Gesture Recognition Using CNN-LSTM and 3D CNN Models

# A) Problem Definition:

The objective of this project was to create a gesture recognition model capable of classifying five distinct hand gestures from video sequences. The model would be implemented on a smart TV where gestures, captured via webcam, could serve as remote control commands. The challenge was to build a model that could generalize well across unseen data while maintaining low computational costs for real-time use.

# B) Dataset and Preprocessing:

The dataset consisted of video sequences, each containing 30 frames. The videos captured various hand gestures performed by users. The key preprocessing steps included:

1. Resizing images to various resolutions (120x120, 160x160) to experiment with how different image resolutions impacted model performance.

2. Normalization of pixel values to a range of [0, 1] by dividing by 255.

3. Data Augmentation: To combat overfitting, data augmentation was applied in some models. Techniques such as random rotations, shifting, and cropping were experimented with to improve generalization.

# C) Model Architectures:

Multiple models were explored, each using different architectures:

1. 3D CNN:

- 3D Convolutions were used to capture spatial and temporal features from the video frames.

- These models were relatively simple but effective for spatial features.

- Different filter sizes and resolutions were used to optimize performance.

2. CNN-LSTM:

- A CNN was used for feature extraction from individual frames.

- An LSTM was applied on top of the CNN outputs to capture the temporal dependencies between frames.

- This hybrid approach leveraged both spatial and temporal aspects of the gesture videos.

D**) Experiments and Results:**

****

Here is a summary of all the models discussed in the gesture recognition case study:

**1. ModelConv3D1**

Architecture: 3D Convolutional Neural Network (Conv3D)

Layers: Multiple Conv3D layers with ReLU activation, Batch Normalization, and MaxPooling3D layers, followed by Dense layers.

Parameters: 1,736,389

Performance: Training Accuracy: 33.16%, Validation Accuracy: 21%

**2. ModelConv3D2**

Architecture: Similar to ModelConv3D1 with added dropout layers.

Parameters: 1,117,061

Performance: Training Accuracy: 77.67%, Validation Accuracy: 23%

**3. ModelConv3D3**

Architecture: Reduced filter size and image resolution.

Parameters: 1,762,613

Performance: Training Accuracy: 76.77%, Validation Accuracy: 22%

**4. ModelConv3D4**

Architecture: Added more layers.

Parameters: 2,556,533

Performance: Training Accuracy: 74.35%, Validation Accuracy: 37%

**5. ModelConv3D6**

Architecture: Reduced number of parameters.

Parameters: 696,645

Performance: Training Accuracy: 77.03%, Validation Accuracy: 27%

**6. RNNCNN1**

Architecture: CNN + LSTM

Layers: Time Distributed Conv2D layers followed by LSTM and Dense layers.

Parameters: 1,657,445

Performance: Training Accuracy: 94.66%, Validation Accuracy: 79%

**7. ModelConv3D13**

Architecture: Similar to previous Conv3D models with data augmentation.

Parameters: 696,645

Performance: Training Accuracy: 77.45%, Validation Accuracy: 65%

**8. ModelConv3D14**

Architecture: Reduced network parameters with data augmentation.

Parameters: 504,709

Performance: Training Accuracy: 78.17%, Validation Accuracy: 71%

**9. RNNCNN2**

Architecture: CNN + GRU with data augmentation.

Parameters: 2,573,925

Performance: Training Accuracy: 98.43%, Validation Accuracy: 76%

**10. RNNCNN\_TL2**

Architecture: Transfer Learning with MobileNet and GRU.

Parameters: 3,693,253

Performance: Training Accuracy: 99.32%, Validation Accuracy: 92%

Final Model Selection

Chosen Model: RNNCNN1 (CNN + LSTM)

Reason: Balanced performance with a good trade-off between training and validation accuracy, and a manageable number of parameters.

Performance: Training Accuracy: 94.66%, Validation Accuracy: 79%

# F) Key Learnings and Challenges:

1. Overfitting: Overfitting was a major challenge in several models, particularly the 3D CNN models, which struggled to generalize beyond the training set. Techniques like dropout regularization and data augmentation helped, but further work is needed to close the training-validation accuracy gap.

2. CNN-LSTM’s Strength: The CNN-LSTM hybrid architecture proved to be the most effective. It managed to capture both spatial and temporal dependencies, which are critical for gesture recognition tasks. The combination of CNN for spatial feature extraction and LSTM for temporal learning allowed the model to perform better on the validation set.

3. Computational Complexity: Higher image resolutions (160x160) and larger filter sizes (3,3,3) increased the computational cost, leading to longer training times and more memory usage. Balancing model complexity with generalization was a key focus throughout the experiments.

# G) Future Work:

1. Further Regularization: Apply stronger regularization (dropouts, L2 weight regularization) to further close the gap between training and validation accuracy.

2. Optimize Model Size: Future work could focus on optimizing the model architecture for a low-memory footprint by experimenting with smaller CNN architectures or pruning to reduce the number of parameters while maintaining performance.

3. Ensemble Learning: Consider exploring ensemble learning techniques that combine multiple models (e.g., 3D CNNs and CNN-LSTM hybrids) to improve performance across different types of gestures.

4. Transfer Learning: Using pre-trained models (e.g., pre-trained CNNs for feature extraction) may reduce the amount of data needed for training while boosting performance.

5. Data Augmentation: Further data augmentation techniques, such as motion blur and brightness changes, could be added to simulate more realistic variations in gesture videos, thereby improving model robustness.