# Gesture Recognition – Deep learning 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 recognize 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

# Understanding the Dataset:

The training data consists of a few hundred videos which are categorised into five classes. Each video is typically 2-3 seconds long. Each video 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 which is very 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 images of the 'train' folder to predict the action performed in each sequence of 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 (CNN, RNN) suggested for analyzing videos using deep learning:

**Model Description:**

1. **CNN + RNN architecture**

The conv2D network will extract a feature vector for each image and will generate a sequence of these feature vectors for all images which is then fed to an RNN-based network architecture. The RNN output is fed to a dense layer (hidden layer) followed by a regular softmax activation function (for a classification problem such as this one).

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

3D convolutions are a natural extension to the 2D convolution. In 2D convolution network, we move the filter in two directions (x and y) and in 3D convolution network, we move the filter in three directions (x, y and z). In this case, the input to a 3D convolution is a video (which is a sequence of 30 frames of RGB images).

If we assume that the shape of each image is 100 x 100 x 3(100 is length, 100 is width and number of channels is 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 and 30 is the number of frames. 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 '3Dconvolve' on each of the three channels of the (100 x 100 x 30) tensor.

# Data Generator:

This is one of the most important parts of the code. We have images of different dimensions (*50 x 50, 70 x 70* and *120 x 120*) which we are going to preprocess in the generator. Also, we create a batch of video frames. The generator will take a batch of videos as input without any error. We are performing the steps like cropping/resizing and normalization through this generator.

# Data Pre-processing:

* **Resizing**: This was done for ensuring that the NN only recognizes the gestures effectively.
* **Normalization of the images**: We do Normalization of the RGB values of an image to get rid of distortions caused by lights and shadows in an image.

# CNN Architecture development and training:

* In this step, we experimented with different model configurations and hyper-parameters combination. We tried various iterations and combinations of batch sizes, frame and number of epochs. We have taken Categorical Cross Entropy as the loss function and Categorical Entropy as Performance metric. We experimented with **Adam**optimizers
* We also made use of **Batch Normalization, pooling, and dropout layers** when our model started to overfit .

# Observations:

* It was observed that with the increase of Number of trainable parameters, the models start taking much more time for training.
* **Batch size Vs GPU memory:** A large batch size can throw **GPU Out of memory error** , and thus here we had to apply different batch sizes till we were able to arrive at an optimal value of the batch size which our GPU could support (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 leads to decrease in 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 or for more accuracy we can choose smaller batch size.
* **Conv3D** had better performance than **CNN2D+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.

|  |  |  |
| --- | --- | --- |
|  | **Conclusion**: By analysing above models, we observe that, with the lower frame shape and  One fifth of the frames, we were able to get much better accuracy in both training and validation. It also computes faster. |  |

# Model Overview: Conv3D:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Model Name** | **Model Type** | **Number of parameters** | **Frames/ Batch\_size** | **Epochs** | **Best Validation accuracy** | **Corresponding Training accuracy** | **Observations** |
| conv\_3d1\_model | Conv3D | 9,006,245 | 30/25 | 15 | 62.00% | 84.33% | Training and validation Accuracy are good but not up to the mark. Next we can try to increase the batch size and slight decreasing the frames. |
| conv\_3d2\_model | Conv3D | 4,050,085 | 20/35 | 12 | 36.66% | 61.4% | Training accuracy is bit well but the validation Accuracy is too low. Next we can try to increase the frames and batch size. |
| conv\_3d3\_model | Conv3D | 9,006,245 | 30/40 | 15 | 53.33% | 85.75% | The difference between training accuracy and testing increased very much showing overfitting of the model. |

**Time Distributed (CNN + LSTM):**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Model Name** | **Model Type** | **Number of parameters** | **Frames/ Batch\_size** | **Epochs** | **Highest Validation accuracy** | **Corresponding Training accuracy** | **Observations** |
| CNN\_RNN\_1 | TimeDistributed | 3,084,133 | 20/15 | 15 | 57.14% | 44.45% | We tried a basic CNN 2d with RNN LSTM and we didn’t get good accuracy and sees overfit. Model not learning much info in training, not performing well in validation also.  Increasing the Batch size of the model and trying again |
| CNN\_RNN\_2 | TimeDistributed | 3,084,133 | 20/25 | 15 | 69.99% | 81.76% | With the increase in batch size we can see improve in training accuracy and validation accuracy but the validation accuracy didn’t improve much. Still, this seems to be overfit. |
| CNN\_RNN\_3 | TimeDistributed | 3,084,133 | 20/35 | 15 | 40.00% | 61.40% | With the increase in batch size we can’t see improvement in training accuracy and validation accuracy. |

# Conclusion:

The Model built with Time Distributed (CNN + LSTM) gave better results compared to all the other models and also the model has a smaller number of parameters compared to other model