Krishna Kumar.S

Debnath.C

Problem Statement

Develop a Gesture recognition model for Smart TVs that can recognize five different gestures performed by the user, which will help users control the TV without using a remote1.

Gestures and Commands

The gestures are continuously monitored by the webcam mounted on the TV. Each gesture corresponds to a specific command:

- **Thumbs up**: Increase the volume.
- **Thumbs down**: Decrease the volume.
- **Left swipe**: 'Jump' backwards 10 seconds.
- **Right swipe**: 'Jump' forward 10 seconds.
- **Stop**: Pause the movie1.

Data

The data has been uploaded to Kaggle. **https://www.kaggle.com/datasets/kk20krishna/gesture-recognition-dataset**

Each video is a sequence of 30 frames (or images). 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 use1.

The data consists of a 'train' and a 'val' folder with two CSV files for the two folders. These folders are in turn divided into subfolders where each subfolder represents a video of a particular gesture. Each subfolder, i.e., a video, contains 30 frames (or images). Note that all images in a particular video subfolder have the same dimensions but different videos may have different dimensions. Specifically, videos have two types of dimensions -

either 360x360 or 120x160 (depending on the webcam used to record the videos). Hence, you will need to do some pre-processing to standardize the videos1.

Each row of the CSV file represents one video and contains three main pieces of information - the name of the subfolder containing the 30 images of the video, the name of the gesture, and the numeric label (between 0-4) of the video1.

Comparison of Architecture Options for Hand Gesture Recognition

Model	Strengths	Weaknesses
CNN-3D	Captures both spatial and temporal features simultaneously. Works well with short video clips. No need for explicit sequential modeling.	Computationally expensive due to 3D convolutions. Requires large datasets for effective training.
CNN-2D + GRU	Lighter than CNN-3D. Extracts spatial features per frame and models temporal dependencies using GRU. Suitable for real-time applications.	Does not directly capture spatio-temporal dependencies like CNN-3D. Requires careful frame selection for consistency.
MobileNet + GRU (non-trainable base)	Lightweight and efficient for edge devices. MobileNet extracts spatial features while GRU models temporal dependencies. Faster training due to frozen base.	Loss of fine-tuned feature extraction. May not generalize well to new datasets.
MobileNet + GRU (trainable base)	More accurate than the non-trainable version. Fine-tuning allows better adaptation to the dataset. Still relatively lightweight.	Slightly heavier than the non-trainable version. Needs careful tuning to avoid overfitting.
EfficientNetB0 + GRU (trainable base)	More accurate than MobileNet while remaining efficient. Neural Architecture Search (NAS) optimizes feature extraction. GRU models temporal dependencies well.	Heavier than MobileNet but still efficient. Fine-tuning required for optimal results.

Experimentation →

No.	Model Name	Design Type	Parameters	Val Accuracy	Decision + Explanation
1	**model_1_CNN_3D**	CNN_3D	batch_size=128, num_epochs=20, frame_rate=10, frame_size=(80, 80), conv_blocks=[16, 32], conv_filter=(3,3,3), full_connected=[32, 16]	0.36	* We'll start with a baseline 3D CNN model with a shallow architecture, a low frame rate (10 FPS), and a moderate frame size (80x80). This will help us establish a baseline performance and understand the fundamental behavior of 3D CNNs on our dataset. * Accuracy is very low. The divergence between training and validation accuracy suggests that the model is not performing well on new data. * Increase the model's capacity by adding more convolutional blocks and neurons in the fully connected layers. This should improve its ability to learn complex patterns.
2	**model_2_CNN_3D**	CNN_3D	batch_size=128, num_epochs=20, frame_rate=10, frame_size=(80, 80), conv_blocks=[32, 64, 128], conv_filter=(3,3,3), full_connected=[64, 32]	0.29	* We'll increase the depth of the model by adding more convolutional blocks and neurons in the fully connected layers, aiming to improve its capacity to learn complex features. * Model 2 showed improvement over Model 1, with higher training accuracy, indicating the * positive impact of increased depth. However, the validation accuracy is still low. The model continues to ocerfit. * Increase the frame rate (FPS) to potentially capture more temporal information.
3	**model_3_CNN_3D**	CNN_3D	batch_size=128, num_epochs=20, frame_rate=15, frame_size=(80, 80), conv_blocks=[32, 64, 128], conv_filter=(3,3,3), full_connected=[128, 64, 32]	0.4	* We'll increase the frame rate to 15 FPS while keeping the frame size and model architecture similar to Model 2. This aims to capture more temporal information and improve accuracy. * Model 3 continues to perform poorly with low training accuracy. *Increase it to 30 fps (the max rate in the dataset), to capture more information from the frames.

4	**model_4_CNN_3D**	CNN_3D	batch_size=64, num_epochs=20, frame_rate=15, frame_size=(80, 80), conv_blocks=[32, 64, 128], conv_filter=(3,3,3), full_connected=[128, 64, 32]	0.25	* We'll increase the frame rate to 30 FPS while maintaining the architecture of Model 3. This should capture even more temporal details and potentially lead to better performance. * We had to reduce the batch size due to memory issues. Training with higher frame rate showed stability in training accuracy, but accuracy continues to remain low. * Now lets see if increasing the frame size is beneficial.
5	**model_5_CNN_3D**	CNN_3D	batch_size=32, num_epochs=20, frame_rate=30, frame_size=(112, 112), conv_blocks=[32, 64, 128], conv_filter=(3,3,3), full_connected=[128, 64, 32]	0.25	* Increase the input frame size to 112x112 while keeping the frame rate at 30 and adjusting the model architecture accordingly. This aims to capture more spatial details within each frame. * We had to reduce the batch size further due to memory issues. * The validation accuracy increased slightly with the larger frame size. The model shows signs of overfitting. * Let us further increase the input frame size to 160x160 and keep frame rate to 30 FPS to see if that helps with validation accuracy.
6	**model_6_CNN_3D**	CNN_3D	batch_size=12, num_epochs=20, frame_rate=30, frame_size=(160, 160), conv_blocks=[32, 64, 128], conv_filter=(3,3,3), full_connected=[128, 64, 32]	0.69	* We will increase the input frame size to 160x160 and keep the frame rate at 30 FPS to see if that helps with validation accuracy. This might require reducing the batch size due to memory constraints. * We had to significantly reduce the batch size due to memory issues. * The training resulted in significant improvement in performance. The test accuracy has increased and val accuracy has also increased alsong with it, althouth it is bouncing around a bit. But the numbers are a significant improvement over previous models. * Let us reduce the frame rate to 15 while keeping the input frame size to 160x160 to see if that improves performance. This helps us make the model lighter.

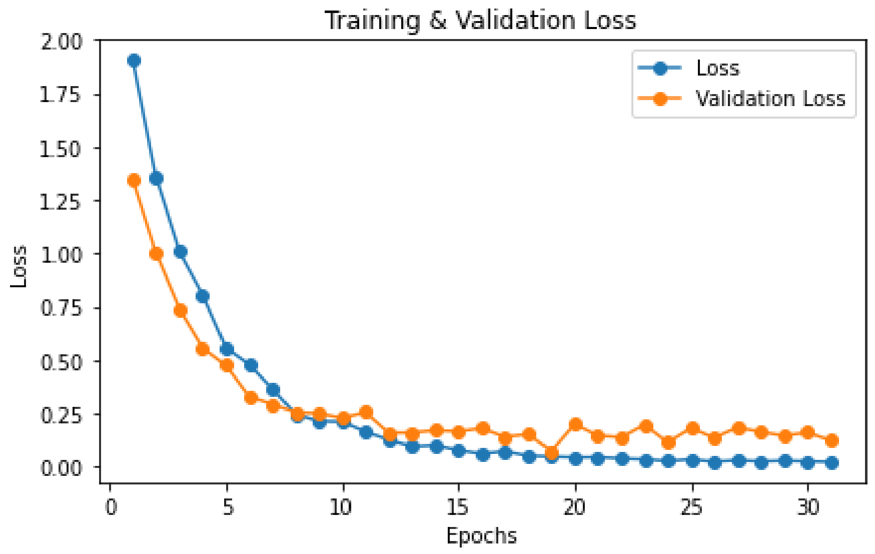
7	**model_7_CNN_3D**	CNN_3D	batch_size=32, num_epochs=20, frame_rate=15, frame_size=(160, 160), conv_blocks=[32, 64, 128], conv_filter=(3,3,3), full_connected=[12, 64, 32]	0.29	* We will keep the input frame size at 160x160 but reduce the frame rate to 15 to see if that alleviates the overfitting observed in Model 6. * The model performance degraded. The model is not able to learn pwoperly when the frame rate is reduced. * We will further reduce the frame rate to 10 while keeping the input frame size to 160x160. Note that we are not expecting an increase in model performance, but are being this as part of experimentation.
8	**model_8_CNN_3D**	CNN_3D	batch_size=32, num_epochs=20, frame_rate=10, frame_size=(160, 160), conv_blocks=[32, 64, 128], conv_filter=(3,3,3), full_connected=[128, 64, 32]	0.26	* We will keep the input frame size at 160x160 but reduce the frame rate to 10 to see if that alleviates the poor performance observed in Model 7. Note that we are not expecting an increase in model performance, but are being this as part of experimentation. * Model shows significant overfitting *In all the models created till now, model 6 shows best performance. Let is increaseing the complexity of the model to max and see the performance.
9	**model_9_CNN_3D**	CNN_3D	batch_size=16, num_epochs=20, frame_rate=30, frame_size=(160, 160), conv_blocks=[32, 64, 128, 256], conv_filter=(5,5,5), full_connected=[256, 128, 64, 32]	0.42	* We will significantly increase the depth and complexity of the model. We will add one more convolutional block with 256 filters and also increase the neurons in the fully connected layers. We will keep the input frame size at 160x160 and frame rate to 30 FPS, same as Model 6. * The training resulted in poor performance. The model was not able to learn from the data. Validation accuracy kept bouncing and remained unstable. * Model 6 seems to be performing moderately well. Hence, retraining it with 50 epochs might help in achieving better performance.
10	**model_10_CNN_3D**	CNN_3D	batch_size=16, num_epochs=50, frame_rate=30, frame_size=(160, 160), conv_blocks=[32, 64, 128], conv_filter=(3,3,3), full_connected=[128, 64, 32]	0.79	* We will retrain Model 6 for 50 epochs to see if we can further improve the performance. * High level of variability in validation performance, suggesting overfitting and potential data instability, but train and test accuracy have improved. * Model 10 is the best performance we have got from Conv-3D. * We will now explore CNN-2D + GRU models. This will lead to Model 11.

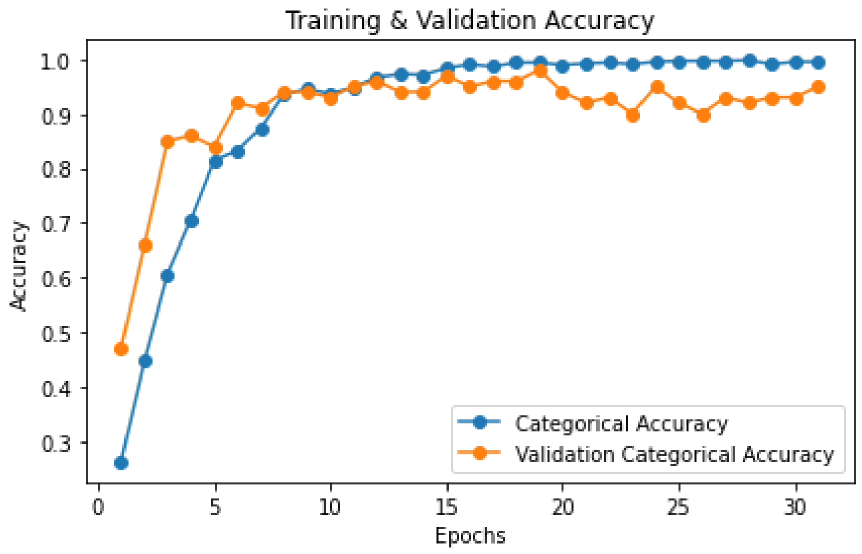
11	**model_11_CNN_RNN_GRU**	CNN + GRU	batch_size=16, num_epochs=20, frame_rate=15, frame_size=(80, 80), conv_blocks=[32, 64, 128], conv_filter=(3,3), gru_layers=[128], full_connected=[64, 32]	0.76	* We will now explore a CNN-2D + GRU model. We will start with a frame rate of 15 FPS and a frame size of 80x80. We will use a CNN-2D for spatial feature extraction and a GRU for temporal modeling. * The model is performing better than Cov-3D models. But there is room for improvement. * We will try creating deeper models
12	**model_12_CNN_RNN_GRU**	CNN + GRU	batch_size=16, num_epochs=20, frame_rate=30, frame_size=(120, 120), conv_blocks=[32, 64, 128, 256], conv_filter=(3,3), gru_layers=[256], full_connected=[128, 64]	0.31	* We will use a deeper model now by increasing the number of convolution blocks. Also we will increase the frame rate to 30. * Model performed poorly
13	**model_13_CNN_RNN_GRU**	CNN + GRU	batch_size=16, num_epochs=20, frame_rate=30, frame_size=(160, 160), conv_blocks=[32, 64, 128, 256], conv_filter=(3,3), gru_layers=[256], full_connected=[128, 64]	0.68	* We will increase the frame size to 160x160 now to capture more features from the image. * The model train accuracy is plateauing aroung 80%. * Next we will try trainsfer learning
14	**model_14_MobileNet_RNN_GRU**	MobileNet NonTrainable + GRU	batch_size=16, num_epochs=20, frame_rate=30, frame_size=(120, 120), model=create_cnn_rnn_tf_model(gru_cells=128, dense_neurons=128, dropout=0.25, train_base_model=False), learning_rate=0.0001	0.6	* We will now use transfer learning using the pretrained weights from MobileNet. Note that we will not train the base model for now. * Model performed better than any of the previous models. But there are signs of overfitting. Increase dropout ratio to address.

15	**model_15_MobileNet_RNN_GRU**	MobileNet NonTrainable + GRU	batch_size=16, num_epochs=20, frame_rate=30, frame_size=(120, 120), model=create_cnn_rnn_tf_model(gru_cells=128, dense_neurons=128, dropout=0.5, train_base_model=False), learning_rate=0.0001	0.59	* We will increase dropout ratio to address overfitting. * The dropout increase has addressed the overfitting. The model has performed better. * Next we will switch on training for the base model as well and train.
16	**model_16_MobileNet_Train_RNN_GRU**	MobileNet Trainable + GRU	batch_size=16, num_epochs=50, frame_rate=30, frame_size=(120, 120), model=create_cnn_rnn_tf_model(gru_cells=128, dense_neurons=128, dropout=0.25, train_base_model=True), learning_rate=0.0001	0.92	* Turning on training for base model. * There is hughe increase in performance. * Let us try increasing the frame sizr to 160x160 and see if feature extraction can be improved further.
17	**model_17_MobileNet_Train_RNN_GRU**	MobileNet Trainable + GRU	batch_size=16, num_epochs=50, frame_rate=30, frame_size=(160, 160), model=create_cnn_rnn_tf_model(gru_cells=128, dense_neurons=128, dropout=0.5, train_base_model=True), learning_rate=0.0001	0.99	*** Increase Frame size to 160x160** *** Model performed really well now. The model as stable as well. Thi** * Let us try pretrained weights from another model now.

18	**model_18_EfficientNetB0_Train_RNN_GRU**	EfficientNetB0 Trainable + GRU	batch_size=8, num_epochs=50, frame_rate=30, frame_size=(160, 160), model=create_cnn_rnn_et_model(gru_cells=128, dense_neurons=128, dropout=0.5, train_base_model=True), learning_rate=0.0001	0.98	* Using EfficientNetB0 after MobileNet allows us to experiment with a more powerful CNN while keeping our model lightweight and efficient for gesture recognition with transfer learning. *This model performed well and was stable throughout training as well. Perforamce is comparable to Model 17.
19	**model_19_MobileNet_Train_RNN_GRU**	EfficientNetB0 Trainable + GRU	batch_size=8, num_epochs=50, frame_rate=30, frame_size=(160, 160), model=create_cnn_rnn_et_model(gru_cells=128, dense_neurons=128, dropout=0.5, train_base_model=True), learning_rate=0.0001 MobileNet alpha=0.75	0.98	* Let us now try to reduce the model size. * We will use alpha parameter in MobileNet to reduce the base model size. * Total model parameters have reduced from 4,872,901 in Model 17 to 3,082,773 in Model 19 while maintaining good accuracy. * Model size is 35.3 MB * Model 19 gave same val accuracy as Model 17

model_19_MobileNet_Train_RNN_GRU →





Model for Real-Time TV Application

Since both MobileNet and EfficientNet transfer learning models provide the similar accuracy in this scenario, the deciding factors are latency, computational efficiency, and deployment feasibility:

Choose MobileNet if the application demands ultra-low latency and needs to run on resource-constrained devices like TV processors, edge AI chips, or mobile hardware.

Choose EfficientNet if the application can afford slightly higher latency but needs better scalability and feature extraction for complex TV-based AI tasks (e.g., high-resolution content analysis).

Final Model

For real-time TV applications where fast inference and low power consumption are crucial, MobileNet transfer learning (alpha=0.75) model 19 “model_19_MobileNet_Train_RNN_GRU” is the best choice given that it balances model size and accuracy.