In [3]:
```
import os
import numpy as np
import cv2
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
from gtts import gTTS
from IPython.display import Audio
import cv2
data_dir = r"C:\Users\pc\OneDrive\Desktop\project\newdata\train"
import os
# Image parameters
img_size = 64
labels = os.listdir(data_dir)
num_classes = len(labels)

# Label mapping
label_map = {label: idx for idx, label in enumerate(labels)}
print("Labels:", label_map)

# Load images
data = []
target = []
for label in labels:
    path = os.path.join(data_dir, label)
    images = os.listdir(path)[:200]  # limit to 200 images per class

    for img_name in images:
        img_path = os.path.join(path, img_name)
        img = cv2.imread(img_path)

        if img is None:
            print(f"Warning: Could not load image {img_path}, skipping...'
            continue  # skip images that fail to load

        img = cv2.resize(img, (img_size, img_size))
        data.append(img)
        target.append(label_map[label])


data = np.array(data) / 255.0
target = to_categorical(target, num_classes)

print("Data shape:", data.shape)
print("Target shape:", target.shape)
```

```
Labels: {'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6, 'H': 7, 'I':
'J': 9, 'K': 10, 'L': 11, 'M': 12, 'N': 13, 'O': 14, 'P': 15, 'Q': 16, 'R': 1
'S': 18, 'T': 19, 'U': 20, 'V': 21, 'W': 22, 'X': 23, 'Y': 24, 'Z': 25}
Data shape: (4154, 64, 64, 3)
Target shape: (4154, 26)
```

In [4]:
```
X_train, X_test, y_train, y_test = train_test_split(data, target, test_si
```

In [5]:
```
model = Sequential()

# CNN layers
model.add(Conv2D(32, (3,3), activation='relu', input_shape=(img_size, img_
model.add(MaxPooling2D((2,2)))

model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D((2,2)))

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

model.add(Flatten())

# Reshape for RNN
model.add(Reshape((1, -1)))

# RNN layer
model.add(LSTM(64))

# Fully connected
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 32)        896

 max_pooling2d (MaxPooling2D  (None, 31, 31, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 29, 29, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 14, 14, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 12, 12, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 6, 6, 128)        0
 2D)

 flatten (Flatten)           (None, 4608)              0

 reshape (Reshape)           (None, 1, 4608)           0

 lstm (LSTM)                 (None, 64)                1196288

 dense (Dense)               (None, 64)                4160

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 26)                1690

=================================================================
Total params: 1,295,386
Trainable params: 1,295,386
Non-trainable params: 0
_________________________________________________________________
```

In [6]:
```
#TRAIN the model
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validatio
```

```
Epoch 1/10
104/104 [==============================] - 15s 102ms/step - loss: 2.4884 -
accuracy: 0.3030 - val_loss: 0.6841 - val_accuracy: 0.8989
Epoch 2/10
104/104 [==============================] - 10s 95ms/step - loss: 0.5749 - acc
0.8727 - val_loss: 0.1048 - val_accuracy: 0.9832
Epoch 3/10
104/104 [==============================] - 10s 94ms/step - loss: 0.2332 - acc
0.9627 - val_loss: 0.0286 - val_accuracy: 0.9976
Epoch 4/10
104/104 [==============================] - 10s 93ms/step - loss: 0.1331 - acc
0.9813 - val_loss: 0.0088 - val_accuracy: 1.0000
Epoch 5/10
104/104 [==============================] - 9s 88ms/step - loss: 0.0926 - accu
0.9850 - val_loss: 0.0057 - val_accuracy: 1.0000
Epoch 6/10
104/104 [==============================] - 9s 89ms/step - loss: 0.0638 - accu
0.9901 - val_loss: 0.0026 - val_accuracy: 1.0000
Epoch 7/10
104/104 [==============================] - 11s 102ms/step - loss: 0.0454 -
accuracy: 0.9955 - val_loss: 0.0015 - val_accuracy: 1.0000
Epoch 8/10
104/104 [==============================] - 11s 103ms/step - loss: 0.0326 -
accuracy: 0.9961 - val_loss: 7.9211e-04 - val_accuracy: 1.0000
Epoch 9/10
104/104 [==============================] - 10s 92ms/step - loss: 0.0283 - acc
0.9964 - val_loss: 0.0010 - val_accuracy: 1.0000
Epoch 10/10
104/104 [==============================] - 11s 105ms/step - loss: 0.0239 -
accuracy: 0.9976 - val_loss: 4.7261e-04 - val_accuracy: 1.0000
```

In [7]:
```
#evaluate the model
loss, acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {acc * 100:.2f}%")
```

```
26/26 [==============================] - 1s 30ms/step - loss: 4.7261e-04 -
accuracy: 1.0000
Test Accuracy: 100.00%
```

In [8]:
```
#PREDICT AND CONVERT TO SPEECH
def predict_and_speak(img_path):
    img = cv2.imread(img_path)
    img = cv2.resize(img, (img_size, img_size))
    img = img / 255.0
    img = np.expand_dims(img, axis=0)

    pred = model.predict(img)
    class_idx = np.argmax(pred)
    for label, idx in label_map.items():
        if idx == class_idx:
            predicted_label = label
            break

    print(f"Predicted Sign: {predicted_label}")

    # Convert to speech
    tts = gTTS(text=f"The sign is {predicted_label}", lang='en')
    tts.save("output.mp3")
    return Audio("output.mp3")

# Example:
# predict_and_speak(r"C:\Users\pc\OneDrive\Desktop\Semester 6\dataset\asl_
```

In [9]:
```
#save the model
model.save(r"C:\Users\pc\OneDrive\Desktop\Semester 6\asl_model.h5")
```

In [10]:
```
# live detection
import cv2
import numpy as np
import tensorflow as tf
from gtts import gTTS
from IPython.display import Audio, display
import os
```

In [11]:
```
#load the trained model
# Load trained model
model = tf.keras.models.load_model(r"C:\Users\pc\OneDrive\Desktop\Semeste

# Label map (same as before)
label_map = {label: idx for idx, label in enumerate(os.listdir(r"C:\Users\
idx_map = {v: k for k, v in label_map.items()}

print("Model and labels loaded   )"
```

```
Model and labels loaded
```

In [15]:
```
#LIVE DETECTION USING WEBCAM
import uuid
import cv2
def speak_word(word):
    from gtts import gTTS
    import os
    tts = gTTS(text=f" {word}", lang='en')
    filename = f"{uuid.uuid4()}.mp3"
    tts.save(filename)
    os.system(f"start {filename}")  # For Windows
# Initialize webcam
cap = cv2.VideoCapture(0)
img_size = 64
last_predicted = ""

print("Press 'q' to quit")
detected_letters = []
import os

# Path to your training dataset folder containing subfolders for each cla
data_dir = r"C:\Users\pc\OneDrive\Desktop\project\newdata\train"

# Get the list of class labels (folder names)
labels = sorted(os.listdir(data_dir))  # Sort for consistency

# Map label names to integer indices
label_map = {label: idx for idx, label in enumerate(labels)}

# Reverse map from indices to label names
idx_map = {idx: label for label, idx in label_map.items()}

print("Label Map:", label_map)
print("Index Map:", idx_map)

while True:
    ret, frame = cap.read()
    if not ret:
        break

    frame = cv2.flip(frame, 1)

    x1, y1, x2, y2 = 100, 100, 300, 300
    cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 0, 0), 2)

    roi = frame[y1:y2, x1:x2]
    roi_resized = cv2.resize(roi, (img_size, img_size))
    roi_normalized = roi_resized / 255.0
    roi_expanded = np.expand_dims(roi_normalized, axis=0)

    pred = model.predict(roi_expanded)
    class_idx = np.argmax(pred)

   # Safely get label, fallback to "Unknown" if index not found
    predicted_label = idx_map.get(class_idx, "Unknown")

    print(f"Predicted class index: {class_idx}, label: {predicted_label}"

    # Show current letter
    cv2.putText(frame, f'Current: {predicted_label}', (10, 50),
                cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255,0), 2, cv2.LINE_AA)
```

```
Press 'q' to quit
Label Map: {'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6, 'H': 7, '
'J': 9, 'K': 10, 'L': 11, 'M': 12, 'N': 13, 'O': 14, 'P': 15, 'Q': 16, 'R': 1
'S': 18, 'T': 19, 'U': 20, 'V': 21, 'W': 22, 'X': 23, 'Y': 24, 'Z': 25}
Index Map: {0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G', 7: 'H', 8
9: 'J', 10: 'K', 11: 'L', 12: 'M', 13: 'N', 14: 'O', 15: 'P', 16: 'Q', 17: 'R
'S', 19: 'T', 20: 'U', 21: 'V', 22: 'W', 23: 'X', 24: 'Y', 25: 'Z'}
1/1 [==============================] - 0s 57ms/step
Predicted class index: 11, label: L
1/1 [==============================] - 0s 46ms/step
Predicted class index: 11, label: L
1/1 [==============================] - 0s 45ms/step
Predicted class index: 11, label: L
1/1 [==============================] - 0s 52ms/step
Predicted class index: 11, label: L
1/1 [==============================] - 0s 51ms/step
Predicted class index: 11, label: L
1/1 [==============================] - 0s 46ms/step
Predicted class index: 11, label: L
1/1 [==============================] - 0s 47ms/step
Predicted class index: 11, label: L
1/1 [==============================] - 0s 52ms/step
Predicted class index: 11, label: L
1/1 [==============================] - 0s 49ms/step
Predicted class index: 11, label: L
1/1 [==============================] - 0s 44ms/step
Predicted class index: 1, label: B
1/1 [==============================] - 0s 44ms/step
Predicted class index: 1, label: B
1/1 [==============================] - 0s 45ms/step
Predicted class index: 1, label: B
1/1 [==============================] - 0s 50ms/step
Predicted class index: 1, label: B
1/1 [==============================] - 0s 46ms/step
Predicted class index: 1, label: B
1/1 [==============================] - 0s 44ms/step
Predicted class index: 13, label: N
1/1 [==============================] - 0s 43ms/step
Predicted class index: 13, label: N
1/1 [==============================] - 0s 53ms/step
Predicted class index: 13, label: N
1/1 [==============================] - 0s 53ms/step
Predicted class index: 13, label: N
1/1 [==============================] - 0s 48ms/step
Predicted class index: 13, label: N
1/1 [==============================] - 0s 65ms/step
Predicted class index: 13, label: N
1/1 [==============================] - 0s 45ms/step
Predicted class index: 13, label: N
1/1 [==============================] - 0s 44ms/step
Predicted class index: 13, label: N
1/1 [==============================] - 0s 49ms/step
Predicted class index: 13, label: N
1/1 [==============================] - 0s 49ms/step
Predicted class index: 13, label: N
1/1 [==============================] - 0s 51ms/step
Predicted class index: 13, label: N
1/1 [==============================] - 0s 48ms/step
Predicted class index: 13, label: N
1/1 [==============================] - 0s 46ms/step
Predicted class index: 13, label: N
```

In [16]:
```
import uuid

def speak(predicted_label):
    filename = f"{uuid.uuid4()}.mp3"  # Generate unique filename
    tts = gTTS(text=f"The sign is {predicted_label}", lang='en')
    tts.save(filename)
    os.system(f"start {filename}")  # For Windows
```

In [ ]: