Gesture Recognition Case Study

- **Aloke Kumar Mukherjee (Group – C54)**

Problem Statement

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

Data Set: https://drive.google.com/drive/my-drive/Project_data.zip

Understanding the Dataset

- The training data consists of a few hundred videos categorised 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.
- 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 images in a particular 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). So some pre-processing is required to standardise 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.

Goal

The task is to train a model on the 'train' folder which performs well on the 'val' folder as well (as usually done in ML projects). There is the test folder for evaluation purposes - the final model's performance will be tested on the 'test' set.

Model Used

- **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 100x100x3, for example, the video becomes a 4-D tensor of shape 100x100x3x30 which can be written as (100x100x30)x3 where 3 is the number of channels. Hence, deriving the analogy from 2-D convolutions where a 2-D kernel/filter (a square filter) is represented as (fxf)xc where f is filter size and c is the number of channels, a 3-D kernel/filter (a 'cubic' filter) is represented as (fxfxf)xc (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 (100x100x30) tensor.

Data Generator / Model Class

We have generated a Model Class which will [perform the following

- All the initialisations of paths, image properties and model properties
- Generator function for generating the data with labels
  - The images will be cropped to the size we want
  - The images with irregular dimension will be centre cropped on the x-axis equally on both sides
  - We save the model only when the validation loss decreases
  - Learning rate decreases when approaching the minima
- A generic train functions.
- Plotting the model
- An abstract method to define our own model by inheriting this class

Model Training Using CONV3D

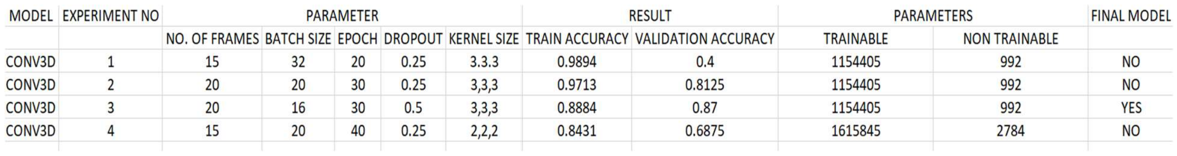
- Experimented with different model configurations and hyper-parameters and various iterations and combinations of batch sizes, image dimensions, etc.
- We used *Adam ()* as it led to improvement in model's accuracy by rectifying high variance in the model's parameters.
- We also made use of *Batch Normalization*, *pooling* and *dropout layers* when our model started to overfit.

Observation / Result Summary

- Run time increases when no. of trainable parameter increases
- The gap between train and validation accuracy decreases with increase in no. of frame size, batch size and dropout layer.

Please find the result summary below:

Here, the model 3 is showing the best performance and so it has been considered as final model



MODEL	EXPERIMENT NO	PARAMETER					RESULT		PARAMETERS		FINAL MODEL
		NO. OF FRAMES	BATCH SIZE	EPOCH	DROPOUT	KERNEL SIZE	TRAIN ACCURACY	VALIDATION ACCURACY	TRAINABLE	NON TRAINABLE	
CONV3D	1	15	32	20	0.25	3.3.3	0.9894	0.4	1154405	992	NO
CONV3D	2	20	20	30	0.25	3,3,3	0.9713	0.8125	1154405	992	NO
CONV3D	3	20	16	30	0.5	3,3,3	0.8884	0.87	1154405	992	YES
CONV3D	4	15	20	40	0.25	2,2,2	0.8431	0.6875	1615845	2784	NO

**** END OF THE DOCUMENT ****