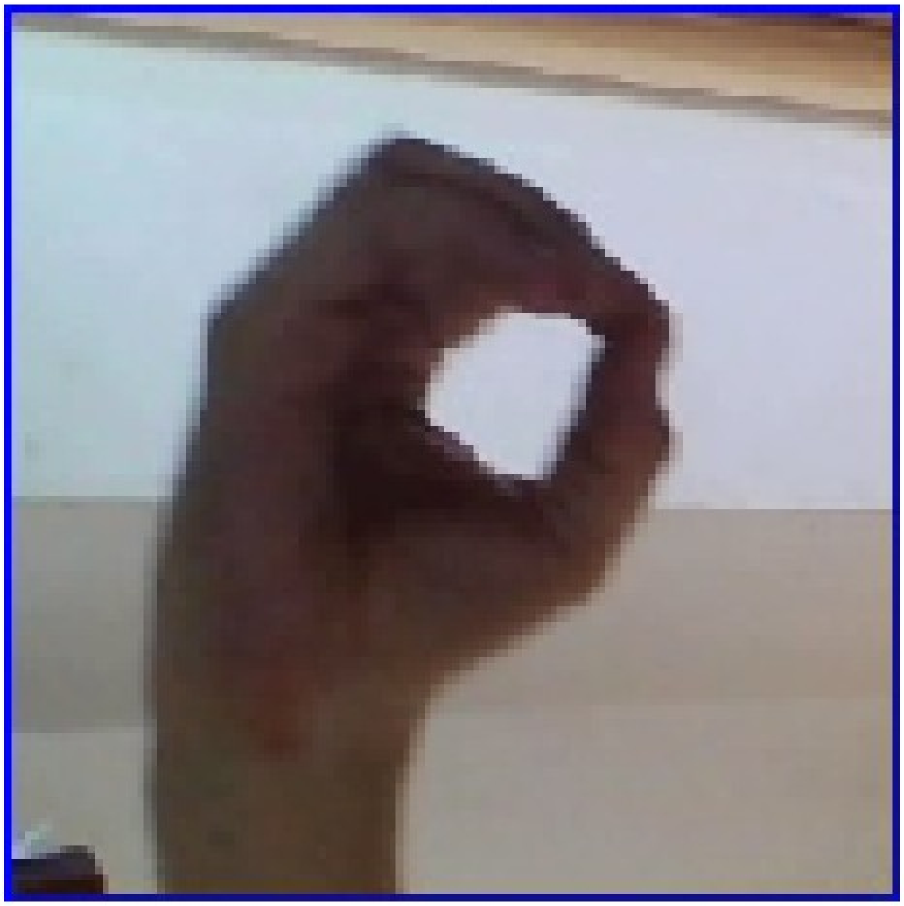
Prediction: D (100.00%)



```
#Tester

tf.config.set_visible_devices([], 'GPU')
print("Configured to use CPU only.")

MODEL_PATH = 'ASL.keras'
IMAGE_PATH = "E:\\Projects\\asl_alphabet_test\\asl_alphabet_test\\
O_test.jpg"

IMAGE_SIZE = (128, 128)

CLASS_NAMES = sorted(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
'X', 'Y', 'Z', 'del', 'nothing', 'space'])

model = tf.keras.models.load_model(MODEL_PATH)

def preprocess_image(image_path, target_size):
    img = tf.keras.preprocessing.image.load_img(image_path,
target_size=target_size)
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    return img_array

test_image_array = preprocess_image(IMAGE_PATH, IMAGE_SIZE)

predictions = model.predict(test_image_array)

predicted_probabilities = predictions[0]
predicted_class_index = np.argmax(predicted_probabilities)
predicted_class_name = CLASS_NAMES[predicted_class_index]
confidence = predicted_probabilities[predicted_class_index] * 100

print(f"\nPredicted class: {predicted_class_name}")
print(f"Confidence: {confidence:.2f}%")

plt.figure(figsize=(6, 6))
display_img = tf.keras.preprocessing.image.load_img(IMAGE_PATH,
target_size=IMAGE_SIZE)
plt.imshow(display_img)
plt.title(f"Prediction: {predicted_class_name} ({confidence:.2f}%)")
plt.axis('off')
plt.show()
```

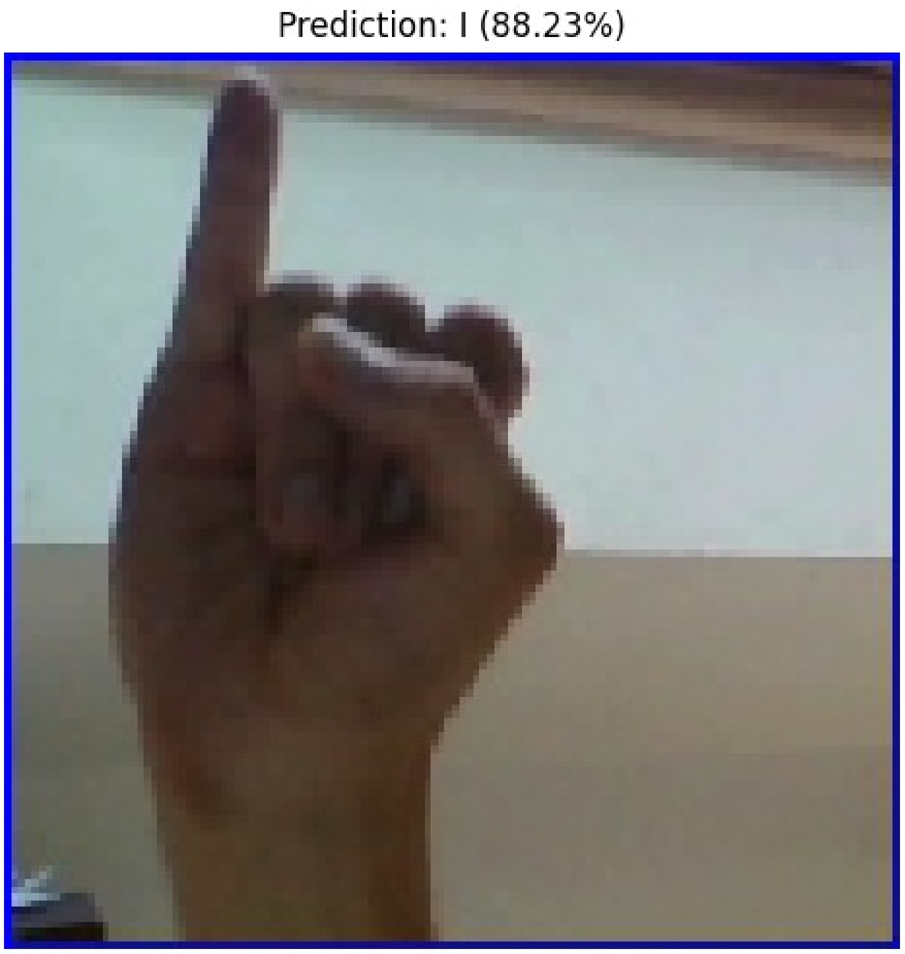
```
Configured to use CPU only.
WARNING:tensorflow:5 out of the last 5 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_dist
ributed at 0x0000017F3ED3F380> triggered tf.function retracing.
Tracing is expensive and the excessive number of tracings could be due
to (1) creating @tf.function repeatedly in a loop, (2) passing tensors
with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True option that can avoid
unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more
details.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 161ms/step

Predicted class: 0
Confidence: 99.91%
```

Prediction: O (99.91%)



```
#Tester

tf.config.set_visible_devices([], 'GPU')
print("Configured to use CPU only.")

MODEL_PATH = 'ASL.keras'
IMAGE_PATH = "E:\\Projects\\asl_alphabet_test\\asl_alphabet_test\\
I_test.jpg"

IMAGE_SIZE = (128, 128)

CLASS_NAMES = sorted(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
'X', 'Y', 'Z', 'del', 'nothing', 'space'])

model = tf.keras.models.load_model(MODEL_PATH)

def preprocess_image(image_path, target_size):
    img = tf.keras.preprocessing.image.load_img(image_path,
target_size=target_size)
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    return img_array

test_image_array = preprocess_image(IMAGE_PATH, IMAGE_SIZE)

predictions = model.predict(test_image_array)

predicted_probabilities = predictions[0]
predicted_class_index = np.argmax(predicted_probabilities)
predicted_class_name = CLASS_NAMES[predicted_class_index]
confidence = predicted_probabilities[predicted_class_index] * 100

print(f"\nPredicted class: {predicted_class_name}")
print(f"Confidence: {confidence:.2f}%")

plt.figure(figsize=(6, 6))
display_img = tf.keras.preprocessing.image.load_img(IMAGE_PATH,
target_size=IMAGE_SIZE)
plt.imshow(display_img)
plt.title(f"Prediction: {predicted_class_name} ({confidence:.2f}%)")
plt.axis('off')
plt.show()
```

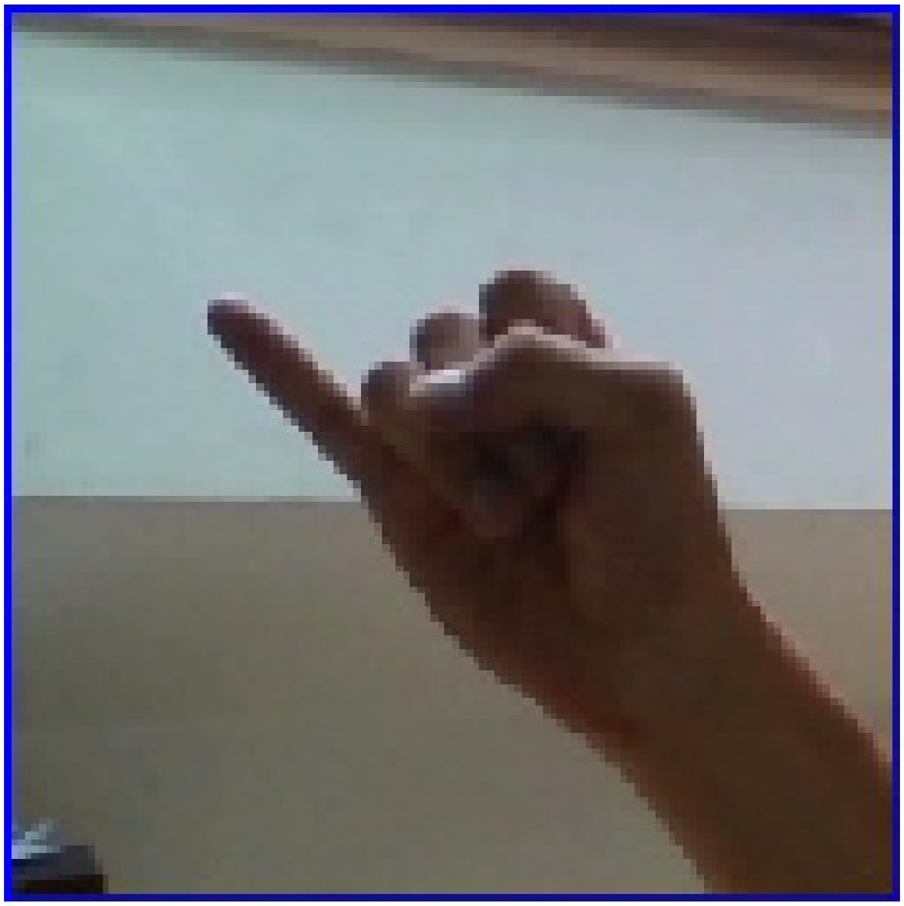
```
Configured to use CPU only.
WARNING:tensorflow:6 out of the last 6 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_dist
ributed at 0x0000017F3EEE7740> triggered tf.function retracing.
Tracing is expensive and the excessive number of tracings could be due
to (1) creating @tf.function repeatedly in a loop, (2) passing tensors
with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True option that can avoid
unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more
details.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step

Predicted class: I
Confidence: 88.23%
```



```
#Tester

tf.config.set_visible_devices([], 'GPU')
print("Configured to use CPU only.")

MODEL_PATH = 'ASL.keras'
IMAGE_PATH = "E:\\Projects\\asl_alphabet_test\\asl_alphabet_test\\
J_test.jpg"

IMAGE_SIZE = (128, 128)

CLASS_NAMES = sorted(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
'X', 'Y', 'Z', 'del', 'nothing', 'space'])

model = tf.keras.models.load_model(MODEL_PATH)

def preprocess_image(image_path, target_size):
    img = tf.keras.preprocessing.image.load_img(image_path,
target_size=target_size)
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    return img_array

test_image_array = preprocess_image(IMAGE_PATH, IMAGE_SIZE)

predictions = model.predict(test_image_array)

predicted_probabilities = predictions[0]
predicted_class_index = np.argmax(predicted_probabilities)
predicted_class_name = CLASS_NAMES[predicted_class_index]
confidence = predicted_probabilities[predicted_class_index] * 100

print(f"\nPredicted class: {predicted_class_name}")
print(f"Confidence: {confidence:.2f}%")

plt.figure(figsize=(6, 6))
display_img = tf.keras.preprocessing.image.load_img(IMAGE_PATH,
target_size=IMAGE_SIZE)
plt.imshow(display_img)
plt.title(f"Prediction: {predicted_class_name} ({confidence:.2f}%)")
plt.axis('off')
plt.show()
```

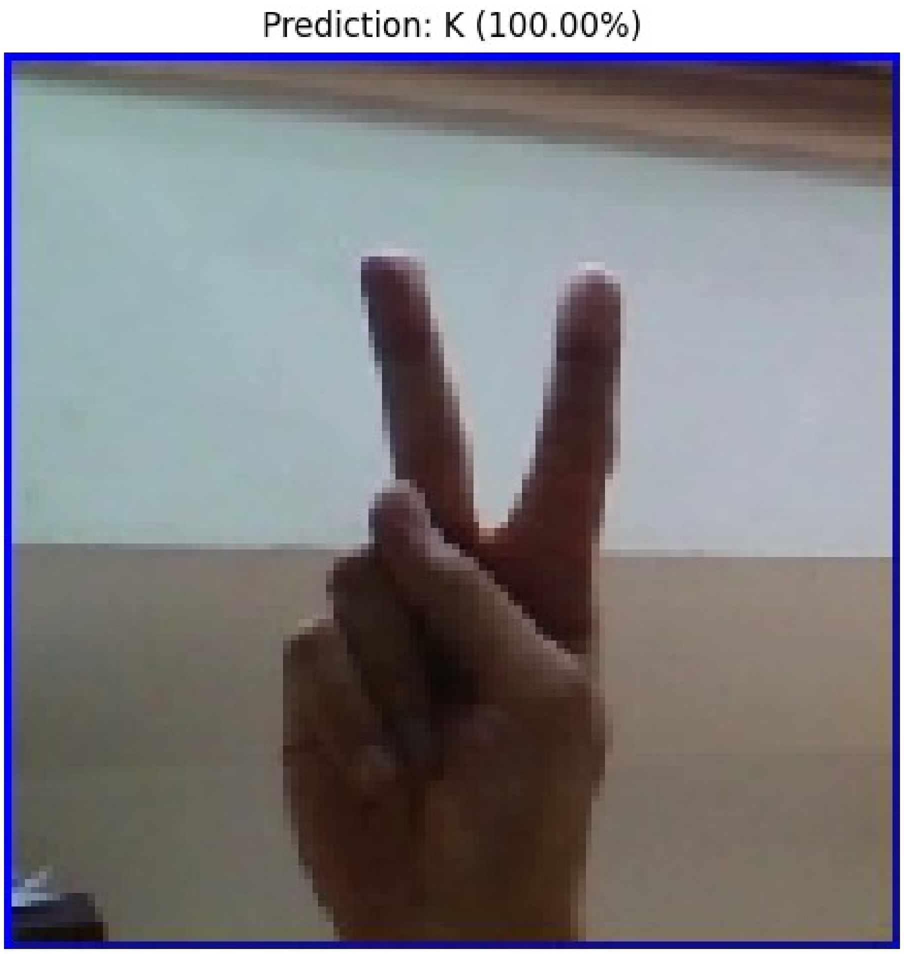
```
Configured to use CPU only.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 138ms/step

Predicted class: J
Confidence: 81.29%
```

Prediction: J (81.29%)



```
#Tester

tf.config.set_visible_devices([], 'GPU')
print("Configured to use CPU only.")

MODEL_PATH = 'ASL.keras'
IMAGE_PATH = "E:\\Projects\\asl_alphabet_test\\asl_alphabet_test\\
K_test.jpg"

IMAGE_SIZE = (128, 128)

CLASS_NAMES = sorted(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
'X', 'Y', 'Z', 'del', 'nothing', 'space'])

model = tf.keras.models.load_model(MODEL_PATH)

def preprocess_image(image_path, target_size):
    img = tf.keras.preprocessing.image.load_img(image_path,
target_size=target_size)
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    return img_array

test_image_array = preprocess_image(IMAGE_PATH, IMAGE_SIZE)

predictions = model.predict(test_image_array)

predicted_probabilities = predictions[0]
predicted_class_index = np.argmax(predicted_probabilities)
predicted_class_name = CLASS_NAMES[predicted_class_index]
confidence = predicted_probabilities[predicted_class_index] * 100

print(f"\nPredicted class: {predicted_class_name}")
print(f"Confidence: {confidence:.2f}%")

plt.figure(figsize=(6, 6))
display_img = tf.keras.preprocessing.image.load_img(IMAGE_PATH,
target_size=IMAGE_SIZE)
plt.imshow(display_img)
plt.title(f"Prediction: {predicted_class_name} ({confidence:.2f}%)")
plt.axis('off')
plt.show()
```

```
Configured to use CPU only.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 152ms/step

Predicted class: K
Confidence: 100.00%
```



```
#Tester

tf.config.set_visible_devices([], 'GPU')
print("Configured to use CPU only.")

MODEL_PATH = 'ASL.keras'
IMAGE_PATH = "E:\\Projects\\asl_alphabet_test\\asl_alphabet_test\\
V_test.jpg"

IMAGE_SIZE = (128, 128)

CLASS_NAMES = sorted(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
'X', 'Y', 'Z', 'del', 'nothing', 'space'])

model = tf.keras.models.load_model(MODEL_PATH)

def preprocess_image(image_path, target_size):
    img = tf.keras.preprocessing.image.load_img(image_path,
target_size=target_size)
    img_array = tf.keras.preprocessing.image.img_to_array(img)
```

```python
    img_array = np.expand_dims(img_array, axis=0)
    return img_array

test_image_array = preprocess_image(IMAGE_PATH, IMAGE_SIZE)

predictions = model.predict(test_image_array)

predicted_probabilities = predictions[0]
predicted_class_index = np.argmax(predicted_probabilities)
predicted_class_name = CLASS_NAMES[predicted_class_index]
confidence = predicted_probabilities[predicted_class_index] * 100

print(f"\nPredicted class: {predicted_class_name}")
print(f"Confidence: {confidence:.2f}%")

plt.figure(figsize=(6, 6))
display_img = tf.keras.preprocessing.image.load_img(IMAGE_PATH,
target_size=IMAGE_SIZE)
plt.imshow(display_img)
plt.title(f"Prediction: {predicted_class_name} ({confidence:.2f}%)")
plt.axis('off')
plt.show()
```

```
Configured to use CPU only.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 136ms/step

Predicted class: V
Confidence: 100.00%
```

```
#Tester

tf.config.set_visible_devices([], 'GPU')
print("Configured to use CPU only.")

MODEL_PATH = 'ASL.keras'
IMAGE_PATH = "E:\\Projects\\asl_alphabet_test\\asl_alphabet_test\\
Z_test.jpg"

IMAGE_SIZE = (128, 128)

CLASS_NAMES = sorted(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
'X', 'Y', 'Z', 'del', 'nothing', 'space'])

model = tf.keras.models.load_model(MODEL_PATH)

def preprocess_image(image_path, target_size):
    img = tf.keras.preprocessing.image.load_img(image_path,
target_size=target_size)
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    return img_array

test_image_array = preprocess_image(IMAGE_PATH, IMAGE_SIZE)

predictions = model.predict(test_image_array)

predicted_probabilities = predictions[0]
predicted_class_index = np.argmax(predicted_probabilities)
predicted_class_name = CLASS_NAMES[predicted_class_index]
confidence = predicted_probabilities[predicted_class_index] * 100

print(f"\nPredicted class: {predicted_class_name}")
print(f"Confidence: {confidence:.2f}%")

plt.figure(figsize=(6, 6))
display_img = tf.keras.preprocessing.image.load_img(IMAGE_PATH,
target_size=IMAGE_SIZE)
plt.imshow(display_img)
plt.title(f"Prediction: {predicted_class_name} ({confidence:.2f}%)")
plt.axis('off')
plt.show()
```

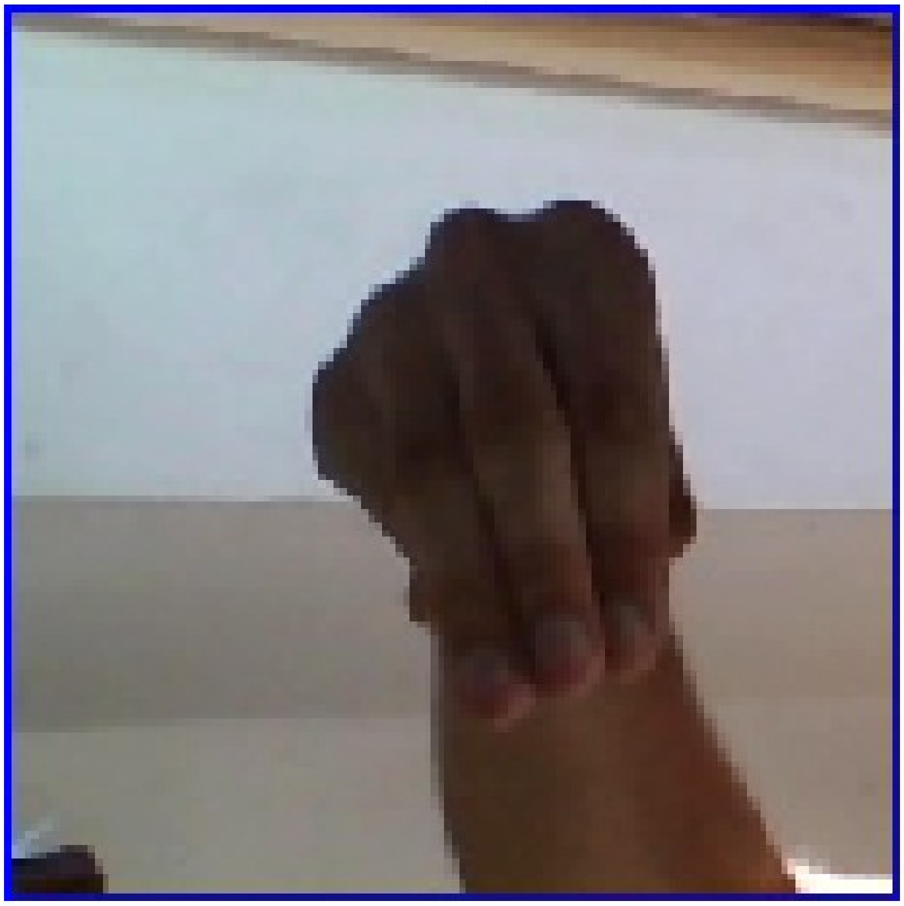
```
Configured to use CPU only.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step

Predicted class: Z
Confidence: 98.81%
```

Prediction: Z (98.81%)

```
#Tester

tf.config.set_visible_devices([], 'GPU')
print("Configured to use CPU only.")

MODEL_PATH = 'ASL.keras'
IMAGE_PATH = "E:\\Projects\\asl_alphabet_test\\asl_alphabet_test\\
M_test.jpg"

IMAGE_SIZE = (128, 128)

CLASS_NAMES = sorted(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
'X', 'Y', 'Z', 'del', 'nothing', 'space'])

model = tf.keras.models.load_model(MODEL_PATH)

def preprocess_image(image_path, target_size):
    img = tf.keras.preprocessing.image.load_img(image_path,
target_size=target_size)
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    return img_array

test_image_array = preprocess_image(IMAGE_PATH, IMAGE_SIZE)

predictions = model.predict(test_image_array)

predicted_probabilities = predictions[0]
predicted_class_index = np.argmax(predicted_probabilities)
predicted_class_name = CLASS_NAMES[predicted_class_index]
confidence = predicted_probabilities[predicted_class_index] * 100

print(f"\nPredicted class: {predicted_class_name}")
print(f"Confidence: {confidence:.2f}%")

plt.figure(figsize=(6, 6))
display_img = tf.keras.preprocessing.image.load_img(IMAGE_PATH,
target_size=IMAGE_SIZE)
plt.imshow(display_img)
plt.title(f"Prediction: {predicted_class_name} ({confidence:.2f}%)")
plt.axis('off')
plt.show()
```

```
Configured to use CPU only.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 138ms/step

Predicted class: M
Confidence: 100.00%
```

Prediction: M (100.00%)



```
#Tester

tf.config.set_visible_devices([], 'GPU')
print("Configured to use CPU only.")

MODEL_PATH = 'ASL.keras'
IMAGE_PATH = "E:\\Projects\\asl_alphabet_test\\asl_alphabet_test\\
N_test.jpg"

IMAGE_SIZE = (128, 128)

CLASS_NAMES = sorted(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
'X', 'Y', 'Z', 'del', 'nothing', 'space'])

model = tf.keras.models.load_model(MODEL_PATH)

def preprocess_image(image_path, target_size):
    img = tf.keras.preprocessing.image.load_img(image_path,
target_size=target_size)
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    return img_array

test_image_array = preprocess_image(IMAGE_PATH, IMAGE_SIZE)

predictions = model.predict(test_image_array)

predicted_probabilities = predictions[0]
predicted_class_index = np.argmax(predicted_probabilities)
predicted_class_name = CLASS_NAMES[predicted_class_index]
confidence = predicted_probabilities[predicted_class_index] * 100

print(f"\nPredicted class: {predicted_class_name}")
print(f"Confidence: {confidence:.2f}%")

plt.figure(figsize=(6, 6))
display_img = tf.keras.preprocessing.image.load_img(IMAGE_PATH,
target_size=IMAGE_SIZE)
plt.imshow(display_img)
plt.title(f"Prediction: {predicted_class_name} ({confidence:.2f}%)")
plt.axis('off')
plt.show()
```

```
Configured to use CPU only.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 137ms/step

Predicted class: N
Confidence: 100.00%
```

Prediction: N (100.00%)