Hand Gesture Recognition Project

Problem Statement:

To develop a feature in the smart-TV that can recognise five different gestures performed by the user which will help users control the TV without using a remote. Let's have professor Raghavan introduce you to the problem statement:

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 movie

Dataset:

The training data consists of a few hundred videos categorised 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.

Data Ingestion:

In this project, we implemented a custom batch data generator using Python's generator functions. Python generators are highly memory-efficient, which is crucial for deep learning models, especially when working with large datasets. Generators offer significant advantages in terms of performance, memory usage, and execution time, particularly for handling massive datasets. Additionally, they provide better readability and leverage the full range of features available with Python's native objects.

The generator yields a batch of data and "pauses" its execution until the fit_generator (or model.fit in newer versions of TensorFlow/Keras) calls next() to request the next batch. In Python 3, this functionality is implemented using the __next__() method. This approach is based on the concept of lazy evaluation, where data is processed only when needed, rather than loading everything into memory at once.

The yield statement in a generator returns one value (or batch) at a time, allowing us to process data incrementally. This is particularly useful for performing batch-wise gradient descent during model training, as it ensures that only the required amount of data is loaded into memory at any given time.

We chose to implement a custom data generator instead of using Keras's built-in ImageDataGenerator because our dataset comprises diverse data types from multiple sources, such as text, images, CSV files, and audio. A custom generator provides the flexibility to handle such heterogeneous data efficiently. Using the standard Keras.fit() method with large datasets like ImageNet could require hundreds of gigabytes of memory, making it impractical. Our custom generator avoids this issue by loading and processing data in manageable batches, ensuring efficient memory usage and scalability.

Model Architectures:

1. CNN3D
2. CNN + RNN

Experiments:

Experiment Number	**Model**	**Result**	**Decision + Explanation**
1	Conv3D 5 epochs, 25 batch size Image size 64X64, image cropping 10%	Training Accuracy: 45% Val accuracy: 21%	The model performs significantly better on the training data than on the validation data which is a sign of overfitting. we will increase the epochs and the batch size in the next iteration
2	Conv3D 5 epochs, 25 batch size	Training Accuracy: 38% Val accuracy: 29%	Training loss decreases steadily, but the validation loss remains high.

	Image size 64X64, image cropping 10% Dropout increase to 50% after 1st Dense layer		validation accuracy is consistently lower than the training accuracy. Both these confirm the case of overfitting. The learning rate is reduced multiple times (Epochs 5, 9, 13, and 17) due to stagnation in validation loss. However, this does not significantly improve validation performance. We will increase the image size and reduce the cropping
3	Conv3D Conv 3D Model with 20 epochs and 30 batch size , image size increased to 84X84, reduced cropping to 3 % from 10 % Dropout in the first dense layer was reverted from 0.50 to 0.25 Adam optimizer with a fixed learning rate of 0.0002 (no decay)	Training Accuracy: 38% Val accuracy: 29%	training accuracy (categorical_accuracy) improves from 0.2308 (Epoch 1) to 0.4721 (Epoch 20), showing that the model is getting better at classifying the training data. The gap in the training and val accuracy shows that the model has overfitted We will experiment with increasing the image size to 100x100 and batch size 50, epoch 30
4	Conv3D Conv 3D Model with 30 epochs and 50 batch size , image size increased to 100X100	Training Accuracy: 45% Val accuracy: 29%	training accuracy (categorical_accuracy) improves from 0.2504 (Epoch 1) to 0.4510 (Epoch 30), showing that the model is getting better at classifying the training data. The validation accuracy is still lower than the training accuracy, indicating some degree of overfitting We will experiment with batch size 30, increased

			image inputs to 30 , image size reduced to 84
5	Conv3D Conv 3D Model with 30 epochs and 30 batch size , image size reduced to 84X84	Training Accuracy: 68% Val accuracy: 71%	The model provides very moderate level accuracy. We will now move to a different CNN + LSTM architecture
6	ConvLSTM	Training Accuracy: 94 % Val accuracy: 85%	The gap between training accuracy (94.87%) and validation accuracy (85%) is relatively small but noticeable We will further experiment with increasing the image size
7	ConvLSTM Increased input image size to 160X160	Training Accuracy: 84% Val accuracy: 68%	The gap between the training and val accuracy is high indicating overfitting. Also the accuracy is low for our use case
8	ConvLSTM Increase layers, increase epoch to 25	Training Accuracy: 94% Val accuracy: 92%	Training accuracy improves consistently, reaching ~94% at Epoch 25. Validation accuracy fluctuates but rises to ~92% in Epoch 24, before settling at 80%. Overfitting appears to be minimal, but fine-tuning could still help.
9	ConvLSTM Increase network params	Training Accuracy: 97% Val accuracy: 92%	Training accuracy improves consistently, reaching ~97.13% at Epoch 25. Validation accuracy climbs to 92.00%, showing that the model generalizes well. The sharp rise after Epoch 10 suggests that the learning rate reduction helped stabilize training.

Final Model	ConvLSTM + GRU	Training Accuracy:98 % Val accuracy: 85%	There's a gap between training (97.74%) and validation accuracy (85.00%). While not drastic, it suggests possible overfitting in later epochs.