National College of Ireland

Configuration Manual for Deep Learning Approaches to Real-Time Sign Language Recognition and Multilingual Translation

MSc Research Project
Data Analytics

Harika Guttula
Student ID: x23237431

School of Computing
National College of Ireland

Supervisor: Dr. David Hamill

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Harika Guttula		
Student ID:	X23237431		
Programme:	Data Analytics	**Year:**	2024
Module:	MSc Research Project		
Supervisor:	David Hamill		
Submission Due Date:	12/12/2024		
Project Title:	Deep Learning Approaches to Real-Time Sign Language Recognition and Multilingual Translation		
Page Count:			
Word Count:			

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Harika Guttula

Date: 11-11-24

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual for Deep Learning Approaches to Real-Time Sign Language Recognition and Multilingual Translation

Harika Guttula
x23237431

1. Introduction

This manual guides you through the process of setting up the Sign Language Recognition system, which uses deep learning techniques (specifically Convolutional Neural Networks or CNN) for recognizing American Sign Language (ASL) gestures. The research study also integrates multilingual speech output for accessibility. The following sections provide detailed steps to configure and run the project on your local machine.

2. Prerequisites

Before you begin, ensure that the following software and hardware requirements are met:

Hardware Requirements:

- A computer with a minimum of 4GB of RAM (8GB recommended).
- A GPU (Graphics Processing Unit) is highly recommended for faster model training (e.g., NVIDIA GPU with CUDA support).

Software Requirements:

- Python (version 3.6 or above)
- pip (Python package installer)

Libraries and Tools:

- TensorFlow (for model training)
- Keras (high-level neural networks API, integrated with TensorFlow)
- OpenCV (for image processing and dataset manipulation)
- gTTS (Google Text-to-Speech for multilingual speech output)
- Matplotlib (for plotting training curves and visualization)
- NumPy (for numerical computations)
- Pandas (for dataset handling)
- Scikit-learn (for model evaluation metrics like confusion matrix)

3. Dataset Configuration

- The dataset used for training the model is an ASL dataset containing labeled images for each ASL sign (numbers 0-9, letters A-Z) is sourced from the kaggle[1].

- Dataset path: 'asl_dataset/asl_dataset/'
- The dataset is organized into folders, with each folder corresponding to a different class of ASL signs.

[1] https://www.kaggle.com/datasets/ayuraj/asl-dataset/data

```
asl_dataset/
    ├── 0/
    ├── 1/
    ├── 2/
    ├── A/
    ├── B/
    ├── ...
```

3.2 Preprocessing

- Image size is standardized to 224x224 pixels to fit the input requirements of pre-trained models like EfficientNetB3.
- Data is split into training, validation, and testing datasets with respective validation and test splits.

```
# Defining training dataset
train_image_ds = tf.keras.preprocessing.image_dataset_from_directory(
    train_data_dir,
    validation_split=validation_split,
    subset="training",
    seed=100,
    image_size=target_size,
    batch_size=batch_size,
)
```

✓ 0.2s

```
Found 2515 files belonging to 36 classes.
Using 2012 files for training.
```

```
# Defining Validation dataset
val_image_ds = tf.keras.preprocessing.image_dataset_from_directory(
    train_data_dir,
    validation_split=validation_split,
    subset="validation",
    seed=200,
    image_size=target_size,
    batch_size=batch_size,
)
```

✓ 0.1s

```
Found 2515 files belonging to 36 classes.
Using 503 files for validation.
```

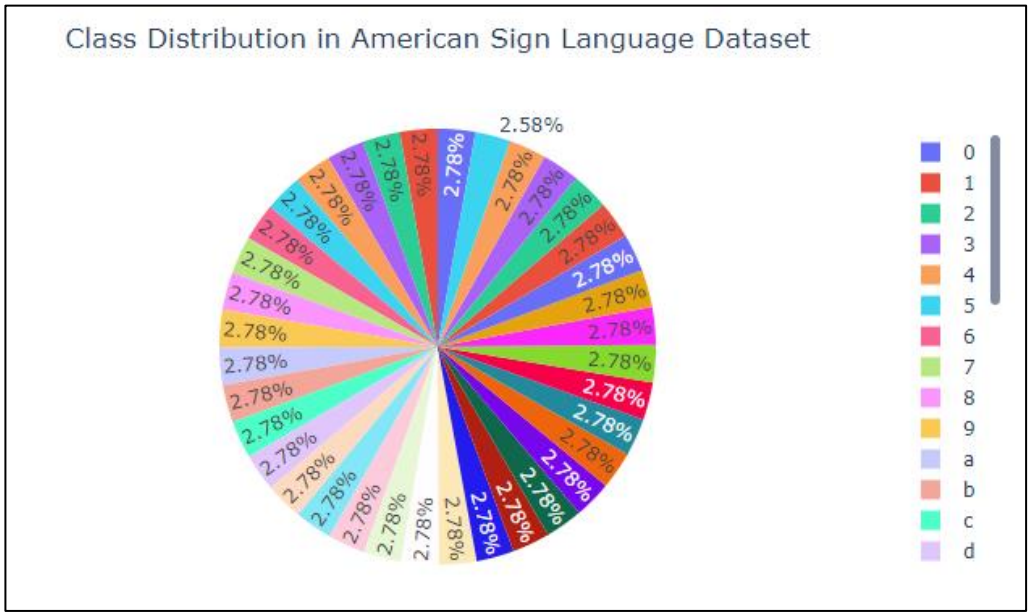
```
# Defining Testing dataset
test_image_ds = tf.keras.preprocessing.image_dataset_from_directory(
    train_data_dir,
    validation_split=testitng_split,
    subset= "validation",
    seed=400,
    image_size=target_size,
    batch_size=batch_size,
)
```

✓ 0.1s

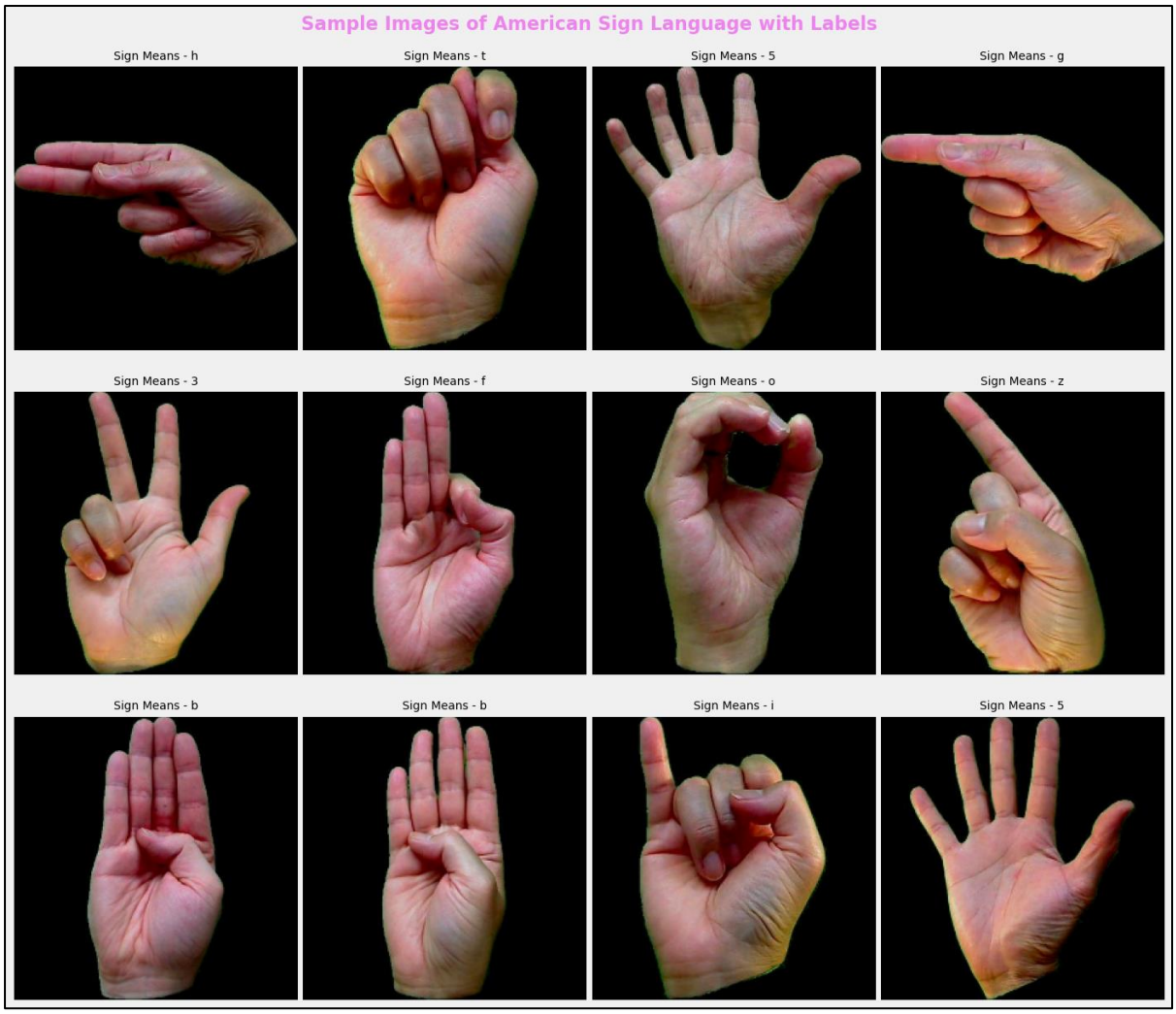
```
Found 2515 files belonging to 36 classes.
Using 251 files for validation.
```

3.3 Data Distribution

- Class distribution is visualized using pie charts and bar graphs, which help assess the dataset's balance across different sign categories.


Class Distribution in American Sign Language Dataset
2.58%
2.78%
0
1
2
3
4
5
6
7
8
9
a
b
c
d


Class Distribution with Mean
Counts
Class Names
Legend
Class Distribution
Mean
0 1 2 3 4 5 6 7 8 9 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



4. Model Architecture

4.1 CNN (EfficientNetB3)

EfficientNetB3 is used as a pre-trained base model for spatial feature extraction:

- Input Size: (224, 224, 3) — 224x224 RGB images.
- Layers: Conv2D, MaxPooling2D, BatchNormalization, Dropout (for regularization).
- Frozen Layers: The base model's layers are frozen during training to retain the learned weights from ImageNet.

4.2 RNN (LSTM)

- Input: Sequential frames of images representing ASL gestures.
- LSTM Layer: Captures temporal dependencies from sequences of images.
- Output Layer: A Dense layer with a softmax activation function, representing a classification for each gesture (36 classes: 0-9, A-Z).

```
# Load EfficientNetB3 as the base model for CNN (spatial feature extraction)
base_model = EfficientNetB3(input_shape=(224, 224, 3),
                            include_top=False,
                            weights='imagenet')

# Freeze the base model
base_model.trainable = False

#Construct the model
model = Sequential()
model.add(base_model)

# Flatten the features from each frame and apply dropout to prevent overfitting
model.add(TimeDistributed(Flatten()))
model.add(Dropout(0.5))

# Add an LSTM layer to capture the temporal dependencies (gesture sequence)
model.add(LSTM(128, return_sequences=False))  # You can adjust the number of units

# Add a Dense layer with Softmax activation for classification
model.add(Dense(36, activation='softmax'))  # Assuming 36 categories (0-9, A-Z)
```

4.3 Model Compilation

The model is compiled using the Adam optimizer with sparse categorical cross-entropy loss and accuracy as the metric:

```
# Set up the optimizer and loss function
model.compile(optimizer=Adam(), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

4.4 Early Stopping

Early stopping is implemented to prevent overfitting during training:

```
# Set up early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
```

5. Training the Model Code Implementation

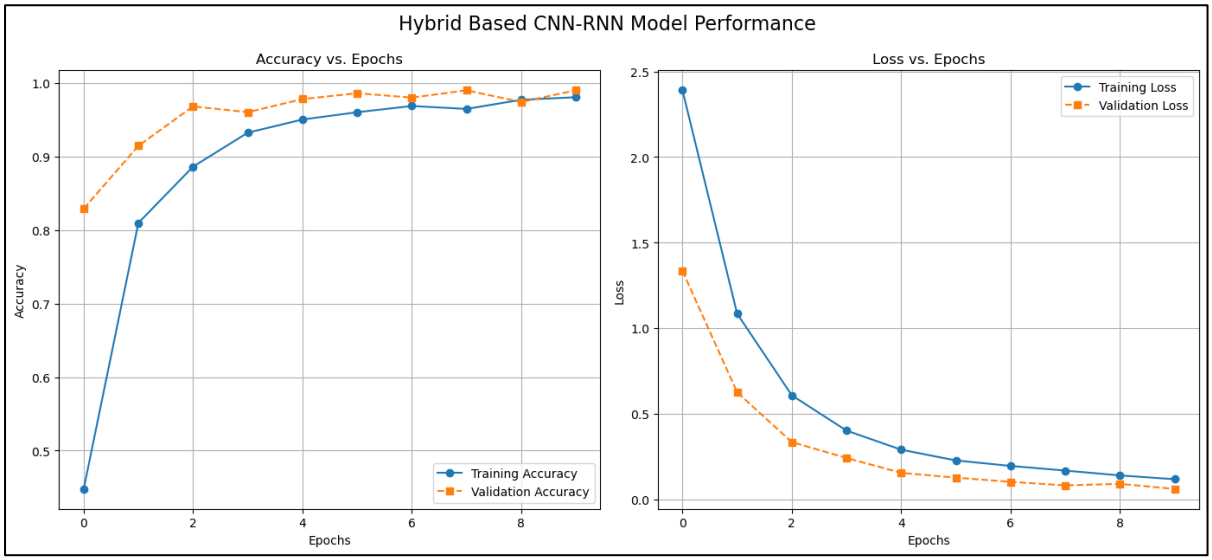
The model is trained on the training dataset and validated on the validation set for 10 epochs. The training accuracy and loss are monitored to adjust model performance.

```
# Fit the model
history = model.fit(
    train_image_ds,
    epochs=10,
    validation_data=val_image_ds,
    callbacks=[early_stopping]
)
```

```
Epoch 1/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 145s 2s/step - accuracy: 0.2710 - loss: 2.9534 - val_accuracy: 0.8290 - val_loss: 1.3338
Epoch 2/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 123s 2s/step - accuracy: 0.7781 - loss: 1.2561 - val_accuracy: 0.9145 - val_loss: 0.6242
Epoch 3/10
```

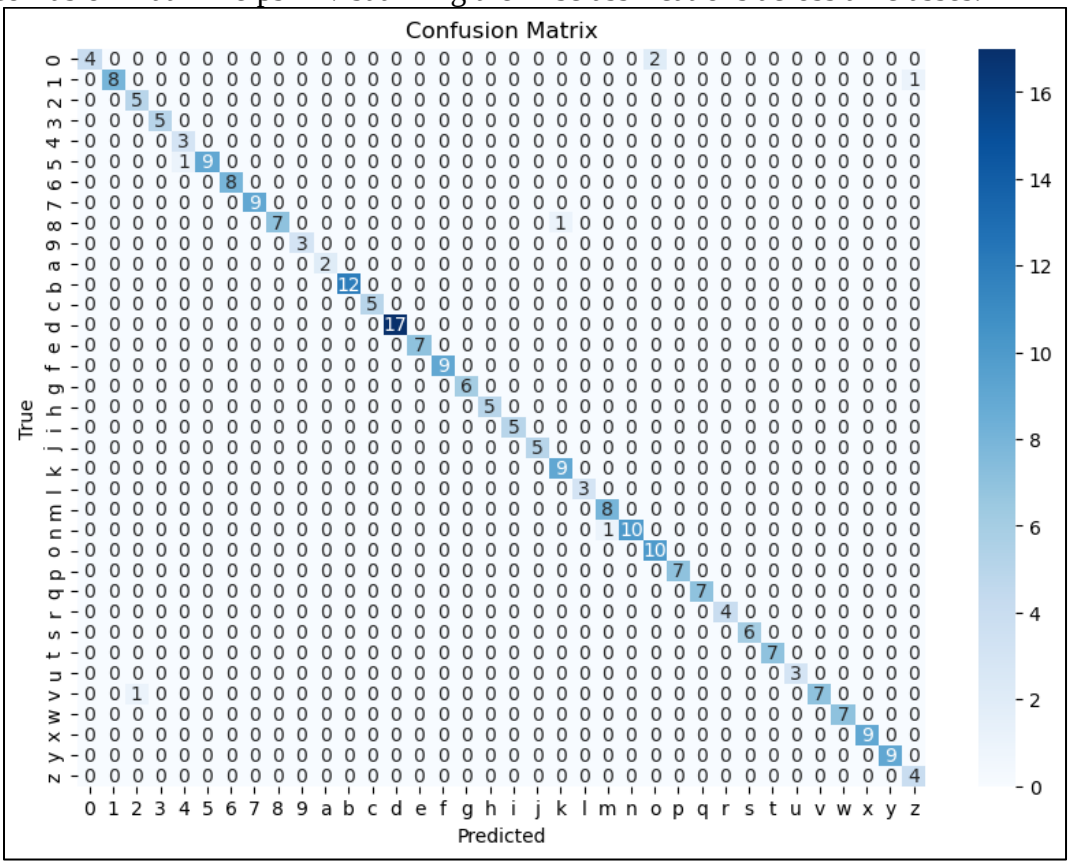
6. Performance Metrics

- Training & Validation Loss/Accuracy: Plot loss and accuracy curves.
- Test Accuracy: Evaluated on the test dataset, measuring how well the model generalizes to unseen data.
- Classification Report: Includes precision, recall, and F1-score for each class.



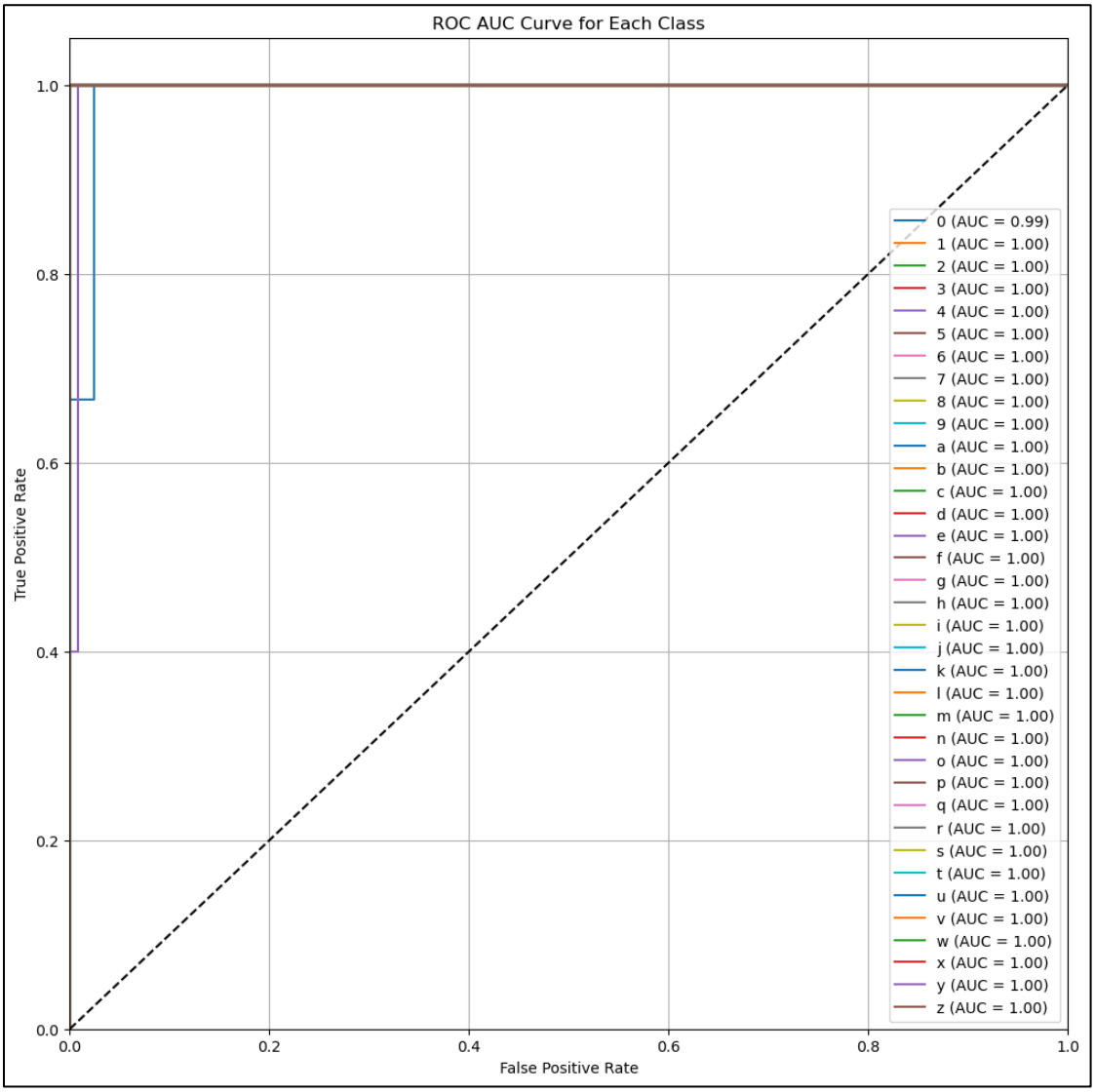
6.1 Confusion Matrix

The confusion matrix helps in visualizing the misclassifications across all classes.



6.2 ROC-AUC

For each class, the Receiver Operating Characteristic (ROC) curve is plotted to evaluate the model's ability to distinguish between classes:



7. Multilingual Translation (Text-to-Speech)

The model integrates with the Google Text-to-Speech (gTTS) API to translate recognized gestures into spoken text. Users can select the language for output speech (English, Spanish, French, etc.).

7.1 Speech Conversion Function

```
# Function to convert text to speech using Google Text-to-Speech (gTTS)
def text_to_speech(text, lang='en'):
    """
    Convert the recognized ASL text to speech using gTTS.

    Args:
    - text: Recognized text (either ASL character or phrase).
    - lang: Language code for the desired output language (default is English).
    """
    try:
        # Convert text to speech using gTTS
        tts = gTTS(text=text, lang=lang, slow=False)
        tts.save("output.mp3")

        # Play the audio file (ensure mpg321 or another player is installed)
        os.system("mpg321 output.mp3")
    except Exception as e:
        print(f"Error in Text-to-Speech conversion: {e}")
```

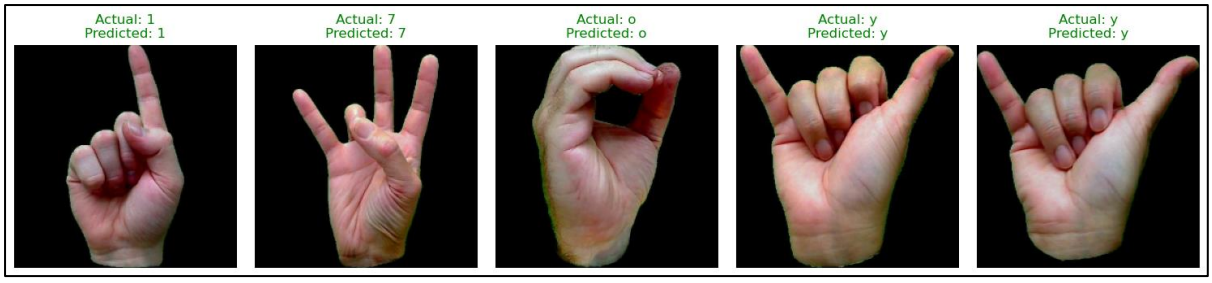
7.2 Multilingual Speech Output

Users can choose the language for speech output, and the ASL gesture predictions are translated into the selected language:

```python
# Function to select language for speech output (multilingual support)
def get_language_choice():
    """
    Prompt the user to select a language for speech output.
    Returns the language code for gTTS.
    """
    print("Select language for spoken output:")
    print("1. English (en)")
    print("2. Spanish (es)")
    print("3. French (fr)")
    print("4. German (de)")
    print("5. Italian (it)")
    print("6. Arabic (ar)")

    # Ask for user input and select the corresponding language code
    choice = input("Enter the number corresponding to the language: ")


    language_map = {
        '1': 'en',
        '2': 'es',
        '3': 'fr',
        '4': 'de',
        '5': 'it',
        '6': 'ar'
    }

    # Return the language code, defaulting to English ('en') if invalid choice
    return language_map.get(choice, 'en')
```

8. Visualization of Predictions

The system plots predicted signs along with their corresponding class names, highlighting the correctness of predictions using color-coded labels (green for correct, red for incorrect).



9. Audio Playback

The system can play the generated speech file using pygame:

```python
# Function to play the saved output file
def play_mp3(file_path):
    # Initialize the mixer module of pygame
    pygame.mixer.init()

    try:
        # Load the mp3 file
        pygame.mixer.music.load(file_path)

        # Play the music
        pygame.mixer.music.play()

        # Keep the program running while the music is playing
        while pygame.mixer.music.get_busy():
            time.sleep(1)  # Check every second if music is still playing

    finally:
        # Stop the music when it's finished or if an error occurs
        pygame.mixer.music.stop()

        # Clean up pygame mixer
        pygame.mixer.quit()

# play the saved output file
play_mp3('output.mp3')
```

Conclusion

This manual outlines the process of setting up and running a deep learning-based system for ASL recognition and multilingual translation. By leveraging hybrid CNN-RNN architecture, the system is capable of recognizing gestures in real-time and converting them into speech, allowing for multilingual accessibility.