Problem Statement

Develop a 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:

- 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

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

Understanding the Dataset

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 which is like what the smart TV will use.

The data is in a zip file. The zip file contains 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).

All the images in a 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, we will need to do some pre-processing to standardize the videos.

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 video.

Architecture: 3D Convolutions and CNN-RNN Stack

There are different architectures to solve problem and for analyzing videos using neural networks, two types of architectures which are used commonly are

1. CNN + RNN architecture

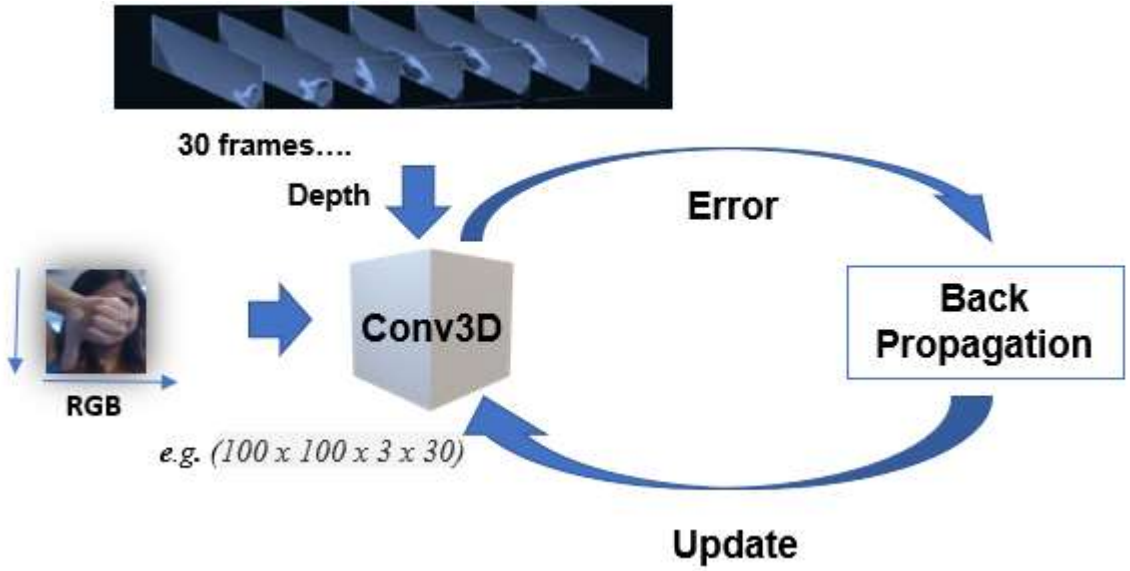
In this architectural setup, you pass the images of a video through a CNN, the conv2D network will extract a feature vector for each image and a sequence of these feature vectors is then fed to an RNN-based network. The output of the RNN is a regular SoftMax for a classification problem. The gesture recognition is an example of classification.

Input with spatial structure, like images, cannot be modeled easily with standard RNN. Hence we are going with CNN RNN(GRU) which is an RNN architecture specifically designed for sequence prediction problems with spatial inputs, like images or videos.

CNN is used as an image "encoder", by first pre-training it for an image classification task and using the last hidden layer as an input to the RNN decoder which will capture the information hidden in the sequence of the images.

2. 3D Convolutional Neural Networks (Conv3D)

3D convolution neural networks are a natural extension to the 2D convolution neural networks.



In 2D conv, the filter is moved in two directions (x and y) whereas in 3D conv, you move the filter in three directions (x, y and z). In this case, the input to a 3D conv is a video (which is a sequence

of 30 frames i.e., RGB images).

We assume that the shape of each image is 100 x 100 x 3, for example, the video becomes a 4D tensor of shape 100 x 100 x 3 x 30 which can be written as (100 x 100 x 30) x 3 where 3 is the number of channels. Hence, deriving the analogy from 2D convolutions where a 2D kernel/filter (a square filter) is represented as (f x f) x c where f is filter size and c is the number of channels, a 3D kernel/filter (a 'cubic' filter) is represented as (f x f x f) x c (here c = 3 since the input images have three channels). This cubic filter will now '3D-convolve' on each of the three channels of the (100 x 100 x 30) tensor.

Data Generators

In the Data Generators, we are going to pre-process the images as we have images of 2 different dimensions (120*120 and 90*90) as well as create a batch of video frames. The generator should be able to take a batch of videos as input without any error. Steps like cropping, resizing and normalization should be performed successfully.

- **Resizing and cropping of the images:** This is done to ensure that the NN focuses only on the gestures effectively rather than the background noise present in the image
- **Normalization of the images:** Normalizing the RGB values of an image can at times be a simple and effective way to get rid of distortions caused by lights and shadows
- During the later stages for improving the model's accuracy, we have also used **data augmentation**, where we have slightly rotated the pre-processed images of the gestures to bring in more data for the model to train on and to make it more generalizable

Analysis & Findings

- As the no. of trainable parameters increase, the model takes much more time for training
- Batch size $\propto$ GPU memory / available compute: A large batch size can throw GPU Out of memory error, and thus here we had to play around with the batch size till we were able to arrive at an optimal value of the batch size which our GPU could support
- With the increase in the batch size, reduces the training time, however it results in negative impact on the model accuracy. This outlines that there is always a trade-off between time vs accuracy i.e., if we want our model to be ready in a short time then we choose larger batch size else choose lower batch size for the model to be more accurate.
- Data Augmentation and Early stopping helps in overcoming the problem of overfitting which our initial version of model was facing
- GRU is a better choice than an LSTM with CNN than Conv3D since it has lesser number of gates (and thus parameters). This entirely depends on the kind of data used, the architecture we developed, and the hyper-parameters chosen

The experiment details i.e., no. of parameters, metric value – Accuracy are mentioned in the table below:

Sr No	Experiment Parameters	Model	Trainable Params	Train Acc (%)	Val Acc (%)	Observation
1	Batch size = 32 No.of Frames = 15 Image Dimension = 90*90 Epochs = 10	Conv3D (Layer = 4, BatchNorm =Y Dropout = Y, activation func = relu)	470K	50	37	Accuracy attained is average. Since we chose only 15 frames out of 30 for training it generated a generic model. Hence, need to increase the frame samples for training.
2	Batch size = 32 No.of Frames = 30 Image Dimension = 90*90 Epochs = 10	Conv3D (Layer = 4, BatchNorm =Y Dropout = Y, activation func = relu)	470K	73	68	After increasing the frames from 15 to 30 for training we got a pretty decent accuracy. To further tune the model to achieve higher accuracy will try with additional epochs.
3	Batch size = 32 No.of Frames = 30 Image Dimension = 90*90 Epochs = 20	Conv3D (Layer = 4, BatchNorm =Y Dropout = Y, activation func = relu)	470K	81	57	Epochs increased from 10 to 20. Training accuracy increased however validation accuracy degraded indicating overfitting of model.
4	Batch size = 64 No.of Frames = 30 Image Dimension = 90*90 Epochs = 20	Conv3D (Layer = 4, BatchNorm =Y Dropout = Y, activation func = relu)	470K	44	62	Batch size increased to 64 from 32. eventhough training time was lesser,this degraded the accuracy from 76 to 41%.
5	Batch size = 16 No.of Frames = 30 Image Dimension = 90*90 Epochs = 10	Conv3D (Layer = 4, BatchNorm =Y Dropout = Y, activation func = relu)	470K	34	46	Batch size decreased to 16 from 32. This model is taking longer Epochs to learn and also accuracy attained is low compared to batch size of 32 for similar Epochs. Hence it is suggested to use batch size of 32.
6	Batch size = 32 No.of Frames = 20 Image Dimension = 90*90 Epochs = 20	Conv3D (Layer = 4, BatchNorm =Y Dropout = Y, activation func = relu)	470K	61	56	no.of frames increase from 15 to 20, and we could see a slight improvement in the accuracy but still model can be improved.
7	Batch size = 32 No.of Frames = 30 Image Dimension = 90*90 Epochs = 20	Conv3D (Layer = 4, BatchNorm =Y Dropout = N, activation func = relu)	470K	99	75	Removed dropout in this model. As it can be seen model is highly overfitting. Hence Dropout is recommended to keep the model generic.
8	Batch size = 32 No.of Frames = 30 Image Dimension = 90*90 Epochs = 20	Conv3D (Layer = 3, BatchNorm =Y Dropout = Y, activation func = relu)	3,100K	42	50	Layer of Conv3D model is decreased to 3 layers from 4. The no.of trainable parameters increased significantly in this case and also accuracy degraded. Hence higher layers are recommended.
9	Batch size = 32 No.of Frames = 30 Image Dimension = 90*90 Epochs = 10	Conv3D (Layer = 4, BatchNorm =N Dropout = Y, activation func = relu)	470K	40	56	Batch normalization is not used in this model. Without batch normalization in place model tends to learn slower and requires more epoch to reach the results.Hence the accuracy is lower for 10 epochs.
10	Batch size = 32 No.of Frames = 30 Image Dimension = 120*120 Epochs = 20	Conv3D (Layer = 4, BatchNorm =Y Dropout = Y, activation func = relu)	863K	99	75	Image dimension increased from 90 to 120, keeping the epochs to 20 overfits the Conv3D model. No.of trainable parameters increased from 470k to 863k increasing the training time as well. This is overfitting.

11	Batch size = 32 No.of Frames = 15 Image Dimension = 90*90 Epochs = 10	Conv2d+RNN(GRU) (Layer = 4, BatchNorm =Y Dropout = Y, activation func = relu)	302K	19	31	Accuracy is very low. Since we chose only 15 frames out of 30 for training it generated a generic model. Hence, need to increase the frame samples for training.
12	Batch size = 32 No.of Frames = 30 Image Dimension = 90*90 Epochs = 10	Conv2d+RNN(GRU) (Layer = 4, BatchNorm =Y Dropout = Y, activation func = relu)	549K	92	87	After increasing the frames from 15 to 30 for training we got a pretty good accuracy. With batch normalization and droput enabled, this model seem to achieve the expected result well without overfitting.
13	Batch size = 64 No.of Frames = 30 Image Dimension = 90*90 Epochs = 10	Conv2d+RNN(GRU) (Layer = 4, BatchNorm =Y Dropout = Y, activation func = relu)	302K	17	5	Batch size increased to 64 from 32. eventhough training time was lesser,this degraded the accuracy significantly. Higher batch size generalized the model.
14	Batch size = 128 No.of Frames = 30 Image Dimension = 90*90 Epochs = 10	Conv2d+RNN(GRU) (Layer = 4, BatchNorm =Y Dropout = Y, activation func = relu)	302K	NA	NA	Batch size increased to 128 from 32. This threw an error due to higher memory requirment for RAM and GPU. Hence, it is not recommended to use higher batch size with given system configuration.
15	Batch size = 16 No.of Frames = 30 Image Dimension = 90*90 Epochs = 10	Conv2d+RNN(GRU) (Layer = 4, BatchNorm =Y Dropout = Y, activation func = relu)	302K	16	14	Batch size decreased to 16 from 32. This model is taking longer Epochs to learn and also accuracy attained is lower compared to batch size of 32 for similar Epochs. Hence it is suggested to use batch size of 32 for faster training.
16	Batch size = 32 No.of Frames = 20 Image Dimension = 90*90 Epochs = 10	Conv2d+RNN(GRU) (Layer = 4, BatchNorm =Y Dropout = Y, activation func = relu)	302K	24	18	No.of frames increase from 15 to 20, and we could see a slight improvement in the accuracy but still model can be improved. Hence suggested to use 30 frames.
17	Batch size = 32 No.of Frames = 30 Image Dimension = 90*90 Epochs = 10	Conv2d+RNN(GRU) (Layer = 4, BatchNorm =N Dropout = Y, activation func = relu)	302K	68	52	Batch normalization is not used in this model. Without batch normalization in place model tends to learn slower and requires more epoch to reach the results.
18	Batch size = 32 No.of Frames = 30 Image Dimension = 90*90 Epochs = 10	Conv2d+RNN(GRU) (Layer = 4, BatchNorm =Y Dropout = N, activation func = relu)	302K	99	68	Removed dropout in this model. As it can be seen model is highly overfitting. Hence Dropout is recommended to keep the model generic.
19	Batch size = 32 No.of Frames = 30 Image Dimension = 90*90 Epochs = 10	Conv2d+RNN(GRU) (Layer = 3, BatchNorm =Y Dropout = Y, activation func = relu)	584K	98	81	Layer of model is decreased to 3 layers from 4 . The no.of trainable parameters increased significantly in this case and and it is slightly overfitting model. Also time consumed for training these parameters is high.Hence, it is recommended to go for more layers.
20	Batch size = 32 No.of Frames = 30 Image Dimension = 120*120 Epochs = 10	Conv2d+RNN(GRU) (Layer = 4, BatchNorm =Y Dropout = Y, activation func = relu)	498K	99	81	Image dimension increased from 90 to 120, keeping the epochs to 10 gives the best possible results for training accuracy for CNN+RNN model. No.of trainable parameters increased though from 302K to 498K. This model is slighlty overfitting compared to 90*90 dimension input.

Summary:

Model using **Conv2d+RNN(GRU)**, gave us the best result of train accuracy of **92%** and validation accuracy of **87%** using all the 30 frames and 90*90 image dimension. The same model is submitted for the review. Cropping and other preprocessing also did not affect much on the final accuracy. While we did try various models with different hyperparameters, there was trade off in either Validation accuracy or time and resource consumption.

Comparatively Conv2D with RNN GRU performed better than Conv3D. The highest validation accuracy without overfitting achieved using Conv3d model was **68%** with train accuracy of **74%**. Hence, GRU is a better choice than Conv3D since it has lesser number of gates and this architecture specifically designed for sequence prediction problems