GESTURE RECOGNITION PROJECT

# Problem Statement :

Imagine you are working as a data scientist at a home electronics company which manufactures state of the art **smart televisions**. You want to develop a cool 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.

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

# 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 - similar to what the smart TV will use. Sample Data is shown below.



# Project Objective and Model Architecture:

Train a model that can correctly identify the 5 hand gestures based on the Test Data.

For the same, there are two Architectures suggested.

* **3D Convs and**
* **CNN-RNN Stack**

3D convolutions are a natural extension to the 2D convolutions. Just like in 2D conv, you move the filter in two directions (x and y), 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 RGB images).

CNN-RNN stack- 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 such as this one).

# Model Architecture and Training:

* Experimentation with different configurations of the model, hyperparameters, number of iterations and combinations of batch sizes with image sizes, choice of kernels(size, padding, stride) for best accuracy.
* Also, tried different Learning Rates and used *ReduceLRonPlateau* to reduce Learning Rate if the metric(*val\_loss*) remains unchanged in between epochs.
* For Optimizers, tried *SGD()* and *Adam()* but went ahead with *Adam()* optimizer as it led to improvement in accuracy by rectifying high variance in the model’s parameters.
* Also did *Early Stopping*, when the model performance would stop improving.

# OBSERVATIONS:

* Training time is directly proportional to number of parameters
* *Batch Size* should be chosen as per the GPU selected. A large Batch size throws Out of Memory error. So, we need to find the optimal Batch size which our GPU can support.
* Increase in Batch size improves on computational cost however it affects accuracy implying a trade-off between both.

So, we need to find the best Batch size which gives best computational time and highest accuracy.

* We achieved better accuracies when the image size selected was 160\*160 .
* *Early Stopping* helped us in overcoming the problem of overfitting to a great extent.
* We obtained a better performing model using *Conv3D* than

*CNN+LSTM and CNN+GRU based model.*

**FINA*L MODEL:***

*We selected Conv3D model#1 as the best performing model over CNN+LSTM and CNN+GRU for following reasons:*

*• Training Accuracy : 89%*

*• Validation Accuracy: 84%*

*• Model is Statistically simpler.*