# Neural Networks Project - Gesture Recognition

## 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.

## 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 - like 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).

## Data Generator

This is one of the most important part of the code. In the generator, we are going to pre-process the images as we have images of 2 different dimensions (*360 x 360* and *120 x 160*) 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.

## Data Pre-processing

- **Resizing and cropping of the images.**
  This was mainly done to ensure that the NN only recognizes the gestures effectively rather than focusing on the other 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 in an image.

- At the later stages for improving the model's accuracy, we have also made use of **data augmentation**, where we have **slightly rotated** the pre-processed images of the gestures in order to bring in more data for the model to train on and to make it more generalizable in nature as sometimes the positioning of the hand won't necessarily be within the camera frame always.

## NN Architecture development and training

- Experimented with different model configurations and hyper-parameters and various iterations and combinations of batch sizes, image dimensions, filter sizes, padding and stride length were experimented with. We also played around with different learning rates and *ReduceLROnPlateau* was used to decrease the learning rate if the monitored metrics (*val_loss*) remains unchanged in between epochs.
- We experimented with *Adam()* as it lead to improvement in model's accuracy by rectifying high variance in the model's parameters.

- *Early stopping* was used to put a halt at the training process when the *val_loss* would start to saturate / model's performance would stop improving.

## Observations

- As the Number of trainable parameters increase, the model takes much more time for training.
- **Batch size ∝ 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.
- Increasing the batch size greatly reduces the training time but this also has a negative impact on the model accuracy. This made us realise that there is always a trade-off here on basis of priority. If we want our model to be ready in a shorter time span, choose larger batch size else you should choose lower batch size if you want your model to be more accurate.
- *Data Augmentation* and *Early stopping* greatly helped in overcoming the problem of overfitting which our initial version of model was facing.
- *CNN+LSTM* based model with *GRU* cells had better performance than *Conv3D.* As per our understanding, this is something which depends on the kind of data we used, the architecture we developed and the hyper-parameters we chose.
- *Transfer learning* **boosted** the overall accuracy of the model. We made use of the MobileNet Architecture due to its lightweight design and high speed performance coupled with low maintenance as compared to other well-known architectures like VGG16, Alex Net, GoogleNet etc.
- For detailed information on the Observations and Inference, please refer the below Table.

## Table

| Experiment Number | Model | Result | Decision + Explanation | Total Parameters |
|---|---|---|---|---|
| **1** | **Conv3D** | **Training Accuracy: 99.3%<br>Validation Accuracy: 23%** | **Model is clearly overfitting with Batch Size = 40 and No. of Epochs = 15** | 1117061 |
| **2** | **Conv3D** | **Training Accuracy: 97.9%<br>Validation Accuracy: 21%** | **Model is still overfitting after adding dropout layers with Batch Size = 40 and No. of Epochs = 15** | 3638981 |
| **3** | **Conv3D** | **Training Accuracy: 72.0%<br>Validation Accuracy: 26%** | **Reduced filter size to (2,2,2) and image res to 120 x 120, - Batch Size = 30 and No. of Epochs = 25** | 1762613 |
| **4** | **Conv3D** | **Training Accuracy: 84.0%<br>Validation Accuracy: 27%** | **Reduction of number of parameters, batch Size = 20 and No. of Epochs = 15** | 696645 |

| | | | | |
|---|---|---|---|---|
| **5** | **CNN +LSTM Model** | **Training Accuracy: 93.67% Validation Accuracy: 48%** | **Cases of overfitting, augmented the data with slight rotation as well** | 1657445 |
| **6** | **CNN + GRU** | **Training Accuracy: 95.25% Validation Accuracy: 78%** | **Model with augmentation seemed to provide good accuracy but overfitting persists** | 2573925 |
| **7** | **Transfer Learning with GRU MobileNet** | **Training Accuracy: 99.63% Validation Accuracy: 99%** | **Model has provided the best training and validation accuracy** | 3693253 |

## Conclusion

After doing all the experiments, we finalized **Model – Transfer Learning with GRU**, which performed well.

**Reason:**

- Training Accuracy: 99.85%, Validation Accuracy: 99%
- Number of Parameters are quite high as compared to other models which were clearly overfitting.
- Although high number of parameters require much time on training but if the model is giving outstanding accuracy on both training and validation set, it is worth to go with that model.