Deep Learning Course Project- Gesture Recognition

- **Aayushi Mittal – Group Facilitator**
- **Bhupender Pillay**

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 recognise 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.

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 analysing 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).

Data Generator

Experiment Number	**Model**	**Result**	**Decision + Explanation**
1	**Conv3D**	**Training accuracy 0.9193 Validation accuracy 0.8400**	**We can see val_loss did not improve from 0.368 Seems overfitting** **Reduce filter size to (2,2,2) and image res to 120 x 120, - Batch Size = 30 and No. of Epochs = 25**
2	**Conv3D**	**Training accuracy 0.3401** **Validation accuracy 0.1600**	**Early stopping at epoch 11/25 Performance drastically decrease. AddingAdditional layers - Batch Size = 20 and No. of Epochs = 25**
3	**Conv3D**	**Training accuracy 0.8959** **Validation Accuracy: 0.9300**	**We get a best validation accuracy of 93 till now**. overfitting. Reducing the number of parameters
4	**Conv3D**	**Validation Accuracy: 0.26**	val_loss did not improve from 2.18794 an validation accuracy decreased. Reducing the number of parameters again
5	**Conv3D**	**Validation Accuracy: 0.26**	performance decreased. Applying data augmentation

6	**Conv3D**	**validation accuracy so far 0.8100**	**Hence choosing this model as final**