Deep Learning Course Project- Gesture Recognition

- **Suraksha G Bharadwaj – Group Facilitator**
- **Harshith R**

Problem Statement

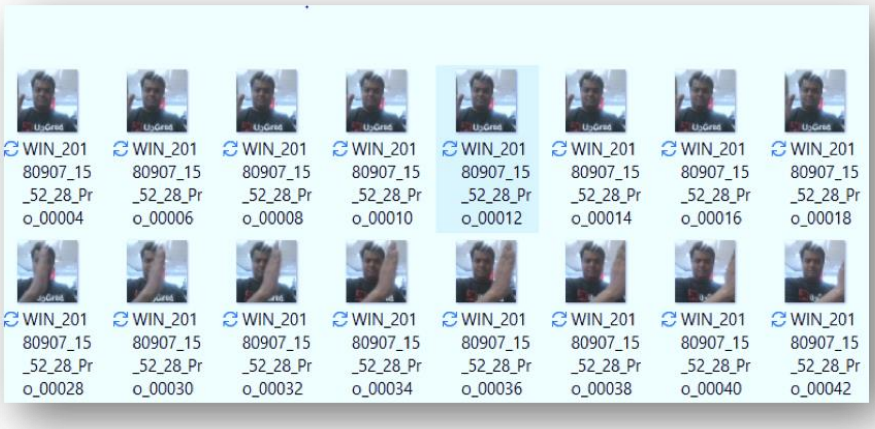
As a data scientist at a home electronics company which manufactures state of the art smart televisions. We want to develop a cool feature in the smart-TV that can recognize five different gestures performed by the user which will help users control the TV without using a remote.

- 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.

Here's the data:https://drive.google.com/uc?id=1ehyrYBQ5rbQQe6yL4XbLWe3FMvuVUGiL

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.



Objective

Our task is to train different models on the 'train' folder to predict the action performed in each sequence or video and which performs well on the 'Val' folder as well. The final test folder for evaluation is withheld - final model's performance will be tested on the 'test' set.

Two types of architectures suggested for analyzing videos using deep learning:

1. **3D Convolutional Neural Networks (Conv3D)**

3D convolutions are a natural extension to the 2D convolutions you are already familiar with. 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). If 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.

2. **CNN + RNN architecture**

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).

The following table represents the results obtained during the model building.

Experiment Number	Model	Result	Decision + Explanation
1	Conv3D	Train Accuracy: 0.17, Validation Accuracy: 0.16	Model seems to have a less accuracy and loss is not reducing.
2	Conv3D	Train Accuracy: 0.17, Validation Accuracy: 0.20	Model doesn't seem to have improved any bit.
3	Conv3D	Negative Dimension Error.	The new CNN kernel sizes are not compatible with the output

			of previous layers. Let's reduce the kernel size of new layers.
4	**Conv3D**	**Train Accuracy: 0.20, Validation Accuracy: 0.20**	**Still there is no improvement in the model. Let's add Batch normalization layers after every CNN and dense layers.**
5	**Conv3D**	**Train Accuracy: 0.9062, Validation Accuracy: 0.2708**	**Model is able to over-fit on less data (Ablation data set), Let's Training on full data and increasing epochs to 50.**
6	**Conv3D**	**Train Accuracy: 0.9062, Validation Accuracy: 0.70**	**Mode is having over-fitting as there is huge gap between training and validation accuracies. Let's add some dropouts that the model can be generalized.**
7	**Conv3D**	**Train Accuracy: 0.9896, Validation Accuracy: 0.7734**	**There is a bit of increase in the model validation accuracy and training accuracy also. Lets increase the drop out values from**
8	**Conv3D**	**Train Accuracy: 0.9896, Validation Accuracy: 0.7734**	**After increase the dropout the model validation score further reduced and the model is over-fitted. Let's use 0.2 only remove a CNN layer to reduce the complexity.**
9	**Conv3D**	**Train Accuracy: 1.00, Validation Accuracy: 0.77**	**Still the model is over-fitting. Let's use a Global Average Pooling instead of Flatten Layer.**
10	**Conv3D**	**Train Accuracy: 0.9509 Validation Accuracy: 0.9061**	**The model is wonderful and the training and validation scores are good. The model has 710,533 trainable parameters. Let's try architectures too.**

11	**Time distributed +GRU**	**Train Accuracy: 0.9554** **Validation Accuracy: 0.8023**	**The model is working quite well on validation dataset with less trainable parameters (98,885), Lets add some drop outs after each layer, so that both train and validation accuracies will be closure.**
12	**Time distributed +GRU**	**Train Accuracy: 0.8720** **Validation Accuracy: 0.6016**	**The model accuracy further deteriorated; Let's replace GRU with a plain Dense Layer Network and some Global Avg Pooling.**
13	**Time distributed +Dense**	**Train Accuracy: 0.8780** **Validation Accuracy: 0.8750**	**This is good model with training and validation accuracies with number of params 128,517. Let's use different architecture of model with time distributed and ConvLSTM2D.**
14	**Time distributed +ConvLSTM2D**	**Train Accuracy: 0.9673** **Validation Accuracy: 0.9375**	**This is the best model so far we can get. The validation accuracy is good and the numbers of parameters are 13,589. The model size is also so small 226KB.**