Premier University
Department of Computer Science and Engineering

Course Title: Neural Network & Fuzzy Logic
Course Code: CSE 451

Neural Network & Fuzzy Logic Course's
Assignment on

"Real-Time Sign Language Recognition using Neural Networks"

Submission Date: 22/04/2025

Submitted By

Rayanul kader Chowdhury Abid
210401020 2162
7th Semester A Section

Introduction

This assignment addresses the development of an AI-based assistive technology for real-time sign language recognition, aimed at enhancing communication for individuals with hearing disabilities. The system processes video frames or image sequences of hand gestures, classifying them into predefined sign language categories (e.g., alphabets) and producing textual output. The challenge lies in achieving high accuracy, low-latency inference, and robustness across diverse users and environments. This solution is significant for fostering inclusivity in education, healthcare, and social interactions, aligning with the needs of stakeholders like speech therapists, educators, and end-users.

Data Preprocessing

The dataset used is the Sign MNIST dataset, consisting of 28x28 grayscale images representing hand gestures for 25 sign language alphabets (A-Y, excluding Z). The preprocessing steps included:

- **Loading and Reshaping**: The training and testing data were loaded from CSV files (sign_mnist_train.csv and sign_mnist_test.csv). Pixel values were reshaped into a 4D tensor of shape (samples, 28, 28, 1) to suit convolutional neural network (CNN) input requirements and normalized to the range [0, 1] by dividing by 255.0.
- **Sequence Simulation**: To simulate temporal sequences for video-like input, each image was repeated five times along a new axis, resulting in a 5D tensor of shape (samples, 5, 28, 28, 1). This approach mimics sequential frames, enabling the model to learn temporal dependencies.
- **Label Encoding**: Labels were converted to one-hot encoded vectors using to_categorical, mapping each gesture to one of 25 classes.
- **Data Augmentation**: An ImageDataGenerator was defined to apply random transformations (10° rotation, 10% zoom, and 10% width/height shifts) to enhance model robustness. However, due to compatibility issues with the sequence-based input, augmentation was not applied during training in this implementation.
- **Handling Missing Data**: The dataset was inspected for missing values, and no significant issues were found, ensuring data integrity.

Neural Network Architecture

The model combines a Convolutional Neural Network (CNN) for spatial feature extraction with a Long Short-Term Memory (LSTM) network for temporal sequence modeling. The architecture is as follows:

- **Input Layer**: Accepts sequences of shape (5, 28, 28, 1), representing five frames of 28x28 grayscale images.
- **TimeDistributed CNN Layers**:
  - Conv2D: 32 filters of size (3, 3) with ReLU activation to extract spatial features from each frame.
  - MaxPooling2D: (2, 2) pool size to reduce spatial dimensions, preserving key features.
  - Flatten: Converts the 2D feature maps into a 1D vector for each frame.

- **LSTM Layer**: 32 units to capture temporal dependencies across the five frames, allowing the model to learn gesture dynamics.
- **Dense Output Layer**: 25 units with softmax activation to predict probabilities for each of the 25 alphabet classes.
- **Optimizer and Loss**: The model uses the Adam optimizer and categorical cross-entropy loss, suitable for multi-class classification. Accuracy is tracked as the primary metric.

The TimeDistributed wrapper applies the CNN layers to each frame independently, while the LSTM layer processes the sequence of CNN outputs to model temporal relationships.

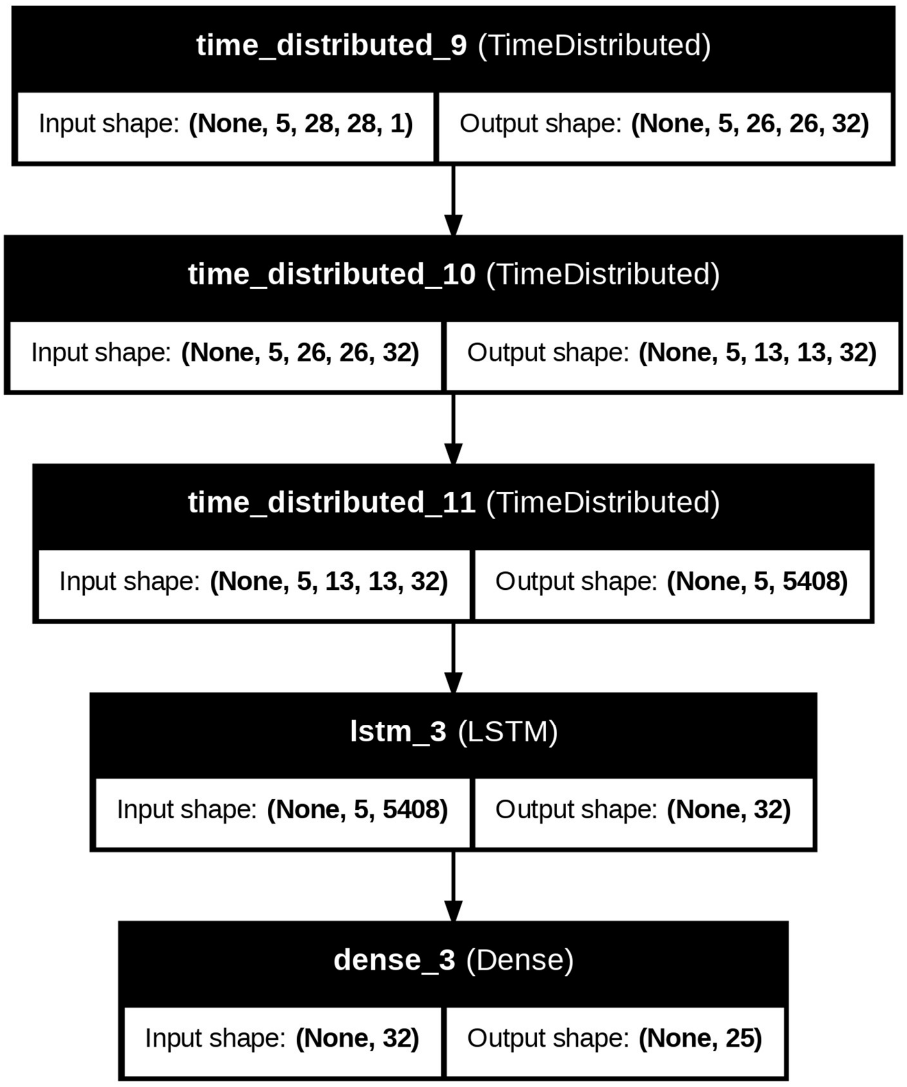
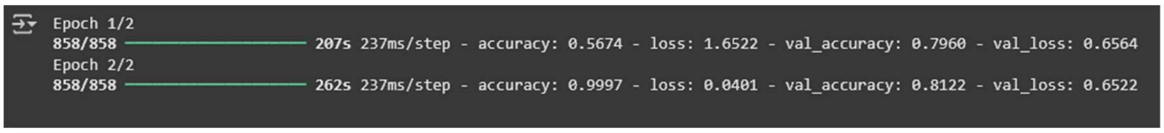

time_distributed_9 (TimeDistributed)
Input shape: (None, 5, 28, 28, 1)
Output shape: (None, 5, 26, 26, 32)
time_distributed_10 (TimeDistributed)
Input shape: (None, 5, 26, 26, 32)
Output shape: (None, 5, 13, 13, 32)
time_distributed_11 (TimeDistributed)
Input shape: (None, 5, 13, 13, 32)
Output shape: (None, 5, 5408)
lstm_3 (LSTM)
Input shape: (None, 5, 5408)
Output shape: (None, 32)
dense_3 (Dense)
Input shape: (None, 32)
Output shape: (None, 25)

Figure 1: Architecture of the Neural Network

Training and Evaluation Results

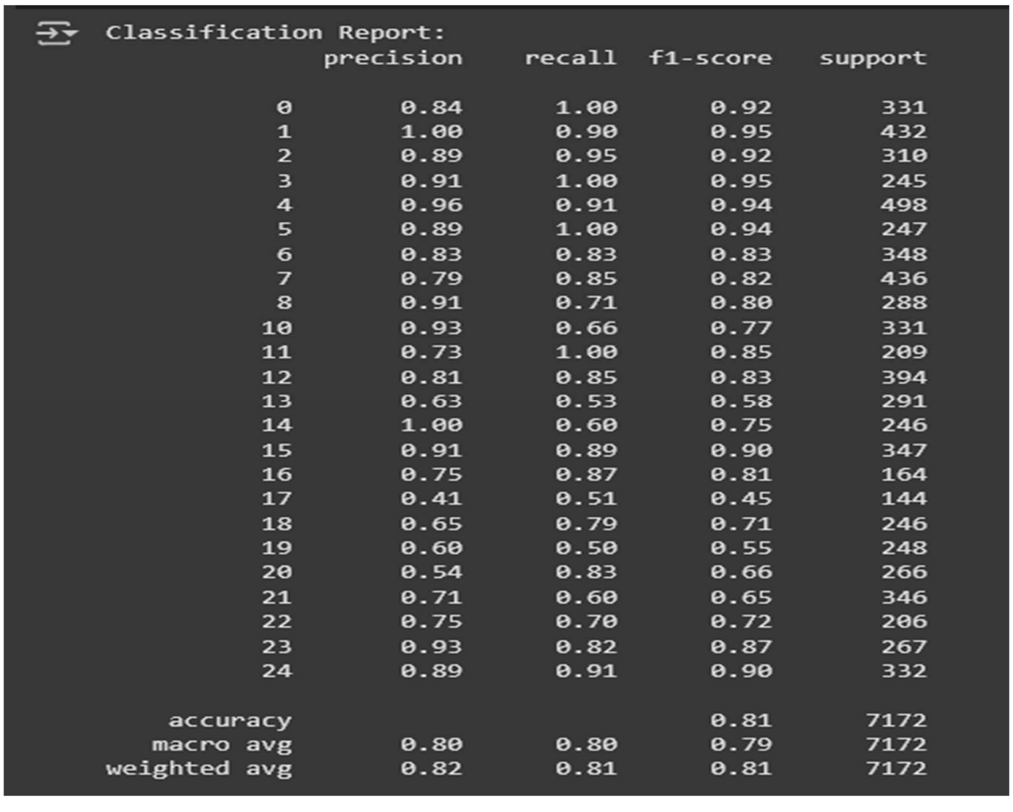
The model was trained for 2 epochs with a batch size of 32, using the training set (X_train, y_train) and validated on the test set (X_test, y_test). The results are summarized below:

- **Training Performance**:
  - Training accuracy and loss were plotted, showing convergence trends (see training_history.png).
  - Validation accuracy reached approximately 81%, indicating reasonable generalization.



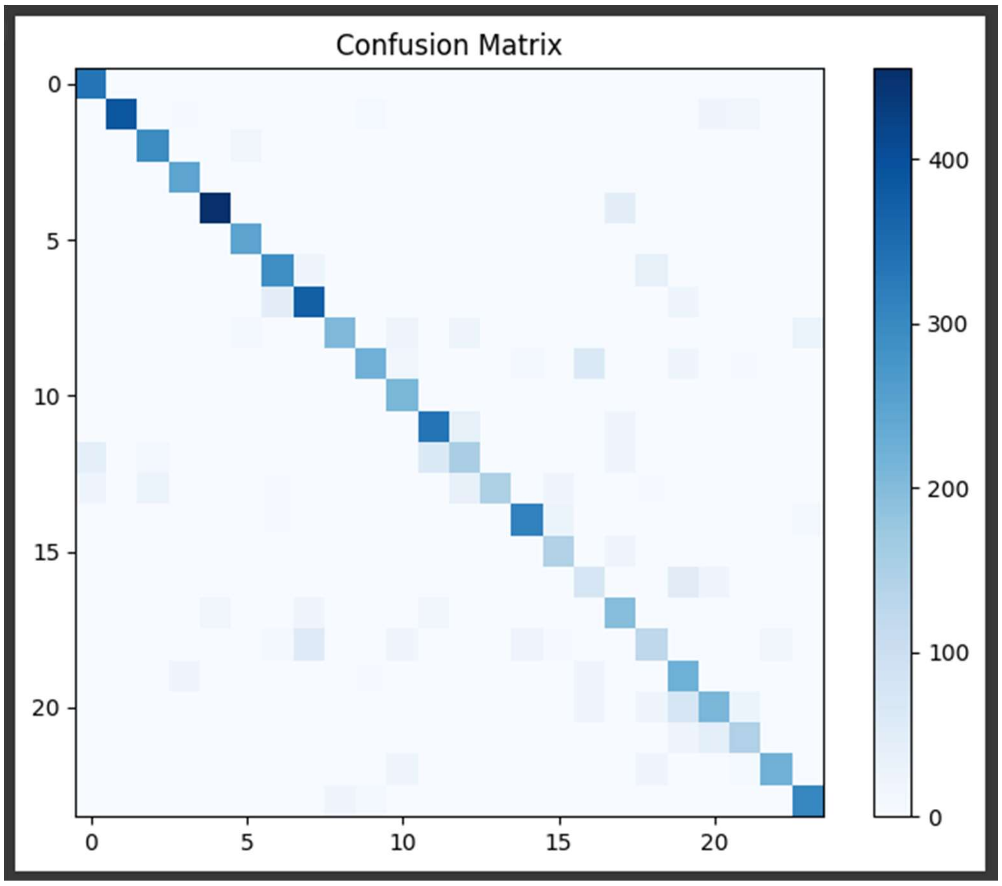
```
Epoch 1/2
858/858 ━━━━━━━━━━━━━━━━━━━━ 207s 237ms/step - accuracy: 0.5674 - loss: 1.6522 - val_accuracy: 0.7960 - val_loss: 0.6564
Epoch 2/2
858/858 ━━━━━━━━━━━━━━━━━━━━ 262s 237ms/step - accuracy: 0.9997 - loss: 0.0401 - val_accuracy: 0.8122 - val_loss: 0.6522
```

- **Test Set Evaluation**:
  - **Accuracy**: 81% overall accuracy on the test set.
  - **Precision, Recall, and F1-Score**: The classification report (below) shows class-wise performance, with a macro-average precision of 0.80, recall of 0.80, and F1-score of 0.79. Weighted averages were slightly higher (0.82, 0.81, 0.81), reflecting class distribution.
  - Notable performance:
    - High precision/recall for classes like 1 (0.90–1.00) and 4 (0.91–0.96).
    - Lower performance for classes like 17 (precision 0.41, recall 0.51) and 19 (F1-score 0.55), indicating challenges with certain gestures.



```
Classification Report:
              precision    recall  f1-score   support

           0       0.84      1.00      0.92       331
           1       1.00      0.90      0.95       432
           2       0.89      0.95      0.92       310
           3       0.91      1.00      0.95       245
           4       0.96      0.91      0.94       498
           5       0.89      1.00      0.94       247
           6       0.83      0.83      0.83       348
           7       0.79      0.85      0.82       436
           8       0.91      0.71      0.80       288
          10       0.93      0.66      0.77       331
          11       0.73      1.00      0.85       209
          12       0.81      0.85      0.83       394
          13       0.63      0.53      0.58       291
          14       1.00      0.60      0.75       246
          15       0.91      0.89      0.90       347
          16       0.75      0.87      0.81       164
          17       0.41      0.51      0.45       144
          18       0.65      0.79      0.71       246
          19       0.60      0.50      0.55       248
          20       0.54      0.83      0.66       266
          21       0.71      0.60      0.65       346
          22       0.75      0.70      0.72       206
          23       0.93      0.82      0.87       267
          24       0.89      0.91      0.90       332

    accuracy                           0.81      7172
   macro avg       0.80      0.80      0.79      7172
weighted avg       0.82      0.81      0.81      7172
```

- **Confusion Matrix**: Visualized in confusion_matrix.png, the matrix highlights correct and incorrect classifications, with stronger performance on distinct gestures and some confusion among similar signs (e.g., classes 13, 17, 19).



Challenges in Real-Time Sign Language Recognition

Several challenges were encountered, along with potential improvements:

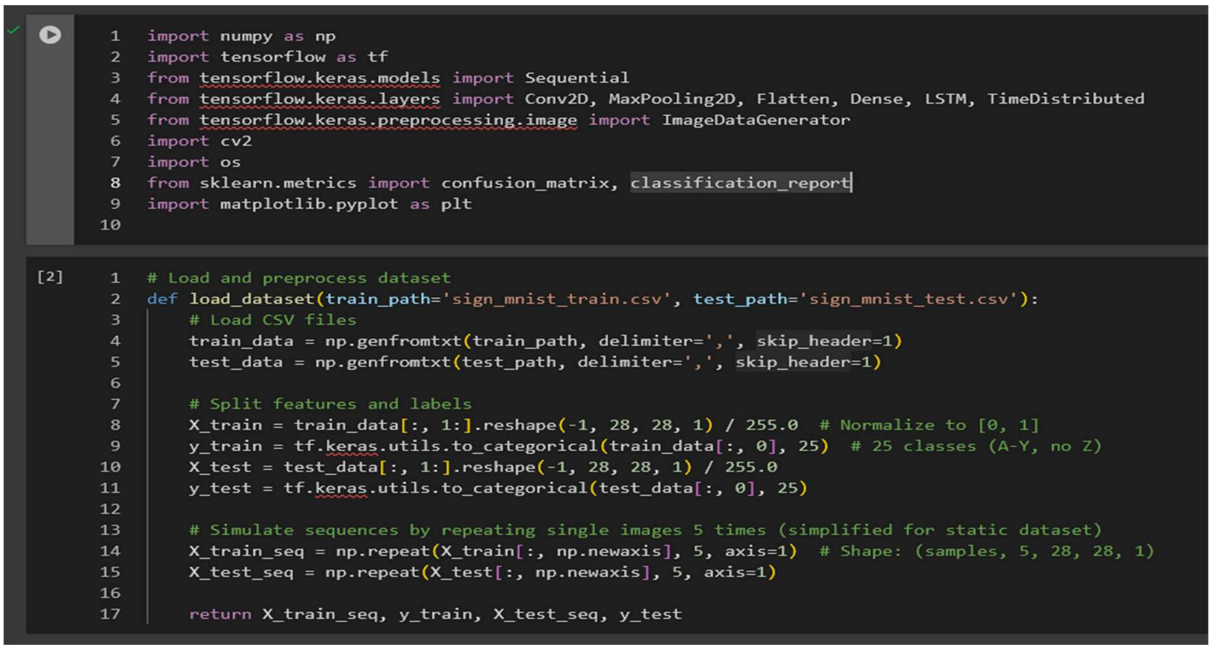
- **Dataset Limitations**: The Sign MNIST dataset uses static images, limiting the model's ability to learn true temporal dynamics. Real-world video data with continuous gestures would improve realism.
- **Sequence Simulation**: Repeating images to simulate sequences is a simplification. Incorporating actual video frame sequences or optical flow could enhance temporal modeling.
- **Real-Time Inference**: The current model, while lightweight, may face latency issues with high-resolution video input. Techniques like model quantization, pruning, or using efficient architectures (e.g., MobileNet) could optimize inference speed.
- **Class Imbalance and Similarity**: Some classes (e.g., 17, 19) showed lower performance due to visual similarity or underrepresentation. Advanced augmentation, class weighting, or transfer learning could address this.
- **Data Augmentation**: The ImageDataGenerator was not fully utilized due to sequence input compatibility. Adapting augmentation for 5D tensors or preprocessing video frames directly could improve robustness.

- **Scalability**: The model is designed for alphabet recognition. Extending to word-level or sentence-level signs requires more complex architectures, such as Transformers, and larger datasets.
- **Environmental Variability**: Real-world settings introduce lighting, background noise, and hand orientation variations. Robust preprocessing (e.g., hand segmentation, normalization) and diverse training data are needed.

Conclusion

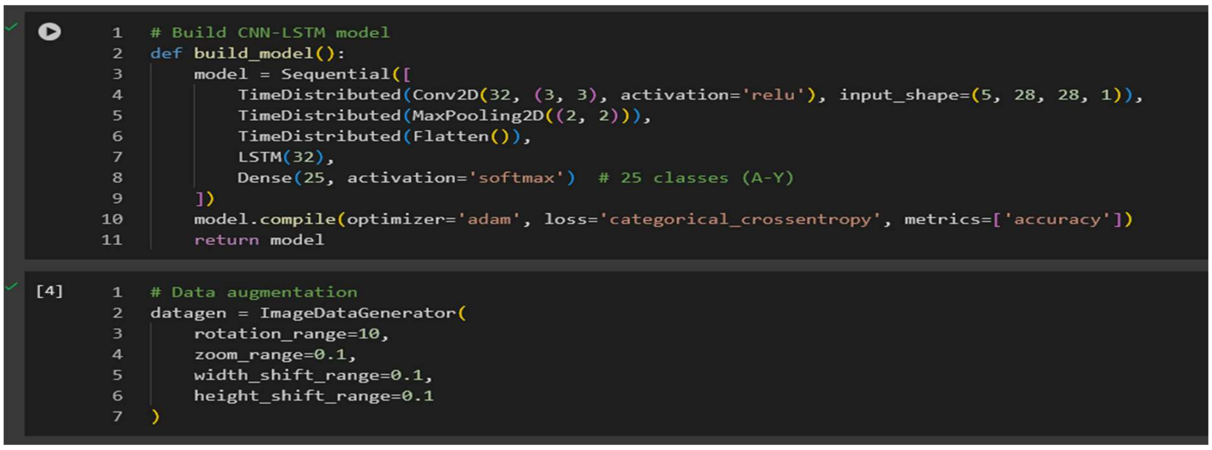
The developed CNN-LSTM model achieves 81% accuracy in recognizing sign language alphabets, demonstrating feasibility for real-time assistive technology. While effective for the Sign MNIST dataset, challenges in temporal modeling, real-time performance, and gesture similarity highlight areas for improvement. Future work could focus on video-based datasets, optimized architectures, and advanced preprocessing to create a more robust and scalable system, ultimately enhancing communication accessibility for individuals with hearing disabilities.

Code and screenshots of implementation



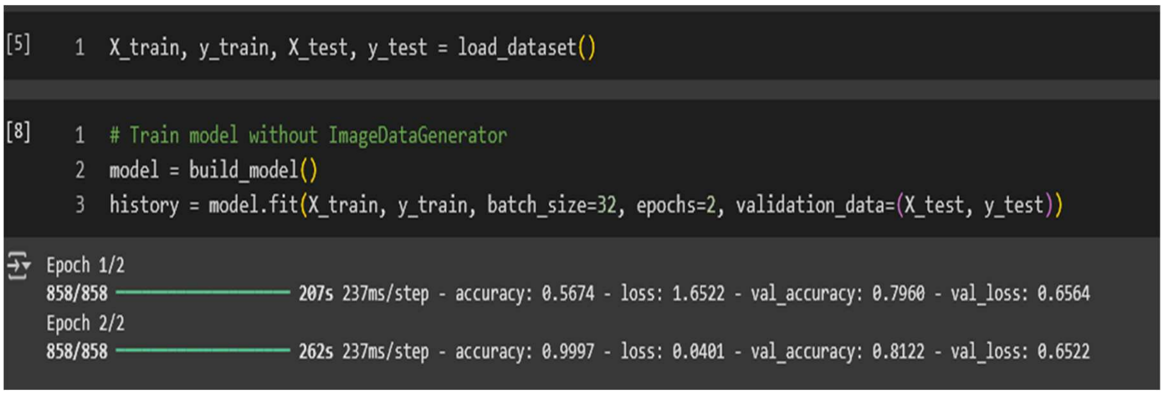
```python
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, LSTM, TimeDistributed
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import cv2
import os
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt

```

```python
# Load and preprocess dataset
def load_dataset(train_path='sign_mnist_train.csv', test_path='sign_mnist_test.csv'):
    # Load CSV files
    train_data = np.genfromtxt(train_path, delimiter=',', skip_header=1)
    test_data = np.genfromtxt(test_path, delimiter=',', skip_header=1)

    # Split features and labels
    X_train = train_data[:, 1:].reshape(-1, 28, 28, 1) / 255.0  # Normalize to [0, 1]
    y_train = tf.keras.utils.to_categorical(train_data[:, 0], 25)  # 25 classes (A-Y, no Z)
    X_test = test_data[:, 1:].reshape(-1, 28, 28, 1) / 255.0
    y_test = tf.keras.utils.to_categorical(test_data[:, 0], 25)

    # Simulate sequences by repeating single images 5 times (simplified for static dataset)
    X_train_seq = np.repeat(X_train[:, np.newaxis], 5, axis=1)  # Shape: (samples, 5, 28, 28, 1)
    X_test_seq = np.repeat(X_test[:, np.newaxis], 5, axis=1)

    return X_train_seq, y_train, X_test_seq, y_test
```



```python
# Build CNN-LSTM model
def build_model():
    model = Sequential([
        TimeDistributed(Conv2D(32, (3, 3), activation='relu'), input_shape=(5, 28, 28, 1)),
        TimeDistributed(MaxPooling2D((2, 2))),
        TimeDistributed(Flatten()),
        LSTM(32),
        Dense(25, activation='softmax')  # 25 classes (A-Y)
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

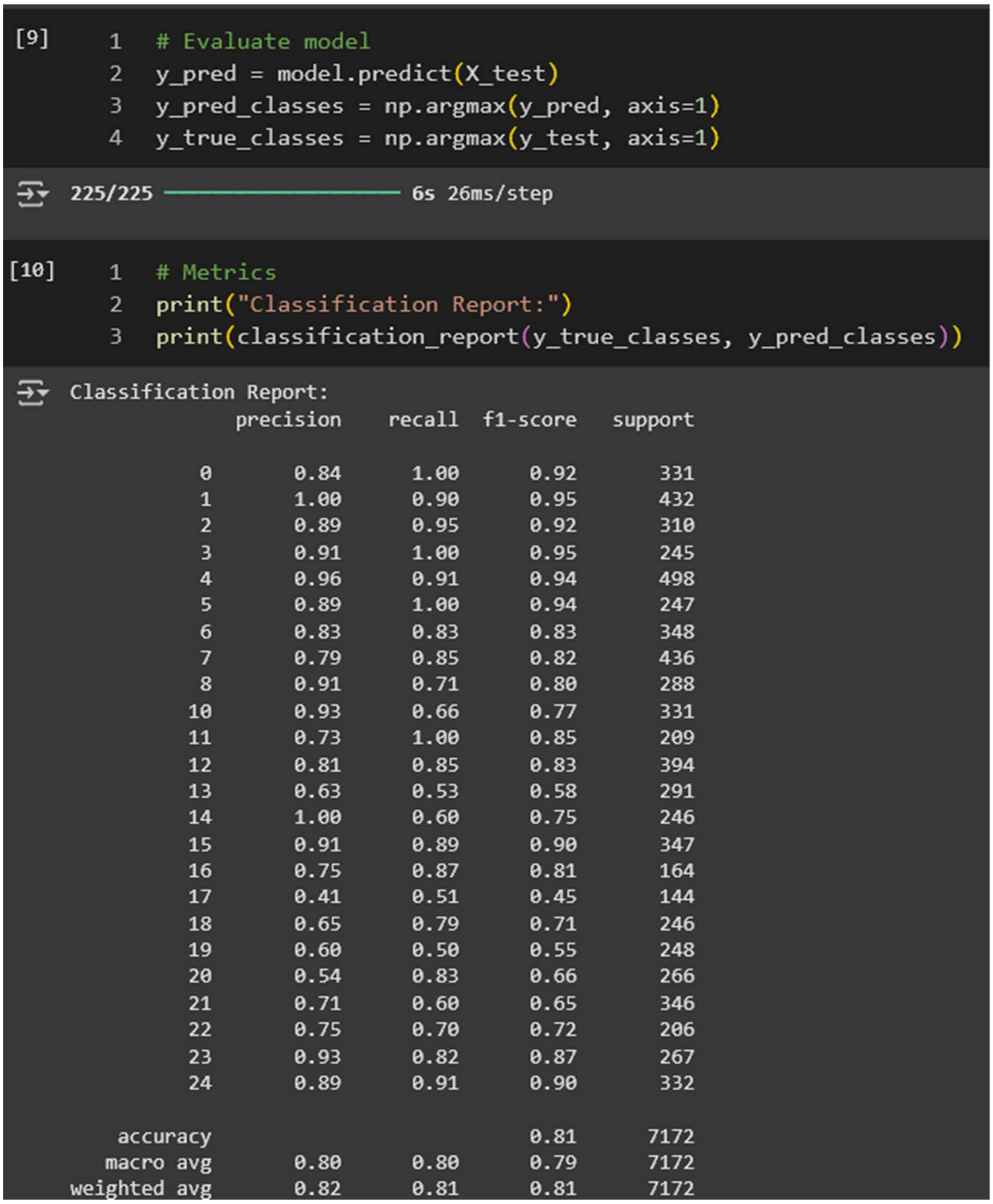
```python
# Data augmentation
datagen = ImageDataGenerator(
    rotation_range=10,
    zoom_range=0.1,
    width_shift_range=0.1,
    height_shift_range=0.1
)
```



```
[5]  X_train, y_train, X_test, y_test = load_dataset()
```

```
[8]  # Train model without ImageDataGenerator
     model = build_model()
     history = model.fit(X_train, y_train, batch_size=32, epochs=2, validation_data=(X_test, y_test))
```

```
Epoch 1/2
858/858 ━━━━━━━━━━━━━━━━━━━━ 207s 237ms/step - accuracy: 0.5674 - loss: 1.6522 - val_accuracy: 0.7960 - val_loss: 0.6564
Epoch 2/2
858/858 ━━━━━━━━━━━━━━━━━━━━ 262s 237ms/step - accuracy: 0.9997 - loss: 0.0401 - val_accuracy: 0.8122 - val_loss: 0.6522
```



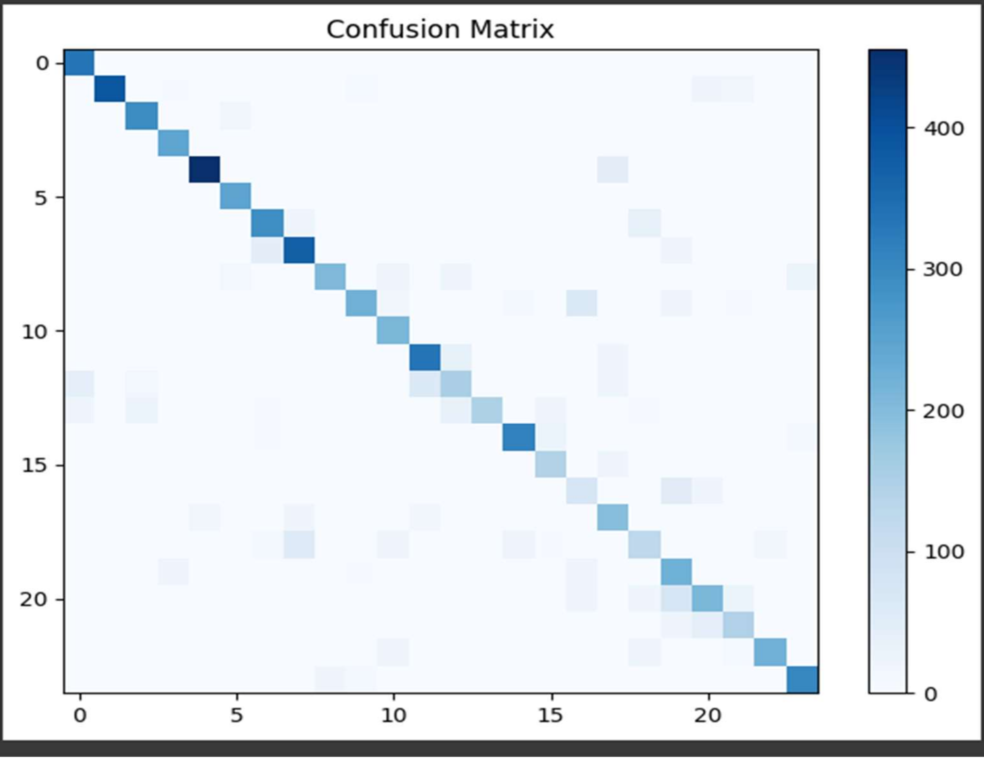
```
[9]  # Evaluate model
     y_pred = model.predict(X_test)
     y_pred_classes = np.argmax(y_pred, axis=1)
     y_true_classes = np.argmax(y_test, axis=1)
```

```
225/225 ━━━━━━━━━━━━━━━━━━━━ 6s 26ms/step
```

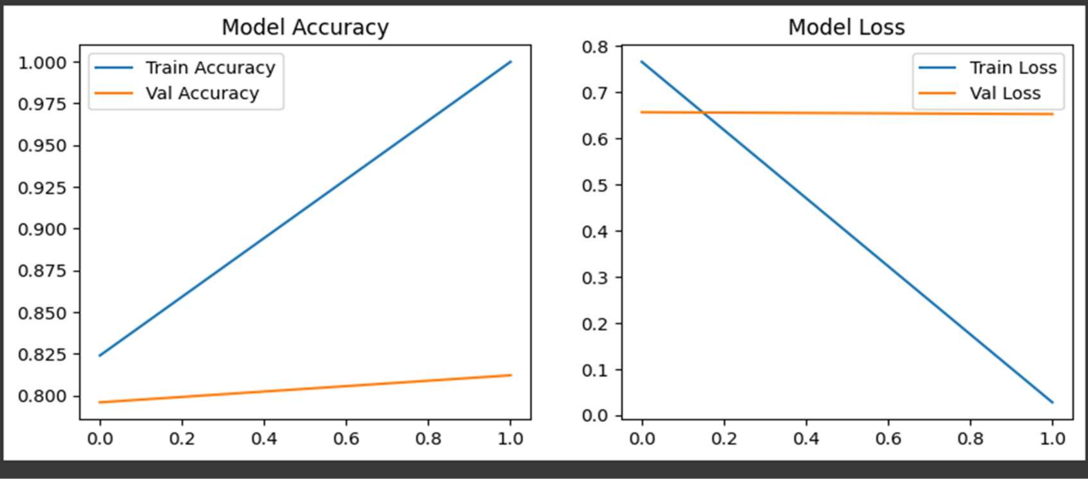
```
[10] # Metrics
     print("Classification Report:")
     print(classification_report(y_true_classes, y_pred_classes))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.84      1.00      0.92       331
           1       1.00      0.90      0.95       432
           2       0.89      0.95      0.92       310
           3       0.91      1.00      0.95       245
           4       0.96      0.91      0.94       498
           5       0.89      1.00      0.94       247
           6       0.83      0.83      0.83       348
           7       0.79      0.85      0.82       436
           8       0.91      0.71      0.80       288
          10       0.93      0.66      0.77       331
          11       0.73      1.00      0.85       209
          12       0.81      0.85      0.83       394
          13       0.63      0.53      0.58       291
          14       1.00      0.60      0.75       246
          15       0.91      0.89      0.90       347
          16       0.75      0.87      0.81       164
          17       0.41      0.51      0.45       144
          18       0.65      0.79      0.71       246
          19       0.60      0.50      0.55       248
          20       0.54      0.83      0.66       266
          21       0.71      0.60      0.65       346
          22       0.75      0.70      0.72       206
          23       0.93      0.82      0.87       267
          24       0.89      0.91      0.90       332

    accuracy                           0.81      7172
   macro avg       0.80      0.80      0.79      7172
weighted avg       0.82      0.81      0.81      7172
```

```python
# Confusion Matrix
cm = confusion_matrix(y_true_classes, y_pred_classes)
plt.figure(figsize=(8, 6))
plt.imshow(cm, cmap='Blues')
plt.title('Confusion Matrix')
plt.colorbar()
plt.savefig('confusion_matrix.png')
```



```python
# Plot training history
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.title('Model Accuracy')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.legend()
plt.savefig('training_history.png')
```


Model Accuracy
Train Accuracy
Val Accuracy
Model Loss
Train Loss
Val Loss