Gesture Recognition

Project by V S S Anirudh Sharma

Problem statement

The problem at hand involves developing a feature for smart televisions that can recognize and interpret different gestures performed by users. These gestures are monitored by the TV's built-in webcam and correspond to specific commands.

Gesture	**Interpretation**
Thumbs up	Increase volume
Thumbs down	Decrease volume
Left swipe	'Jump' backward 10 seconds
Right swipe	'Jump' forward 10 seconds
Stop	Pause the movie

Objective

The objective of the project is to develop a model that can accurately recognize and interpret gestures performed by users in videos captured by a webcam.

Setup

Each video has 30 frames. We chose alternate frames between the 5^{th} and 25^{th} frames, in the interest of computational efficiency, throughout the experiments. The first 5 and last 5 were left out since most of the “activity” would be in between the video.

Each frame was cropped to a square in the center and was resized to 100 x 100 shape, in the interest of computational efficiency. The original 360 x 360 squares were proving too heavy for the GPU.

These 100x100 frames were further normalized (z-score style normalization). This size was kept constant throughout the experiments.

We have chosen batch size 32 to start with, which we didn’t change as a hyperparameter during the experiments.

Base models

Both Conv3D and RNN-based models were considered in the experiments.

For 3D convolution, the base model was a single Conv3D layer model.

For RNN, a basic model with 1 Conv2D, 1 GRU, and 1 Dense layer was employed.

Base models are highlighted below in yellow and the best models in green.

Experiments

Experiment Number	Model	Best epoch accuracies (train, validation)	Number of Parameters	Observation and Action for the next experiment.
1	Simple Conv3D	99,65	59,010,373	Too much overfitting. Introduce dropouts.
2	Simple Conv3D with dropout.	100,57	59,010,373	Add layers. Model unable to capture the complexity of data properly.
3	Simple 2-layer Conv3D. Max pooling after every layer.	99.7,74	17,393,797	Really good. Mild overfitting. Add dropouts.
4	2 Layer Conv3D with dropouts. Max pooling after every layer.	100,65	17,393,797	Dropouts reduced performance, so the model must not be complex enough. Increase layers.
5	4-layer Conv3D. Max pooling after every 2 layers	93,67	8,127,717	Good but not best. Let's experiment on RNN-based models
6	1 Conv2D, 1 GRU, 1 Dense	98,60	14,765,893	Too much overfitting. Use GlobalAveragePooling2D and add dropout to GRU layer
7	1 Conv2D, 1 GRU with dropout, 1 Dense	~20, ~f20	20,037	Underfitting! Reduce the dropout ratio on the GRU layer from 0.5 to 0.2
8	1 Conv2D, 1 GRU with dropout 0.2, 1 Dense	~30, ~30	20,037	Still underfitting! Increase complexity. Add CNN layer.
9	2 Conv2D, 1 GRU, 1 Dense	~45, ~40	44,677	Still underfitting! Replacing GRU with LSTM.
10	2 Conv2D, 1 ConvLSTM2D, Dense	99, 71	1,186,885	Really good. Mild overfitting.

Final model

The aim was to get a model with greater than 70% validation accuracy. The best model would be the one with the least number of parameters.

- Using the Conv3D-based model, we were able to best achieve 74% validation accuracy.
- Using LSTM based model, we were able to best achieve 71% validation accuracy.

Both are decent results. But let's compare the total number of parameters:

- Conv3D-based model: 17,393,797
- LSTM-based model: 1,186,885

With just 7% of the parameters used by the Conv3D-based model, our LSTM-based model brings about the same accuracy (71% validation accuracy).

Thus, the LSTM-based model would be our final model.

You can the architecture on the right.