Gesture Recognition Project Write-Up

1. Introduction

This project explores gesture recognition using deep learning models, specifically designed for a smart television interface. The goal is to create a model that can recognize five hand gestures, allowing users to control the TV without a remote. Each gesture corresponds to a specific action, such as increasing or decreasing volume, skipping forward or backward, and pausing content. The challenge is to correctly classify these gestures using video data captured by a webcam.

Our approach involved experimenting with two different neural network architectures: a 3D Convolutional Network (Conv3D) and a CNN-RNN hybrid, combining Convolutional Neural Networks (CNNs) with Gated Recurrent Units (GRUs). Both architectures are well-suited to processing video sequences and were tested to determine which would provide the most accurate gesture classification.

2. Data Preprocessing

The dataset consists of short video clips, each divided into 30 frames, representing one of the five gestures. Each video is stored in a separate folder, with frames as images, and videos have varying dimensions (either 120x160 or 360x360 pixels). This variability required preprocessing to ensure consistency across all input data.

Our preprocessing steps were:

1. **Resizing**: All frames were resized to a fixed dimension of 64x64 pixels to reduce computational complexity while retaining essential information.
2. **Normalization**: Each pixel value was normalized to a range of [0, 1] by dividing by 255.0. This scaling helped stabilize the training process and improve model convergence.
3. **Data Loading via Generators**: Given the size of the dataset, we implemented a data generator function. This generator loaded each video in batches, resized and normalized the frames, and converted the gesture labels into one-hot encoded vectors. This allowed the model to process data in memory-efficient batches during training and validation.

3. Model Architectures

We implemented and evaluated two architectures for this task:

A. Conv3D Model

The Conv3D model is a deep 3D convolutional neural network designed to capture spatial and temporal information within video frames. It applies 3D convolutions across both spatial and temporal dimensions, allowing the model to understand changes in frames over time, which is essential for recognizing dynamic gestures.

Architecture Outline:

- **Input Layer**: 3D Conv layer with 32 filters, followed by Batch Normalization, ReLU activation, and Max Pooling.
- **Intermediate Layers**: Two additional 3D Conv layers (64 and 128 filters) with similar operations, followed by Max Pooling.
- **Flattening and Dense Layers**: After the convolutional layers, the output is flattened, followed by a dense layer with 512 units, ReLU activation, and Dropout for regularization.
- **Output Layer**: A softmax layer with 5 units for gesture classification.

This model performed well, leveraging 3D convolutions to analyze frame sequences, and achieved strong accuracy scores.

B. CNN + RNN (GRU) Model

The CNN-RNN model combines the spatial feature extraction capabilities of CNNs with the sequential processing abilities of RNNs, specifically using GRUs. The CNN layers extract spatial features from each frame, while the GRU layer processes these features over the temporal dimension, identifying gesture patterns.

Architecture Outline:

- **TimeDistributed Conv2D Layers**: Two layers of 2D convolutions, each wrapped in TimeDistributed to apply across each frame independently, followed by Batch Normalization, ReLU activation, and Max Pooling.
- **Flattening and GRU Layer**: The features from each frame are flattened, and a GRU layer processes them sequentially, aggregating information over time.
- **Fully Connected Layer**: A dense layer with 512 units and ReLU activation, followed by Dropout.
- **Output Layer**: A softmax layer with 5 units for gesture classification.

The CNN-RNN model demonstrated an ability to identify temporal dependencies effectively but required more training time than the Conv3D model due to its sequential structure.

4. Experiments

We conducted several experiments with different parameters, especially batch size, learning rate, and dropout rate, to optimize the models:

- **Experiment 1**: Initial Conv3D model with default settings. Accuracy was around 60% with a high loss, suggesting the need for more complex feature extraction.
- **Experiment 2**: Increased the batch size and added more filters to the Conv3D model. The accuracy improved to ~70%.
- **Experiment 3**: Optimized learning rate and applied dropout to reduce overfitting. This improved the Conv3D model's validation accuracy to over 80%.
- **Experiment 4**: Trained the CNN-RNN model using a similar setup. This model required more epochs to converge but achieved comparable accuracy.

After tuning, the Conv3D model achieved a validation accuracy of 83%, while the CNN-RNN model reached about 82%.

5. Results

The best-performing model was the Conv3D model, which achieved a validation accuracy of 83% and a loss of 0.43. The CNN-RNN model had slightly lower accuracy at 82%, indicating that Conv3D was more effective for this dataset, likely due to its direct handling of spatiotemporal patterns without the additional complexity of RNNs.

Final Model and Weights

The final weights for the Conv3D model were saved using the `.h5` format, fulfilling the requirement of storing model weights separately. This `.h5` file can be loaded to reproduce the best model's results.

6. Experiment Summary

Experiment	Model	Result	Decision + Explanation
1	Conv3D	Accuracy: 0.60, Loss: 2.0	Increased filters to capture more complex spatial features.
2	Conv3D	Accuracy: 0.70	Increased batch size and added filters. Observed improvement, but noticed overfitting.
3	Conv3D	Accuracy: 0.80	Added dropout layers to control overfitting. Improved validation accuracy.
4	CNN-RNN	Accuracy: 0.82	Tried hybrid approach; converged slower, but achieved similar results.
Final	Conv3D	Accuracy: 0.83	Selected Conv3D for best accuracy, faster convergence, and efficiency.

7. Conclusion

This project successfully implemented gesture recognition for smart TV control. The Conv3D model, which captures spatiotemporal relationships effectively, outperformed the CNN-RNN model in both accuracy and training efficiency. The final model is accurate and fast enough for real-time applications, achieving a strong balance of accuracy and inference speed.

Further improvements could involve:

- **Data Augmentation**: Increasing the dataset's diversity with augmented video sequences could improve generalization.
- **Hybrid Architectures**: Experimenting with ConvLSTM layers to capture even richer temporal features may improve results, though at a higher computational cost.

The project has demonstrated the feasibility of gesture recognition for smart TV applications and lays the groundwork for future enhancements.