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 project aims 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. Throughout the experiments, we chose alternate frames between the 5th and 25th frames, in the interest of computational efficiency. 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 by dividing with 255. The reason this works would be that the normalized values must not be affected by other colors in the frame such as a red shirt or a green background, and that we are looking at natural images. 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 model

We considered 3D convolutional layers for this project. The base model has only one Conv3D layer. We have taken the bare minimum structure for a model that can deal with video data:

1. One 3D convolution layer
2. One Max pooling layer
3. One dense layer
4. And one output layer

Experiments

Experiment Number	Model	Best epoch accuracies (train, validation)	Number of Parameters	Model Size	Observation and Action for the next experiment.
1	Conv3d, Max-pool, Dense	100, 83	7,377,381	28.14 MB	mild overfitting. Introduce dropouts.
2	Conv3D with dropout, Max-pool, Dense	100, 80	7,377,381	28.14 MB	Worse off. Reduce dropout ratio
3	Conv3D with reduced dropout, Max-pool, Dense	100,78	7,377,381	28.14 MB	Worse off. Revert and add l2 regularization on conv3d layer
4	Conv3D with dropout and l2 reg, Max-pool, Dense	100, 82	7,377,381	28.14 MB	Overfitting. Reduce model complexity
5	Conv3D, Max-pool, output	100, 60	554,509	2.12 MB	Heavy overfitting. Increase pool size from (3,3,3) to (4,4,4)
6	Conv3D, Max-pool, Output	96, 81	70,669	0.27 MB	Fine model but still overfitting. Add dropouts
7	Conv3D, Dropout, Max-pool, Output	100, 82	70,669	0.27 MB	Overfitting. Try padding.
8	Conv3D with padding, Dropout, Max-pool, Output	100, 75	76,549	0.29 MB	Didn't help. Even increased size. Revert and reduce filters from 8 to 4
9 (BEST MODEL)	Conv3D, Dropout 0.9, Max-pool, Output	89, 87	35,337	0.13 MB	Overfitting resolved. Decent results and lightweight. Try reducing dropout to 0.8 and see impact
10	Conv3D, Dropout 0.8, Max-pool, Output	93, 80	35,337	0.13 MB	Overfitting just by small reduction in dropout

Final model

The goal was to maximize validation accuracy (above 80%), using the least possible number of parameters.

Throughout the experiments, we have been simplifying the architecture, size and increasing regularization.

In our pursuit of the lightest and the best model, we have our best model with

1. 89% training accuracy
2. 87% validation accuracy
3. 0.13 MB size
4. 35337 parameters

This was the simplest model conceivable with Conv3D-based architectures.