TestTrain

May 9, 2025

[1]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
from torch.utils.data import Dataset, DataLoader
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import os
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import json
import matplotlib.pyplot as plt
import gc
from tqdm import tqdm

# Constants
NUM_POSE_LANDMARKS = 19
NUM_HAND_LANDMARKS = 21
NUM_NODES = NUM_POSE_LANDMARKS + NUM_HAND_LANDMARKS*2  # Total nodes in the␣
 ↪graph
FEATURE_DIM = 3  # x, y coordinates + visibility

class GCNLayer(nn.Module):
    def __init__(self, in_features, out_features,dropout=0.5):
        super(GCNLayer, self).__init__()
        # self.linear = nn.Linear(in_features, out_features,bias=True)
        self.conv = nn.Conv1d(in_features, out_features, kernel_size=1,␣
 ↪bias=True)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x, adj):
        # x: (batch_size * seq_len, num_nodes, in_features)
        # adj: (num_nodes, num_nodes) - shared across all samples

        # Ensure adj is on the same device as x
        if adj.device != x.device:
            adj = adj.to(x.device)
```

```
        # Expand adj to match batch dimension
        batch_size_seq = x.size(0)
        adj_expanded = adj.unsqueeze(0).expand(batch_size_seq, -1, -1)

        # Graph convolution: first aggregate neighborhood features
        x = torch.bmm(adj_expanded, x)  # (batch_size * seq_len, num_nodes,␣
↪in_features)

        # For Conv1d: input needs to be (batch, channels, length)
        # So we permute from (batch, nodes, features) to (batch, features,␣
↪nodes)
        x = x.permute(0, 2, 1)  # -> (batch_size * seq_len, in_features,␣
↪num_nodes)

        # Apply convolution
        x = self.conv(x)  # (batch_size * seq_len, out_features, num_nodes)

        # Permute back to original format
        x = x.permute(0, 2, 1)  # -> (batch_size * seq_len, num_nodes,␣
↪out_features)

        x = F.gelu(x)
        x = self.dropout(x)
        return x

class GCNBiLSTM(nn.Module):
    def __init__(self, num_nodes=NUM_NODES, in_features=FEATURE_DIM,
                 gcn_hidden=64, lstm_hidden=128, num_classes=10,
                 num_gcn_layers=2, dropout=0.5, label_map=None):
        super(GCNBiLSTM, self).__init__()

        # Create multiple GCN layers
        self.gcn_layers = nn.ModuleList()
        self.gcn_layers.append(GCNLayer(in_features, gcn_hidden,dropout))

        for _ in range(num_gcn_layers - 1):
            self.gcn_layers.append(GCNLayer(gcn_hidden, gcn_hidden,dropout))

        # Bidirectional LSTM layer
        self.lstm = nn.LSTM(
            input_size=num_nodes * gcn_hidden,
            hidden_size=lstm_hidden,
            num_layers=2,
            batch_first=True,
            bidirectional=True,
            dropout=dropout if num_gcn_layers > 1 else 0
```

```
        )

        # Attention mechanism
        self.attention = nn.Sequential(
            nn.Linear(lstm_hidden * 2, 64),
            nn.Tanh(),
            nn.Linear(64, 1)
        )

        # Output classification layers
        self.classifier = nn.Sequential(
            nn.Linear(lstm_hidden * 2, lstm_hidden),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(lstm_hidden, num_classes)
        )

        self.dropout = nn.Dropout(dropout)
        self.label_map = label_map
        self.num_nodes = num_nodes
        self.gcn_hidden = gcn_hidden

    def forward(self, x, adj):
        # x shape: (batch_size, seq_len, num_nodes * in_features)
        # Reshape to (batch_size, seq_len, num_nodes, in_features)
        batch_size, seq_len, _ = x.size()
        x = x.view(batch_size, seq_len, self.num_nodes, -1)

        # Process each time step through GCN
        gcn_outputs = []
        for t in range(seq_len):
            # Get current time step data
            curr_x = x[:, t, :, :]  # (batch_size, num_nodes, in_features)

            # Process through GCN layers
            for gcn_layer in self.gcn_layers:
                curr_x = gcn_layer(curr_x, adj)
                curr_x = self.dropout(curr_x)

            # Flatten node features
            # curr_x = curr_x.view(batch_size, -1)  # (batch_size, num_nodes *␣
↪gcn_hidden)
            curr_x = curr_x.contiguous().view(batch_size, -1)
            gcn_outputs.append(curr_x)

        # Stack outputs to (batch_size, seq_len, num_nodes * gcn_hidden)
        gcn_out = torch.stack(gcn_outputs, dim=1)
```

```
        # Process through BiLSTM
        lstm_out, _ = self.lstm(gcn_out)  # (batch_size, seq_len, lstm_hidden *␣
↪2)

        # Apply attention mechanism
        attn_weights = self.attention(lstm_out).squeeze(-1)  # (batch_size,␣
↪seq_len)
        attn_weights = F.softmax(attn_weights, dim=1).unsqueeze(-1)  #␣
↪(batch_size, seq_len, 1)

        # Weighted sum of LSTM outputs
        context = torch.sum(lstm_out * attn_weights, dim=1)  # (batch_size,␣
↪lstm_hidden * 2)

        # Final classification
        output = self.classifier(context)

        return output

    def predict_label(self, x, adj):
        self.eval()
        with torch.no_grad():
            logits = self.forward(x, adj)  # Forward pass
            pred_classes = torch.argmax(logits, dim=1)  # Get the predicted␣
↪class (index)

            if self.label_map is not None:
                pred_labels = [self.label_map[int(idx)] for idx in pred_classes.
↪cpu().numpy()]
                return pred_labels
            else:
                return pred_classes



class GraphSequenceDataset(Dataset):
    def __init__(self, sequences, labels):
        self.sequences = torch.tensor(sequences, dtype=torch.float32)
        self.labels = torch.tensor(labels, dtype=torch.long)

    def __len__(self):
        return len(self.sequences)

    def __getitem__(self, idx):
        return self.sequences[idx], self.labels[idx]
```

```
def parse_frame(frame,):
    keypoints = []
    for part in ['pose', 'left_hand', 'right_hand']:
        for landmark in frame.get(part, []):
            keypoints.extend([landmark['x'], landmark['y'],␣
 ↪landmark['visibility']])
    return keypoints


def load_and_preprocess_data(data_dir, sequence_length=15):
    """
    Load and preprocess the JSON files into sequences.

    Args:
        data_dir: Directory containing the data
        sequence_length: Number of frames in each sequence

    Returns:
        sequences: array of shape (num_sequences, sequence_length, num_nodes *␣
 ↪features)
        sequence_labels: array of class labels
        label_encoder: fitted LabelEncoder
    """
    frame_data = []
    raw_labels = []

    # Step 1: Collect all labels
    for root, dirs, files in os.walk(data_dir):
        for file in files:
            if file.endswith(".json"):
                label = os.path.basename(os.path.dirname(os.path.join(root,␣
 ↪file)))
                raw_labels.append(label)

    # Step 2: Fit label encoder
    encoder = LabelEncoder()
    encoder.fit(raw_labels)
    label_map = {label: int(encoder.transform([label])[0]) for label in␣
 ↪set(raw_labels)}

    sequences = []
    sequence_labels = []

    # Step 3: Parse frames and assign encoded labels
    for root, dirs, files in os.walk(data_dir):
```

```
        for file in files:
            if file.endswith(".json"):
                label = os.path.basename(os.path.dirname(os.path.join(root,␣
 ↪file)))
                encoded_label = label_map[label]
                with open(os.path.join(root, file), 'r') as f:
                    frames = json.load(f)
                    features=[]
                    for frame in frames:
                        features.append(parse_frame(frame))
                        # frame_data.append([np.array(features), encoded_label])
                    sequences.append(np.stack(features))
                    sequence_labels.append(encoded_label)
    idx_to_label = {v: k for k, v in label_map.items()}
    label_map = idx_to_label
    del idx_to_label
    gc.collect()

    return np.array(sequences), np.array(sequence_labels), label_map


def create_adjacency_matrix():
    """Create the adjacency matrix for the graph."""

    pose_connections = [
        # Mouth
        (9,10),
        # Left Eyes
        (1,2),(2,3),(3,7),
        # Right Eyes
        (4,5),(5,6),(6,8),
        # Nose
        (0,4),(0,1),
        # Shoulders
        (11, 12),
        # Connect shoulders to hip
        (11, 17), (12, 18),
        # Connect hip points
        (17, 18),
        # Left arm
        (11, 13), (13, 15),
        # Right arm
        (12, 14), (14, 16)
    ]

    hand_connections = [
        # Thumb
```

```
        (0, 1), (1, 2), (2, 3), (3, 4),
        # Index finger
        (0, 5), (5, 6), (6, 7), (7, 8),
        # Middle finger
        (0, 9), (9, 10), (10, 11), (11, 12),
        # Ring finger
        (0, 13), (13, 14), (14, 15), (15, 16),
        # Pinky
        (0, 17), (17, 18), (18, 19), (19, 20),
        # Palm connections
        (5, 9), (9, 13), (13, 17)
    ]

    def create_adj_matrix(num_nodes, connections):
        adj_matrix = np.zeros((num_nodes, num_nodes))
        for i, j in connections:
            adj_matrix[i, j] = 1
            adj_matrix[j, i] = 1
        # Add self-loops
        for i in range(num_nodes):
            adj_matrix[i, i] = 1
        return adj_matrix

    pose_adj_matrix = create_adj_matrix(NUM_POSE_LANDMARKS, pose_connections)
    left_hand_adj_matrix = create_adj_matrix(NUM_HAND_LANDMARKS,␣
↪hand_connections)
    right_hand_adj_matrix = create_adj_matrix(NUM_HAND_LANDMARKS,␣
↪hand_connections)

    # Calculate the total number of nodes
    total_nodes = NUM_POSE_LANDMARKS + NUM_HAND_LANDMARKS + NUM_HAND_LANDMARKS

    # Initialize a global adjacency matrix
    global_adj_matrix = np.zeros((total_nodes, total_nodes))

    # start_pose = NUM_FACE_LANDMARKS
    start_pose=0
    end_pose = start_pose + NUM_POSE_LANDMARKS
    global_adj_matrix[start_pose:end_pose, start_pose:end_pose] =␣
↪pose_adj_matrix

    start_lh = end_pose
    end_lh = start_lh + NUM_HAND_LANDMARKS
    global_adj_matrix[start_lh:end_lh, start_lh:end_lh] = left_hand_adj_matrix

    start_rh = end_lh
    end_rh = start_rh + NUM_HAND_LANDMARKS
```

```
    global_adj_matrix[start_rh:end_rh, start_rh:end_rh] = right_hand_adj_matrix

    # Connect pose to hands
    pose_hand_connections = [
        (start_pose + 15, start_lh),  # Left hand wrist to left hand base
        (start_pose + 16, start_rh),  # Right hand wrist to right hand base
    ]
    for i, j in pose_hand_connections:
        global_adj_matrix[i, j] = 1
        global_adj_matrix[j, i] = 1

    # Normalize adjacency matrix (D^-0.5 * A * D^-0.5)
    # Add identity matrix to include self-connections
    adj_matrix = global_adj_matrix + np.eye(total_nodes)

    # Calculate degree matrix
    degree_matrix = np.diag(np.sum(adj_matrix, axis=1))

    # D^-0.5
    deg_inv_sqrt = np.linalg.inv(np.sqrt(degree_matrix))

    # Normalized adjacency matrix
    normalized_adj_matrix = deg_inv_sqrt @ adj_matrix @ deg_inv_sqrt

    return torch.FloatTensor(normalized_adj_matrix)


def train_model(model, train_loader, val_loader, adj_matrix, num_epochs=50,␣
 ↪lr=0.001,
                weight_decay=1e-5, patience=10, model_save_path='best_model.pt'):
    """
    Train the GCNBiLSTM model

    Args:
        model: GCNBiLSTM model
        train_loader: DataLoader for training data
        val_loader: DataLoader for validation data
        adj_matrix: Normalized adjacency matrix
        num_epochs: Number of training epochs
        lr: Learning rate
        weight_decay: Weight decay factor
        patience: Early stopping patience
        model_save_path: Path to save best model

    Returns:
        model: Trained model
        train_losses: List of training losses
```

```
        val_losses: List of validation losses
        train_accs: List of training accuracies
        val_accs: List of validation accuracies
    """
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    print(f"Using device: {device}")

    model = model.to(device)
    adj_matrix = adj_matrix.to(device)

    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=lr,␣
↪weight_decay=weight_decay)

    scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min',␣
↪patience=5, factor=0.5)


    train_losses = []
    val_losses = []
    train_accs = []
    val_accs = []

    best_val_loss = float('inf')
    early_stop_counter = 0

    for epoch in range(num_epochs):
        # Training phase
        model.train()
        train_loss = 0.0
        correct = 0
        total = 0

        progress_bar = tqdm(train_loader, desc=f"Epoch {epoch+1}/{num_epochs}␣
↪[Train]")
        for batch_idx, (inputs, targets) in enumerate(progress_bar):
            inputs, targets = inputs.to(device), targets.to(device)

            optimizer.zero_grad()
            outputs = model(inputs, adj_matrix)

            loss = criterion(outputs, targets)
            loss.backward()

            # Optional: gradient clipping
            torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=5.0)
```

```
            optimizer.step()

            train_loss += loss.item()
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += predicted.eq(targets).sum().item()

            progress_bar.set_postfix({
                'loss': train_loss/(batch_idx+1),
                'acc': 100.*correct/total
            })

        train_loss = train_loss / len(train_loader)
        train_acc = 100. * correct / total
        train_losses.append(train_loss)
        train_accs.append(train_acc)

        # Validation phase
        model.eval()
        val_loss = 0.0
        correct = 0
        total = 0

        with torch.no_grad():
            for batch_idx, (inputs, targets) in enumerate(val_loader):
                inputs, targets = inputs.to(device), targets.to(device)
                outputs = model(inputs, adj_matrix)

                loss = criterion(outputs, targets)
                val_loss += loss.item()

                _, predicted = outputs.max(1)
                total += targets.size(0)
                correct += predicted.eq(targets).sum().item()

        val_loss = val_loss / len(val_loader)
        val_acc = 100. * correct / total
        val_losses.append(val_loss)
        val_accs.append(val_acc)

        # Update learning rate
        scheduler.step(val_loss)

        print(f"Epoch {epoch+1}/{num_epochs} - "
              f"Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.2f}% - "
              f"Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.2f}%")
```

```
        # Save best model
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            # Save only the model state dict
            torch.save({'model_state_dict':model.state_dict(),'label_map':model.
 ↪label_map}, model_save_path)


            early_stop_counter = 0
            print(f"Saved best model to {model_save_path}")
        else:
            early_stop_counter += 1

        # Early stopping
        if early_stop_counter >= patience:
            print(f"Early stopping triggered after {epoch+1} epochs")
            break

    # Load best model
    checkpoint = torch.load(model_save_path)
    model.load_state_dict(checkpoint['model_state_dict'])
    model.label_map = checkpoint['label_map']
    return model, train_losses, val_losses, train_accs, val_accs


def evaluate_model(model, test_loader, adj_matrix):
    """
    Evaluate the model on test data

    Args:
        model: Trained GCNBiLSTM model
        test_loader: DataLoader for test data
        adj_matrix: Normalized adjacency matrix

    Returns:
        test_acc: Test accuracy
        predictions: Predicted labels
        true_labels: True labels
    """
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model = model.to(device)
    adj_matrix = adj_matrix.to(device)

    model.eval()
    correct = 0
    total = 0
```

```
    all_preds = []
    all_targets = []

    with torch.no_grad():
        for inputs, targets in test_loader:
            inputs, targets = inputs.to(device), targets.to(device)
            outputs = model(inputs, adj_matrix)

            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += predicted.eq(targets).sum().item()

            all_preds.extend(predicted.cpu().numpy())
            all_targets.extend(targets.cpu().numpy())

    test_acc = 100. * correct / total
    print(f"Test Accuracy: {test_acc:.2f}%")

    return test_acc, np.array(all_preds), np.array(all_targets)


def plot_results(train_losses, val_losses, train_accs, val_accs):
    """
    Plot training and validation metrics

    Args:
        train_losses: List of training losses
        val_losses: List of validation losses
        train_accs: List of training accuracies
        val_accs: List of validation accuracies
    """
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.plot(train_losses, label='Train Loss')
    plt.plot(val_losses, label='Val Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.title('Training and Validation Loss')

    plt.subplot(1, 2, 2)
    plt.plot(train_accs, label='Train Accuracy')
    plt.plot(val_accs, label='Val Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy (%)')
    plt.legend()
```

```
    plt.title('Training and Validation Accuracy')

    plt.tight_layout()
    plt.savefig('training_history.png')
    plt.show()


def main():
    torch.serialization.add_safe_globals([LabelEncoder])
    # Set random seeds for reproducibility
    SEED = 42
    torch.manual_seed(SEED)
    np.random.seed(SEED)

    # 1. Load and preprocess data
    data_dir = "data"  # Update with your data directory
    test_dir = "test_data"
    sequences, sequence_labels, label_map = load_and_preprocess_data(data_dir)

    print(f"Loaded {len(sequences)} sequences with shape {sequences.shape}")
    print(f'Sequence Label {len(sequence_labels)}')
    print(f"Number of classes: {len(label_map)}")

    # 2. Create adjacency matrix

    adj_matrix = create_adjacency_matrix()
    print(f'Unique Label : {len(np.unique(sequence_labels))}')
    X_train,X_val,y_train,y_val = train_test_split(
        sequences, sequence_labels, test_size=0.4, random_state=SEED,␣
 ↪stratify=sequence_labels
    )
    print(f'Unique Train : {len(np.unique(y_train))}')
    print(f'Unique Val : {len(np.unique(y_val))}')
    X_val,X_test,y_val,y_test = train_test_split(
        X_val, y_val, test_size=0.5, random_state=SEED, stratify=y_val
    )
    print(f'Unique Train : {len(np.unique(y_train))}')
    print(f'Unique Test : {len(np.unique(y_test))}')
    # X_test,y_test, _ = load_and_preprocess_data(test_dir)


    # 4. Create datasets and dataloaders
    train_dataset = GraphSequenceDataset(X_train, y_train)
    val_dataset = GraphSequenceDataset(X_val, y_val)
    test_dataset = GraphSequenceDataset(X_test, y_test)

    train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
```

```
    val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
    test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

    # 5. Create and train the model
    num_classes = len(label_map)
    model = GCNBiLSTM(
        num_nodes=NUM_NODES,
        in_features=FEATURE_DIM,
        gcn_hidden=256,
        lstm_hidden=512,
        num_classes=num_classes,
        num_gcn_layers=2,
        dropout=0.3,
        label_map=label_map
    )

    # 6. Train the model
    trained_model, train_losses, val_losses, train_accs, val_accs = train_model(
        model, train_loader, val_loader, adj_matrix,
        num_epochs=100, lr=0.001, weight_decay=5e-4,
        patience=15, model_save_path='best_gcn_bilstm_model.pt'
    )

    # 7. Evaluate the model
    test_acc, predictions, true_labels = evaluate_model(trained_model,␣
↪test_loader, adj_matrix)

    # 8. Plot results
    plot_results(train_losses, val_losses, train_accs, val_accs)

    # 9. Print classification report


    print("\nClassification Report:")
    actual_classes = np.unique(true_labels)
    print(actual_classes)
    class_names = [label_map[int(idx)] for idx in actual_classes]
    print(classification_report(true_labels, predictions,␣
↪target_names=class_names))

    # 10. Plot confusion matrix
    plt.figure(figsize=(12, 10))
    cm = confusion_matrix(true_labels, predictions)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names,␣
↪yticklabels=class_names)
    plt.xlabel('Predicted')
    plt.ylabel('True')
```

```
    plt.title('Confusion Matrix')
    plt.tight_layout()
    plt.savefig('confusion_matrix.png')
    plt.show()
```

[2]:
```
main()
```

```
Loaded 5643 sequences with shape (5643, 3, 183)
Sequence Label 5643
Number of classes: 51
Unique Label : 51
Unique Train : 51
Unique Val : 51
Unique Train : 51
Unique Test : 51
Using device: cuda

Epoch 1/100 [Train]: 100%|      | 106/106 [00:04<00:00, 21.63it/s,
loss=3.24, acc=8.39]

Epoch 1/100 - Train Loss: 3.2447, Train Acc: 8.39% - Val Loss: 2.9506, Val Acc:
10.89%
Saved best model to best_gcn_bilstm_model.pt

Epoch 2/100 [Train]: 100%|      | 106/106 [00:04<00:00, 24.73it/s,
loss=2.94, acc=11.8]

Epoch 2/100 - Train Loss: 2.9434, Train Acc: 11.82% - Val Loss: 2.8766, Val Acc:
14.08%
Saved best model to best_gcn_bilstm_model.pt

Epoch 3/100 [Train]: 100%|      | 106/106 [00:04<00:00, 25.03it/s,
loss=2.87, acc=13.3]

Epoch 3/100 - Train Loss: 2.8734, Train Acc: 13.32% - Val Loss: 2.9031, Val Acc:
11.60%

Epoch 4/100 [Train]: 100%|      | 106/106 [00:04<00:00, 24.63it/s, loss=2.8,
acc=15.1]

Epoch 4/100 - Train Loss: 2.7979, Train Acc: 15.07% - Val Loss: 2.7328, Val Acc:
16.21%
Saved best model to best_gcn_bilstm_model.pt

Epoch 5/100 [Train]: 100%|      | 106/106 [00:04<00:00, 24.47it/s,
loss=2.61, acc=20.1]

Epoch 5/100 - Train Loss: 2.6057, Train Acc: 20.12% - Val Loss: 2.4283, Val Acc:
26.13%
Saved best model to best_gcn_bilstm_model.pt

Epoch 6/100 [Train]: 100%|      | 106/106 [00:04<00:00, 24.76it/s,
loss=2.45, acc=26]
```

```
Epoch 6/100 - Train Loss: 2.4539, Train Acc: 25.97% - Val Loss: 2.3292, Val Acc:
30.65%
Saved best model to best_gcn_bilstm_model.pt

Epoch 7/100 [Train]: 100%|     | 106/106 [00:04<00:00, 24.74it/s,
loss=2.29, acc=30]

Epoch 7/100 - Train Loss: 2.2889, Train Acc: 29.99% - Val Loss: 2.0924, Val Acc:
34.01%
Saved best model to best_gcn_bilstm_model.pt

Epoch 8/100 [Train]: 100%|     | 106/106 [00:04<00:00, 23.79it/s,
loss=2.08, acc=36.4]

Epoch 8/100 - Train Loss: 2.0797, Train Acc: 36.37% - Val Loss: 1.8916, Val Acc:
40.39%
Saved best model to best_gcn_bilstm_model.pt

Epoch 9/100 [Train]: 100%|     | 106/106 [00:04<00:00, 23.34it/s,
loss=1.92, acc=39.7]

Epoch 9/100 - Train Loss: 1.9168, Train Acc: 39.68% - Val Loss: 1.7903, Val Acc:
44.46%
Saved best model to best_gcn_bilstm_model.pt

Epoch 10/100 [Train]: 100%|     | 106/106 [00:04<00:00, 23.83it/s,
loss=1.75, acc=45.2]

Epoch 10/100 - Train Loss: 1.7506, Train Acc: 45.17% - Val Loss: 1.6826, Val
Acc: 48.18%
Saved best model to best_gcn_bilstm_model.pt

Epoch 11/100 [Train]: 100%|     | 106/106 [00:04<00:00, 26.01it/s,
loss=1.71, acc=45.5]

Epoch 11/100 - Train Loss: 1.7085, Train Acc: 45.49% - Val Loss: 1.7242, Val
Acc: 44.64%

Epoch 12/100 [Train]: 100%|     | 106/106 [00:04<00:00, 23.39it/s,
loss=1.62, acc=48.9]

Epoch 12/100 - Train Loss: 1.6178, Train Acc: 48.89% - Val Loss: 1.5700, Val
Acc: 51.02%
Saved best model to best_gcn_bilstm_model.pt

Epoch 13/100 [Train]: 100%|     | 106/106 [00:03<00:00, 27.28it/s,
loss=1.57, acc=49.7]

Epoch 13/100 - Train Loss: 1.5671, Train Acc: 49.72% - Val Loss: 1.4661, Val
Acc: 53.85%
Saved best model to best_gcn_bilstm_model.pt

Epoch 14/100 [Train]: 100%|     | 106/106 [00:03<00:00, 27.51it/s,
loss=1.51, acc=51.5]
```

```
Epoch 14/100 - Train Loss: 1.5106, Train Acc: 51.52% - Val Loss: 1.4410, Val
Acc: 55.80%
Saved best model to best_gcn_bilstm_model.pt

Epoch 15/100 [Train]: 100%|     | 106/106 [00:03<00:00, 26.63it/s,
loss=1.45, acc=52.8]

Epoch 15/100 - Train Loss: 1.4452, Train Acc: 52.82% - Val Loss: 1.4643, Val
Acc: 54.30%

Epoch 16/100 [Train]: 100%|     | 106/106 [00:03<00:00, 27.25it/s,
loss=1.38, acc=55]

Epoch 16/100 - Train Loss: 1.3826, Train Acc: 54.98% - Val Loss: 1.4156, Val
Acc: 57.40%
Saved best model to best_gcn_bilstm_model.pt

Epoch 17/100 [Train]: 100%|     | 106/106 [00:03<00:00, 28.18it/s,
loss=1.36, acc=56.6]

Epoch 17/100 - Train Loss: 1.3581, Train Acc: 56.60% - Val Loss: 1.3688, Val
Acc: 57.66%
Saved best model to best_gcn_bilstm_model.pt

Epoch 18/100 [Train]: 100%|     | 106/106 [00:03<00:00, 27.73it/s,
loss=1.31, acc=56.8]

Epoch 18/100 - Train Loss: 1.3122, Train Acc: 56.81% - Val Loss: 1.3383, Val
Acc: 58.81%
Saved best model to best_gcn_bilstm_model.pt

Epoch 19/100 [Train]: 100%|     | 106/106 [00:03<00:00, 28.10it/s,
loss=1.26, acc=58.8]

Epoch 19/100 - Train Loss: 1.2594, Train Acc: 58.79% - Val Loss: 1.3111, Val
Acc: 60.76%
Saved best model to best_gcn_bilstm_model.pt

Epoch 20/100 [Train]: 100%|     | 106/106 [00:03<00:00, 26.97it/s,
loss=1.22, acc=60]

Epoch 20/100 - Train Loss: 1.2156, Train Acc: 60.03% - Val Loss: 1.3677, Val
Acc: 58.90%

Epoch 21/100 [Train]: 100%|     | 106/106 [00:03<00:00, 27.15it/s,
loss=1.2, acc=60.2]

Epoch 21/100 - Train Loss: 1.2008, Train Acc: 60.21% - Val Loss: 1.3543, Val
Acc: 56.86%

Epoch 22/100 [Train]: 100%|     | 106/106 [00:03<00:00, 27.23it/s,
loss=1.17, acc=61.4]

Epoch 22/100 - Train Loss: 1.1728, Train Acc: 61.42% - Val Loss: 1.2542, Val
Acc: 60.23%
Saved best model to best_gcn_bilstm_model.pt
```

```
Epoch 23/100 [Train]: 100%|     | 106/106 [00:03<00:00, 27.74it/s,
loss=1.13, acc=62.6]

Epoch 23/100 - Train Loss: 1.1329, Train Acc: 62.60% - Val Loss: 1.2982, Val
Acc: 61.65%

Epoch 24/100 [Train]: 100%|     | 106/106 [00:03<00:00, 28.62it/s,
loss=1.08, acc=64.7]

Epoch 24/100 - Train Loss: 1.0817, Train Acc: 64.73% - Val Loss: 1.1734, Val
Acc: 64.48%
Saved best model to best_gcn_bilstm_model.pt

Epoch 25/100 [Train]: 100%|     | 106/106 [00:03<00:00, 28.17it/s,
loss=1.08, acc=64.1]

Epoch 25/100 - Train Loss: 1.0775, Train Acc: 64.11% - Val Loss: 1.1945, Val
Acc: 64.57%

Epoch 26/100 [Train]: 100%|     | 106/106 [00:03<00:00, 26.94it/s,
loss=1.06, acc=63.7]

Epoch 26/100 - Train Loss: 1.0634, Train Acc: 63.69% - Val Loss: 1.1430, Val
Acc: 63.68%
Saved best model to best_gcn_bilstm_model.pt

Epoch 27/100 [Train]: 100%|     | 106/106 [00:03<00:00, 26.76it/s,
loss=1.05, acc=64.8]

Epoch 27/100 - Train Loss: 1.0518, Train Acc: 64.82% - Val Loss: 1.1720, Val
Acc: 64.48%

Epoch 28/100 [Train]: 100%|     | 106/106 [00:03<00:00, 27.54it/s,
loss=1.02, acc=66.1]

Epoch 28/100 - Train Loss: 1.0159, Train Acc: 66.14% - Val Loss: 1.1750, Val
Acc: 64.04%

Epoch 29/100 [Train]: 100%|     | 106/106 [00:03<00:00, 27.46it/s,
loss=0.992, acc=67.6]

Epoch 29/100 - Train Loss: 0.9924, Train Acc: 67.59% - Val Loss: 1.1648, Val
Acc: 66.52%

Epoch 30/100 [Train]: 100%|     | 106/106 [00:03<00:00, 27.89it/s,
loss=0.981, acc=67.4]

Epoch 30/100 - Train Loss: 0.9810, Train Acc: 67.39% - Val Loss: 1.1160, Val
Acc: 67.32%
Saved best model to best_gcn_bilstm_model.pt

Epoch 31/100 [Train]: 100%|     | 106/106 [00:03<00:00, 27.97it/s,
loss=0.965, acc=67.9]

Epoch 31/100 - Train Loss: 0.9653, Train Acc: 67.86% - Val Loss: 1.1381, Val
Acc: 65.63%
```

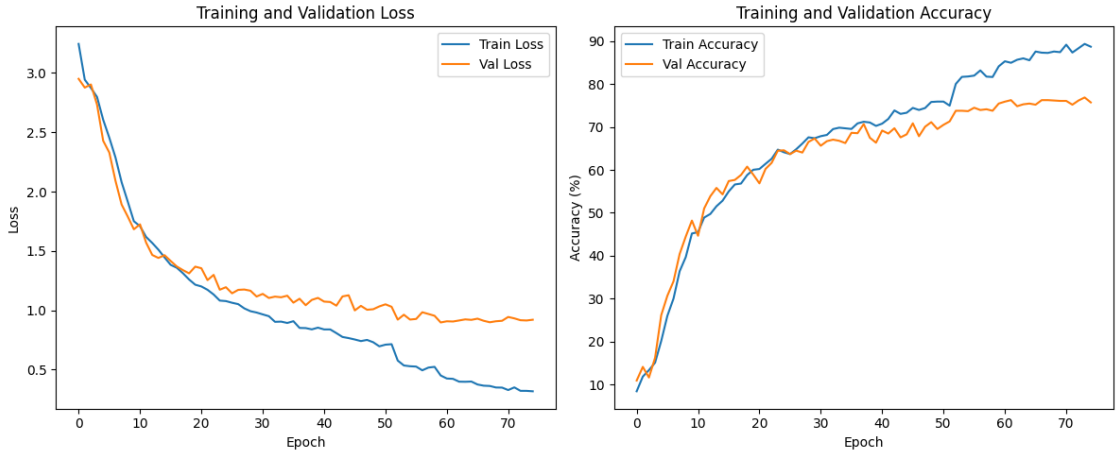
```
Epoch 32/100 [Train]: 100%|     | 106/106 [00:03<00:00, 27.20it/s,
loss=0.95, acc=68.2]

Epoch 32/100 - Train Loss: 0.9501, Train Acc: 68.15% - Val Loss: 1.1037, Val
Acc: 66.70%
Saved best model to best_gcn_bilstm_model.pt

Epoch 33/100 [Train]: 100%|     | 106/106 [00:03<00:00, 27.25it/s,
loss=0.903, acc=69.5]

Epoch 33/100 - Train Loss: 0.9027, Train Acc: 69.51% - Val Loss: 1.1151, Val
Acc: 67.05%

Epoch 34/100 [Train]: 100%|     | 106/106 [00:03<00:00, 27.69it/s,
loss=0.904, acc=69.8]

Epoch 34/100 - Train Loss: 0.9043, Train Acc: 69.84% - Val Loss: 1.1096, Val
Acc: 66.78%

Epoch 35/100 [Train]: 100%|     | 106/106 [00:03<00:00, 27.22it/s,
loss=0.893, acc=69.7]

Epoch 35/100 - Train Loss: 0.8932, Train Acc: 69.69% - Val Loss: 1.1229, Val
Acc: 66.25%

Epoch 36/100 [Train]: 100%|     | 106/106 [00:03<00:00, 27.14it/s,
loss=0.908, acc=69.5]

Epoch 36/100 - Train Loss: 0.9080, Train Acc: 69.54% - Val Loss: 1.0642, Val
Acc: 68.64%
Saved best model to best_gcn_bilstm_model.pt

Epoch 37/100 [Train]: 100%|     | 106/106 [00:03<00:00, 26.65it/s,
loss=0.851, acc=70.8]

Epoch 37/100 - Train Loss: 0.8512, Train Acc: 70.84% - Val Loss: 1.0972, Val
Acc: 68.56%

Epoch 38/100 [Train]: 100%|     | 106/106 [00:03<00:00, 27.79it/s,
loss=0.85, acc=71.2]

Epoch 38/100 - Train Loss: 0.8500, Train Acc: 71.23% - Val Loss: 1.0428, Val
Acc: 70.68%
Saved best model to best_gcn_bilstm_model.pt

Epoch 39/100 [Train]: 100%|     | 106/106 [00:03<00:00, 27.11it/s,
loss=0.839, acc=71]

Epoch 39/100 - Train Loss: 0.8392, Train Acc: 71.05% - Val Loss: 1.0875, Val
Acc: 67.49%

Epoch 40/100 [Train]: 100%|     | 106/106 [00:03<00:00, 27.40it/s,
loss=0.854, acc=70.3]

Epoch 40/100 - Train Loss: 0.8536, Train Acc: 70.25% - Val Loss: 1.1043, Val
Acc: 66.34%
```

```
Epoch 41/100 [Train]: 100%|      | 106/106 [00:03<00:00, 27.36it/s,
loss=0.839, acc=70.8]

Epoch 41/100 - Train Loss: 0.8386, Train Acc: 70.78% - Val Loss: 1.0732, Val
Acc: 69.18%

Epoch 42/100 [Train]: 100%|      | 106/106 [00:04<00:00, 22.10it/s,
loss=0.839, acc=71.9]

Epoch 42/100 - Train Loss: 0.8386, Train Acc: 71.88% - Val Loss: 1.0698, Val
Acc: 68.47%

Epoch 43/100 [Train]: 100%|      | 106/106 [00:04<00:00, 24.58it/s,
loss=0.808, acc=73.9]

Epoch 43/100 - Train Loss: 0.8076, Train Acc: 73.86% - Val Loss: 1.0390, Val
Acc: 69.71%
Saved best model to best_gcn_bilstm_model.pt

Epoch 44/100 [Train]: 100%|      | 106/106 [00:03<00:00, 27.68it/s,
loss=0.775, acc=73.1]

Epoch 44/100 - Train Loss: 0.7751, Train Acc: 73.06% - Val Loss: 1.1167, Val
Acc: 67.58%

Epoch 45/100 [Train]: 100%|      | 106/106 [00:03<00:00, 27.62it/s,
loss=0.766, acc=73.3]

Epoch 45/100 - Train Loss: 0.7655, Train Acc: 73.32% - Val Loss: 1.1265, Val
Acc: 68.29%

Epoch 46/100 [Train]: 100%|      | 106/106 [00:03<00:00, 28.05it/s,
loss=0.754, acc=74.4]

Epoch 46/100 - Train Loss: 0.7539, Train Acc: 74.45% - Val Loss: 0.9989, Val
Acc: 70.86%
Saved best model to best_gcn_bilstm_model.pt

Epoch 47/100 [Train]: 100%|      | 106/106 [00:04<00:00, 25.63it/s,
loss=0.74, acc=74]

Epoch 47/100 - Train Loss: 0.7404, Train Acc: 73.97% - Val Loss: 1.0373, Val
Acc: 67.85%

Epoch 48/100 [Train]: 100%|      | 106/106 [00:03<00:00, 27.64it/s,
loss=0.75, acc=74.4]

Epoch 48/100 - Train Loss: 0.7501, Train Acc: 74.42% - Val Loss: 1.0045, Val
Acc: 70.06%

Epoch 49/100 [Train]: 100%|      | 106/106 [00:03<00:00, 28.11it/s,
loss=0.731, acc=75.8]

Epoch 49/100 - Train Loss: 0.7307, Train Acc: 75.83% - Val Loss: 1.0088, Val
Acc: 71.12%
```

```
Epoch 50/100 [Train]: 100%|      | 106/106 [00:04<00:00, 26.37it/s,
loss=0.696, acc=75.9]

Epoch 50/100 - Train Loss: 0.6961, Train Acc: 75.92% - Val Loss: 1.0327, Val
Acc: 69.53%

Epoch 51/100 [Train]: 100%|      | 106/106 [00:03<00:00, 28.10it/s,
loss=0.71, acc=75.9]

Epoch 51/100 - Train Loss: 0.7103, Train Acc: 75.92% - Val Loss: 1.0503, Val
Acc: 70.50%

Epoch 52/100 [Train]: 100%|      | 106/106 [00:03<00:00, 28.06it/s,
loss=0.714, acc=75]

Epoch 52/100 - Train Loss: 0.7142, Train Acc: 74.98% - Val Loss: 1.0294, Val
Acc: 71.30%

Epoch 53/100 [Train]: 100%|      | 106/106 [00:03<00:00, 27.52it/s,
loss=0.576, acc=80]

Epoch 53/100 - Train Loss: 0.5762, Train Acc: 80.03% - Val Loss: 0.9211, Val
Acc: 73.78%
Saved best model to best_gcn_bilstm_model.pt

Epoch 54/100 [Train]: 100%|      | 106/106 [00:03<00:00, 27.52it/s,
loss=0.535, acc=81.7]

Epoch 54/100 - Train Loss: 0.5348, Train Acc: 81.68% - Val Loss: 0.9625, Val
Acc: 73.78%

Epoch 55/100 [Train]: 100%|      | 106/106 [00:04<00:00, 25.91it/s,
loss=0.528, acc=81.8]

Epoch 55/100 - Train Loss: 0.5283, Train Acc: 81.77% - Val Loss: 0.9218, Val
Acc: 73.69%

Epoch 56/100 [Train]: 100%|      | 106/106 [00:03<00:00, 27.32it/s,
loss=0.526, acc=82]

Epoch 56/100 - Train Loss: 0.5255, Train Acc: 81.98% - Val Loss: 0.9268, Val
Acc: 74.49%

Epoch 57/100 [Train]: 100%|      | 106/106 [00:03<00:00, 27.98it/s,
loss=0.494, acc=83.2]

Epoch 57/100 - Train Loss: 0.4940, Train Acc: 83.19% - Val Loss: 0.9834, Val
Acc: 73.96%

Epoch 58/100 [Train]: 100%|      | 106/106 [00:04<00:00, 25.08it/s,
loss=0.517, acc=81.7]

Epoch 58/100 - Train Loss: 0.5171, Train Acc: 81.74% - Val Loss: 0.9688, Val
Acc: 74.14%

Epoch 59/100 [Train]: 100%|      | 106/106 [00:03<00:00, 26.57it/s,
loss=0.523, acc=81.7]
```

```
Epoch 59/100 - Train Loss: 0.5233, Train Acc: 81.65% - Val Loss: 0.9537, Val
Acc: 73.78%

Epoch 60/100 [Train]: 100%|     | 106/106 [00:03<00:00, 26.80it/s,
loss=0.451, acc=84.1]

Epoch 60/100 - Train Loss: 0.4505, Train Acc: 84.14% - Val Loss: 0.8974, Val
Acc: 75.47%
Saved best model to best_gcn_bilstm_model.pt

Epoch 61/100 [Train]: 100%|     | 106/106 [00:04<00:00, 21.42it/s,
loss=0.425, acc=85.3]

Epoch 61/100 - Train Loss: 0.4250, Train Acc: 85.32% - Val Loss: 0.9072, Val
Acc: 75.91%

Epoch 62/100 [Train]: 100%|     | 106/106 [00:05<00:00, 21.06it/s,
loss=0.422, acc=85]

Epoch 62/100 - Train Loss: 0.4223, Train Acc: 84.96% - Val Loss: 0.9054, Val
Acc: 76.26%

Epoch 63/100 [Train]: 100%|     | 106/106 [00:04<00:00, 23.24it/s,
loss=0.398, acc=85.7]

Epoch 63/100 - Train Loss: 0.3984, Train Acc: 85.67% - Val Loss: 0.9141, Val
Acc: 74.84%

Epoch 64/100 [Train]: 100%|     | 106/106 [00:04<00:00, 23.50it/s,
loss=0.397, acc=86]

Epoch 64/100 - Train Loss: 0.3975, Train Acc: 86.00% - Val Loss: 0.9233, Val
Acc: 75.29%

Epoch 65/100 [Train]: 100%|     | 106/106 [00:04<00:00, 23.69it/s,
loss=0.399, acc=85.6]

Epoch 65/100 - Train Loss: 0.3994, Train Acc: 85.55% - Val Loss: 0.9193, Val
Acc: 75.47%

Epoch 66/100 [Train]: 100%|     | 106/106 [00:04<00:00, 23.41it/s,
loss=0.375, acc=87.6]

Epoch 66/100 - Train Loss: 0.3746, Train Acc: 87.59% - Val Loss: 0.9294, Val
Acc: 75.20%

Epoch 67/100 [Train]: 100%|     | 106/106 [00:04<00:00, 22.96it/s,
loss=0.365, acc=87.3]

Epoch 67/100 - Train Loss: 0.3646, Train Acc: 87.33% - Val Loss: 0.9115, Val
Acc: 76.26%

Epoch 68/100 [Train]: 100%|     | 106/106 [00:04<00:00, 23.35it/s,
loss=0.362, acc=87.3]

Epoch 68/100 - Train Loss: 0.3622, Train Acc: 87.27% - Val Loss: 0.8990, Val
Acc: 76.26%
```

```
Epoch 69/100 [Train]: 100%|      | 106/106 [00:04<00:00, 23.49it/s,
loss=0.349, acc=87.6]

Epoch 69/100 - Train Loss: 0.3493, Train Acc: 87.59% - Val Loss: 0.9075, Val
Acc: 76.17%

Epoch 70/100 [Train]: 100%|      | 106/106 [00:04<00:00, 23.50it/s,
loss=0.348, acc=87.4]

Epoch 70/100 - Train Loss: 0.3484, Train Acc: 87.44% - Val Loss: 0.9114, Val
Acc: 76.09%

Epoch 71/100 [Train]: 100%|      | 106/106 [00:04<00:00, 22.24it/s,
loss=0.328, acc=89.2]

Epoch 71/100 - Train Loss: 0.3277, Train Acc: 89.19% - Val Loss: 0.9436, Val
Acc: 76.09%

Epoch 72/100 [Train]: 100%|      | 106/106 [00:04<00:00, 22.23it/s,
loss=0.35, acc=87.4]

Epoch 72/100 - Train Loss: 0.3502, Train Acc: 87.36% - Val Loss: 0.9318, Val
Acc: 75.20%

Epoch 73/100 [Train]: 100%|      | 106/106 [00:04<00:00, 22.40it/s,
loss=0.321, acc=88.4]

Epoch 73/100 - Train Loss: 0.3212, Train Acc: 88.36% - Val Loss: 0.9162, Val
Acc: 76.17%

Epoch 74/100 [Train]: 100%|      | 106/106 [00:04<00:00, 22.33it/s,
loss=0.321, acc=89.4]

Epoch 74/100 - Train Loss: 0.3209, Train Acc: 89.36% - Val Loss: 0.9140, Val
Acc: 76.88%

Epoch 75/100 [Train]: 100%|      | 106/106 [00:04<00:00, 22.28it/s,
loss=0.317, acc=88.7]

Epoch 75/100 - Train Loss: 0.3173, Train Acc: 88.71% - Val Loss: 0.9203, Val
Acc: 75.73%
Early stopping triggered after 75 epochs
Test Accuracy: 77.59%
```

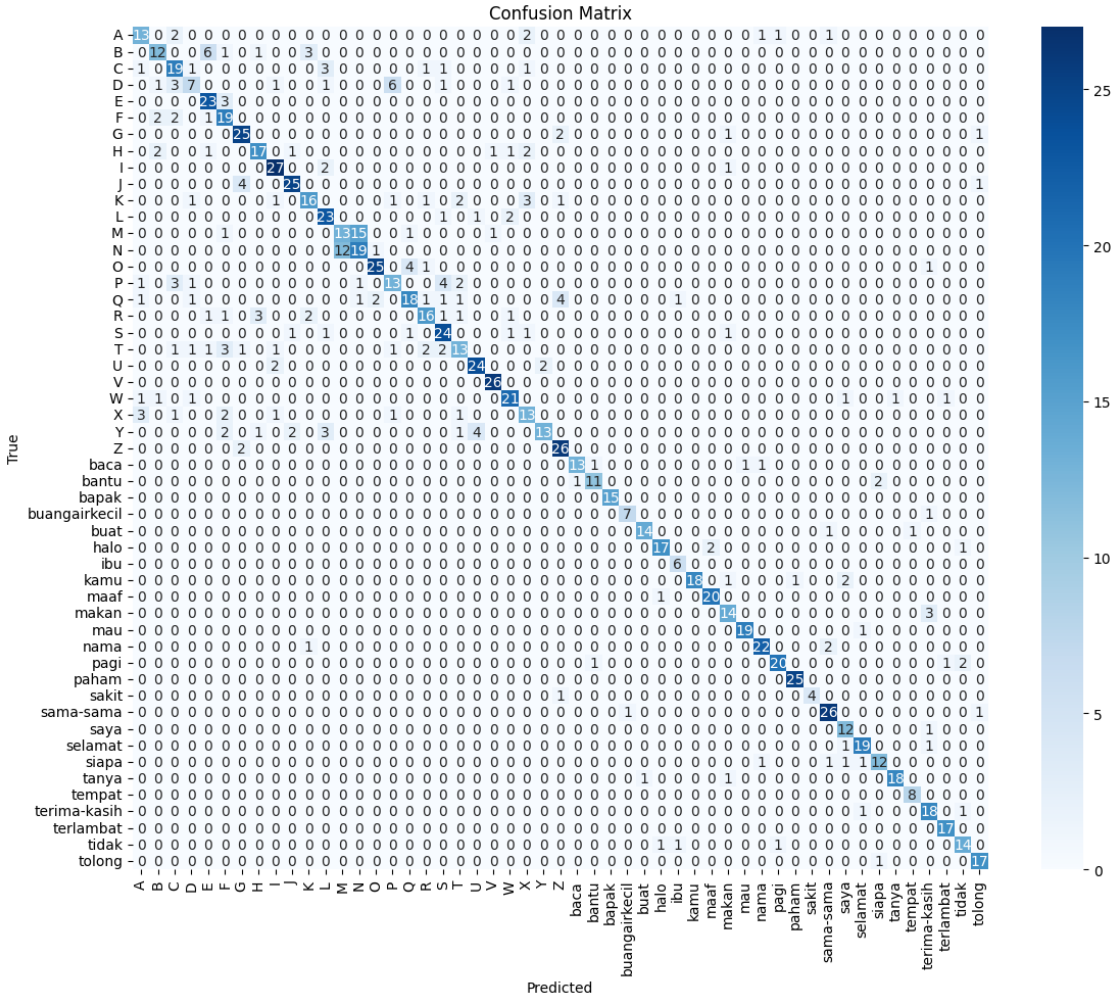


Classification Report:

[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50]

	precision	recall	f1-score	support
A	0.65	0.65	0.65	20
B	0.67	0.52	0.59	23
C	0.61	0.70	0.66	27
D	0.54	0.33	0.41	21
E	0.70	0.88	0.78	26
F	0.59	0.79	0.68	24
G	0.78	0.86	0.82	29
H	0.77	0.68	0.72	25
I	0.82	0.90	0.86	30
J	0.86	0.83	0.85	30
K	0.73	0.62	0.67	26
L	0.70	0.85	0.77	27
M	0.52	0.42	0.46	31
N	0.53	0.59	0.56	32
O	0.89	0.81	0.85	31
P	0.59	0.52	0.55	25
Q	0.75	0.58	0.65	31
R	0.73	0.62	0.67	26
S	0.69	0.80	0.74	30
T	0.62	0.50	0.55	26
U	0.83	0.86	0.84	28
V	0.93	1.00	0.96	26
W	0.78	0.78	0.78	27
X	0.59	0.59	0.59	22
Y	0.87	0.50	0.63	26

Z	0.76	0.93	0.84	28
baca	0.93	0.81	0.87	16
bantu	0.85	0.79	0.81	14
bapak	1.00	1.00	1.00	15
buangairkecil	0.88	0.88	0.88	8
buat	0.93	0.88	0.90	16
halo	0.89	0.85	0.87	20
ibu	0.75	1.00	0.86	6
kamu	1.00	0.82	0.90	22
maaf	0.91	0.95	0.93	21
makan	0.74	0.82	0.78	17
mau	0.95	0.95	0.95	20
nama	0.88	0.88	0.88	25
pagi	0.91	0.83	0.87	24
paham	0.96	1.00	0.98	25
sakit	1.00	0.80	0.89	5
sama-sama	0.84	0.93	0.88	28
saya	0.71	0.92	0.80	13
selamat	0.86	0.90	0.88	21
siapa	0.80	0.75	0.77	16
tanya	0.95	0.90	0.92	20
tempat	0.89	1.00	0.94	8
terima-kasih	0.72	0.90	0.80	20
terlambat	0.89	1.00	0.94	17
tidak	0.78	0.82	0.80	17
tolong	0.85	0.94	0.89	18
accuracy			0.78	1129
macro avg	0.79	0.79	0.79	1129
weighted avg	0.78	0.78	0.77	1129


Confusion Matrix
True
Predicted
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
baca
bantu
bapak
buangairkecil
buat
halo
ibu
kamu
maaf
makan
mau
nama
pagi
paham
sakit
sama-sama
saya
selamat
siapa
tanya
tempat
terima-kasih
terlambat
tidak
tolong
25
20
15
10
5
0