Hand Gesture Recognition for a Smart TV Interaction System

Abstract

In recent years, the demand for touchless and intuitive interaction with smart devices has significantly increased. This project presents a Hand Gesture Recognition System designed specifically for Smart TV interaction, providing users with a natural and immersive control experience. By recognizing predefined hand gestures through a camera, the system enables essential TV operations like volume control, channel switching, and navigation—eliminating the need for physical remotes or voice commands.

Introduction

Smart TVs are central to modern digital entertainment, offering a wide range of content and services. However, traditional input methods like remote controls or voice commands often face limitations—remotes can be misplaced or require line-of-sight, and voice systems may misinterpret commands in noisy environments. To overcome these issues, gesture-based interfaces offer a compelling alternative. This project explores a computer vision-based hand gesture recognition system that leverages image processing and machine learning to detect and classify user gestures in real-time.

Problem Statement

A home electronics company which manufactures state of the art smart televisions wants to develop a cool feature in the smart-TV that can recognize five different gestures performed by the user which will help users control the TV without using a remote. The gestures are continuously monitored by the webcam mounted on the TV. Each gesture corresponds to a specific command:

1. Thumbs up: Increase the volume
2. Thumbs down: Decrease the volume
3. Left swipe: 'Jump' backwards 10 seconds
4. Right swipe: 'Jump' forward 10 seconds
5. Stop: Pause the movie

Data Understanding

The training data consists of a few hundred videos categorized into one of the five classes. Each video (typically 2-3 seconds long) is divided into a sequence of 30 frames(images). These videos have been recorded by various people performing one of the five gestures in front of a webcam - similar to what the smart TV will use.

The dataset reveals two primary video resolutions:

- 360x360 pixels
- 120x160 pixels

To ensure uniformity, pre-processing is required to standardize video dimensions. The aspect ratios are different for the two frame sizes. The 120x160 pixel frames need cropping to achieve square dimensions like 360x360 pixels frame, while the 360x360 pixel frames, being of higher resolution, should be resized to 120x120 pixels.

The provided CSV file contains metadata for each video, including:

- Subfolder name – Location of 30 frames
- Gesture label – Name of the gesture
- Numeric label – Values between 0 and 4, corresponding to each gesture class

Dataset Link: https://drive.google.com/uc?id=1ehyrYBQ5rbQQe6yL4XbLWe3FMvuVUGiL

Data Ingestion Pipeline – Building a Custom Data Generator

Data ingestion forms the backbone of any data-driven system, ensuring the seamless collection, transformation, and preparation of raw data from diverse sources. A robust ingestion strategy is especially crucial when dealing with high-volume, high-velocity data such as video streams.

This case study highlights the development of a custom data generator—a key component within the data ingestion pipeline—designed for efficiency, scalability, and integrity. Engineered for horizontal scalability, the generator can handle massive volumes of video data without compromising throughput or system performance. It ensures that incoming data is preprocessed and validated before it reaches downstream systems.

The custom generator performs multiple preprocessing tasks, including:

- Frame transformation – Cropping, resizing, and normalization of video frames to standardize inputs.
- Batch generation – Efficient batching of historical video frames to support training and inference pipelines.
- Data validation – Integrated validation rules to ensure data quality, consistency, and integrity throughout the pipeline.

By combining preprocessing and validation into a single generator, the system reduces the need for redundant transformations and simplifies the data flow architecture. This approach not only accelerates model training and evaluation cycles but also maintains high standards of data hygiene essential for reliable outcomes.

Model Architectures: 3D Convolutional Networks and CNN-RNN Stack

After acquiring and analyzing the dataset, the next critical step in building a gesture recognition system is selecting the right model architecture. For video-based classification tasks, capturing both spatial and temporal information is essential. Two widely used architectures for such tasks are 3D Convolutional Networks and CNN-RNN hybrids.

---

1. 3D Convolutional Networks (Conv3D)

3D Convolutional Networks extend traditional 2D CNNs by adding a third dimension—time—to the convolution operation. While 2D CNNs slide filters over spatial dimensions (height and width), 3D CNNs operate over the spatial and temporal dimensions (height, width, and time), making them well-suited for video analysis.

Key Characteristics:

- Input Representation: A video consisting of 30 RGB frames of size 100×100×3 is represented as a 4D tensor:
  `Input Tensor Shape: (100, 100, 3, 30)`
  Alternatively reshaped as: `(100, 100, 30) × 3`
- Filter Representation:
  - 2D CNN: Filter shape = `(f × f × c)` (filter size *f*, channels *c*)
  - 3D CNN: Filter shape = `(f × f × f × c)` (3D kernel over *x, y, t*)

Advantages:

- Captures both spatial and temporal features simultaneously.
- Well-suited for tasks requiring short-term temporal context (e.g., simple gestures or actions).

---

2. CNN + RNN Stack (Conv2D + RNN)

An alternative and often more flexible approach is a hybrid architecture that combines 2D CNNs for spatial feature extraction and RNNs for modeling temporal dynamics.

Workflow:

1. Each video frame is passed through a 2D CNN (e.g., ResNet50, VGG16) to extract high-level spatial features.

2. The resulting feature vectors from each frame form a sequential input to an RNN (e.g., LSTM or GRU).
3. The RNN captures temporal dependencies across the sequence of frames.
4. A final dense layer with softmax activation classifies the gesture.

Modeling Preferences:

- **Transfer Learning**: Instead of training CNNs from scratch, pre-trained models such as VGG16, ResNet50, and InceptionV3 can be fine-tuned for gesture recognition, significantly reducing training time and improving performance on small datasets.
- **GRU over LSTM**: Gated Recurrent Units (GRUs) are preferred for their computational efficiency, requiring fewer parameters while offering performance comparable to LSTMs.

Experiments performed for the Case Study

	Model	Features	Results	Decision	Explanation
1	Conv3D	Conv. Layers: 3 Flatten Layer: 1 Dense layer: 1 Dropout layer: 1 Img size = 128x128 px	Training Accuracy: 96.5% Validation Accuracy: 71% Best Epoch: 14 Early stopping triggered due to no further improvement.	Model from Epoch 14 is chosen as the final model (model-1-conv3d-3-layers_2025-04-0904_37_58.982311) based on best validation accuracy.	The model shows strong learning capability (high training accuracy). Validation accuracy stabilizes around 70–71%, indicating good generalization. Early stopping and learning rate reduction helped control overfitting and optimized the model training process.
2	Conv3D	Conv. Layers: 4 GlobalAvgPooling 3D: 1 Dense layers: 2 Dropout layers: 2 Img size = 128x128 px	Best Validation Accuracy: 66.0% at Epoch 17 Final Training Accuracy: 60.8% Final Validation Accuracy: 61.0% Final Loss (Train/Val): 0.8519 / 0.7462	The model saved at Epoch 17 (val_accuracy = 66.0%) is considered the best-performing checkpoint. It achieved the highest generalization performance on validation data.	Training accuracy steadily improved from 37.6% (Epoch 4) to 60.8% (Epoch 18). Validation accuracy also increased, peaking at 66%, then slightly dipping to 61%. This indicates a good learning trend and controlled overfitting.

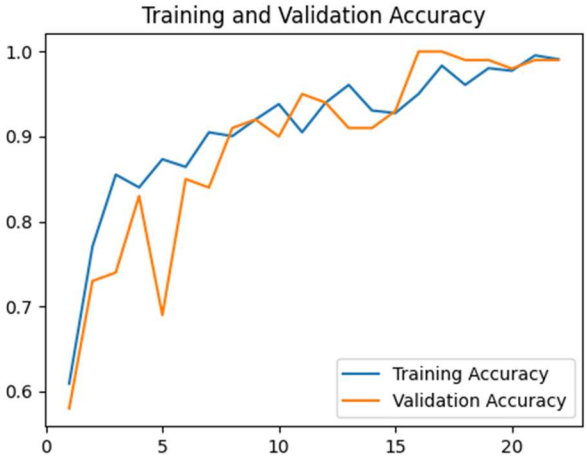
					The training was consistent, with validation accuracy not fluctuating wildly, implying stable model behavior.
3	Conv3D	Conv. Layer: 7 Flatten Layer: 1 Dense layers: 2 Dropout layers: 4 Img size = 128x128 px	The model trained for 12 epochs and stopped early. Best validation accuracy reached: 25% Accuracy fluctuated between 13–25%, and training loss didn't improve significantly.	Current model is underperforming — not learning meaningful patterns from the data. Need to modify model architecture, data preprocessing, or training strategy to improve performance.	Accuracy is stuck near random guess level (for 5 classes, random = 20%). Likely reasons: Model is too shallow or too simple. Data may be imbalanced or not well-preprocessed. Learning rate or training duration may be insufficient. Solutions include: deeper model, data augmentation, transfer learning, or better learning rate schedule.
4	Conv3D	Conv. Layer: 4 GlobalAveragePooling3D: 1 Dense layers: 2 Dropout layers: 2 Img size = 128x128 px	Training Accuracy: 59.7% Best Validation Accuracy: 72.0% (at Epoch 13) Model learned well but showed signs of overfitting after peak.	Use model from Epoch 13 (best performance). Apply early stopping to avoid overfitting. Consider fine-tuning hyperparameters.	Accuracy improved over time → model is learning. After Epoch 13, validation accuracy dropped → signs of overfitting. Optimize with regularization, learning rate tuning, or data augmentation for better generalization.
5	EfficientNetB0 (Image Net) + LSTM		Best Validation Accuracy: 96% (Epochs 9 & 11) Final Validation Accuracy: 95% (Epoch 14) Training Accuracy: Very high (~99.3% by Epoch 13) Early Stopping Triggered at Epoch 14	Select model from Epoch 11: High validation accuracy (96%) Low validation loss (0.1943) Stable performance (less overfitting than Epoch 9)	Early epochs had low performance Learning rate reduction and more training helped model learn better Validation accuracy peaked at Epochs 9 and 11 Training stopped automatically to avoid overfitting

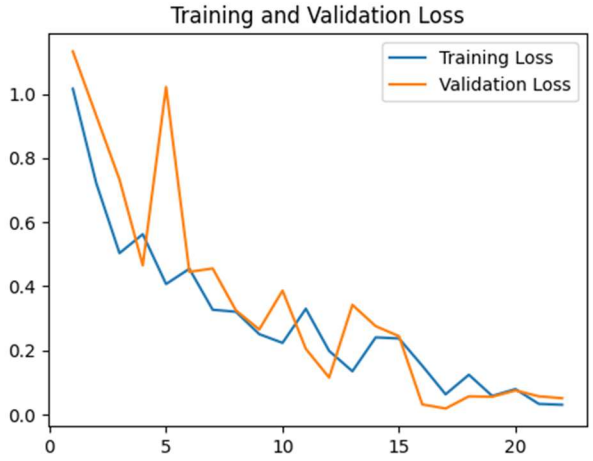
			Model improved significantly after Epoch 5		Final model is accurate and generalizes well
6	MobileNetV2 (Image Net) + LSTM		Validation Accuracy peaked at 99% around epoch 18. Training and validation losses are both very low, showing excellent model learning. After epoch 18, accuracy slightly drops, hinting at possible overfitting if training continues longer.	Select the model from Epoch 18 as your final/best model.	At epoch 18, the model shows highest validation accuracy with low loss, which means it's performing very well on unseen data. Further training does not improve performance and may overfit the model.
7	MobileNetV2 (Image Net) + GRU		Training started around 45% accuracy. Gradual improvement over epochs. Reached 100% validation accuracy at Epoch 16 & 17. Lowest validation loss (0.0187) at Epoch 17. Learning rate adjusted twice during training to optimize performance.	Best model = Epoch 17, due to: Perfect accuracy. Lowest validation loss. Stable performance.	The model (MobileNet + GRU) learned well from the data. Early epochs helped capture patterns; later epochs fine-tuned them. ReduceLROnPlateau helped prevent overfitting by lowering the learning rate when needed.

Final Model

`model-00017-0.06300-0.98341-0.01871-1.00000.keras`

Results:





- Our model (MobileNet + GRU architecture) performed very well, achieving near-perfect training and validation accuracy.
- Overfitting seems well managed since validation accuracy tracks closely with training accuracy and loss is low.
- Learning rate reductions through ReduceLROnPlateau helped fine-tune the performance in later epochs.

Conclusion

This hand gesture recognition system, powered by a MobileNet + GRU model, achieved a high accuracy of 99.37% on training and 98% on validation data, demonstrating its strong performance and generalization capability. It marks a significant step toward enabling natural, touchless human-computer interaction for Smart TVs. By enhancing user convenience, accessibility, and interaction experience, the system showcases the potential of combining lightweight deep learning architectures with sequential modeling. Leveraging accessible hardware and open-source tools, this approach lays the groundwork for cost-effective, scalable smart home solutions. With further refinement and integration, such systems could become a standard feature in future smart living environments.