# 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

Each video is a sequence of 30 frames (or images)

### Google drive link to download the best model:

Since the model size is huge hence we are uploading the model *.h5 file to Google Drive

https://drive.google.com/file/d/1r3wvw7z3bBl6DNuxRwe-0hSp7i2O7oiZ/view?usp=sharing

# 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.

# 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:

**Model Description:**

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 parts of the code. In the generator, we are going to pre-process the images as we have images of different dimensions (*50 x 50, 70 x 70* and *120 x 120*) 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*.** This was mainly done to ensure that the NN only recognizes the gestures effectively.
- ***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.

## 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 *SGD()* and *Adam()* optimizers but went forward with *SGD* as it lead to improvement in model's accuracy by rectifying high variance in the model's parameters. Played with multiple parameters of the SGD like decay_rate, starting learning rate.
- We also made use of *Batch Normalization*, *pooling* and *dropout layers* when our model started to overfit, this could be easily witnessed when our model started giving poor validation accuracy in spite of having good training accuracy.
- *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

- It was observed that 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 (e.g 64 on cloud platform and 25 in local GPU),* 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 ( NVIDIA GTX 1650 and RTX 5000 in Jarvis Labs).
- We also found out that the middle frames gives us most of the information and because the train images were chosen so carefully, data augmentation was not required though left-right flipping and zoom, slight rotation could have been done.
- 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.
- *Conv3D* had better performance than *CNN+LSTM* based model with *GRU* cells. 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 light weight design and high-speed performance coupled with low maintenance as compared to other well-known architectures like VGG16, AlexNet, GoogleNet etc.

For detailed information on the Observations and Inference, please refer

# Model Overview

| Model Name | Model Type | Number of parameters | Frames/ Batch_size | Epochs | Highest Validation accuracy | Corres-ponding Training accuracy | Observations |
|---|---|---|---|---|---|---|---|
| conv_3d1_model | Conv3D | 18,615,813 | 10/64 | 15 | - | - | Due to higher batch size, the GPU is not able to load the entire model and fit failed in OOM (out of memory). So we will proceed to the next one with less batch size. |
| conv_3d2_model | Conv3D | 6,557,189 | 6/20 | 20 | 93% | 95% | Training and validation Accuracy are good so that we can conclude that with above set of parameters model is giving good results. Frame shape used 50,50. Next we can try to increase the frames and batch size. |
| conv_3d3_model | Conv3D | 6,557,189 | 10/30 | 20 | 82% | 92% | Though the parameters remain same, we have tried with 10 frames per video and 30 batch size of videos in hopes of higher learning, the shape remains 50 * 50, here both training and validation accuracy has suffered. Let's try resizing to 120*120 in the next iteration. |
| conv_3d4_model | Conv3D | 18,615,813 | 10/50 | 25 | 87% | 95% | With the increase of resize shape and the batch, we see there is a slight improvement in both training and validation but still it shows somewhere the signs of overfitting. We can try some shape in the middle: 70 * 70 and check |

| Model Name | Model Type | Number of parameters | Frames/ Batch_size | Epochs | Highest Validation accuracy | Corres-ponding Training accuracy | Observations |
|---|---|---|---|---|---|---|---|
| | | | | | | | with more frames in the hope of better results. |
| conv_3d5_model | Conv3D | 14,683,653 | 18/50 | 25 | 85% | 95% | With shape 70 * 70 and 18 frames out of 30, the accuracy is sitting somewhere in the same bracket like the last 2 iterations where there is high difference between the training accuracy and the validation accuracy. |

**Conclusion**: With the lower frame shape and one fifth of the frames, we were able to get much higher accuracy in both training and validation and faster compute.

# Time Distributed (CNN + LSTM/GRU)

| Model Name | Model Type | Number of parameters | Frames/ Batch_size | Epochs | Highest Validation accuracy | Corres-ponding Training accuracy | Observations |
|---|---|---|---|---|---|---|---|
| CNN_RNN_1 | TimeDistributed | 3,807,589 | 18/50 | 25 | 60% | 76% | We tried a basic CNN 2d with RNN LSTM and we didn't get good accuracy. Model not learning much info in training , not performing well in validation. We took image shape 70 *70, using 18 frames. We will try with our best shape till now 50*50. |
| CNN_RNN_2 | TimeDistributed | 2,234,725 | 10/20 | 20 | 67% | 99% | With the new shape and the frames reduced, and keeping the same layers we got awesome accuracy but the validation result didn't improve much. Let's replace LSTM with GRU and check. Also we will tweak the layers to reduce the overfitting. |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| CNN_RNN_3 | TimeDistributed | 336,741 | 18/20 | 20 | 75% | 94% | With the same shape 50*50, we have got better validation but still there is a overfit visible between the training and the val accuracy. |

## Transfer Learning Models (CNN + RNN)

Mobilenet model is considered as its parameter size is less compared to Inception and Resnet models

| Model Name | Model Type | Number of parameters | Frames/ Batch_size | Epochs | Highest Validation accuracy | Corres-ponding Training accuracy | Observations |
|---|---|---|---|---|---|---|---|
| MobileNet Transfer learning + GRU | Transfer Learning | 3,693,253 | 18/5 | 15 | 98% | 99% | Usage of transfer learning with all the layers trainable=True, we just added few layers and GRU which gave us the almost perfect score we can think of, it is almost learning everything in training and validation. |

**Conclusion**: Transfer learning model worked best for us with all the layers trainable, we can see the conv3d played well from the time distributed one which are enough to test on the image set.

**Final model:** conv_3d2_model

## Further suggestions for improvement:

- **Using Transfer Learning**: Using a pre-trained *ResNet50/ResNet152/Inception V3* to identify the initial feature vectors and passing them further to a *RNN* for sequence information before finally passing it to a softmax layer for classification of gestures. (This was attempted but other pre-trained models couldn't be tested due to lack of time and disk space in the nimblebox.ai platform.)
- **Using GRU:** A *GRU* model in place of *LSTM* appears to be a good choice. Trainable Parameters of a *GRU* are far less than that of a *LSTM*. Therefore, would have resulted in faster computations. However, its effect on the validation accuracies could be checked to determine if it is actually a good alternative over LSTM.
- **Deeper Understanding of Data:** The video clips were recorded in different backgrounds, lightings, persons and different cameras where used. Further exploration on the available images could give some more information about them and bring more diversity in the dataset.

This added information can be exploited in favour inside the generator function adding more stability and accuracy to model.

- **Tuning hyperparameters:** Experimenting with other combinations of hyperparameters like, activation functions (*ReLU, Leaky ReLU, mish, tanh, sigmoid*), other optimizers like *Adagrad()* and *Adadelta()* can further help develop better and more accurate models. Experimenting with other combinations of hyperparameters like the *filter size, paddings, stride_length, batch_normalization, dropouts* etc. can further help improve performance.