## **Deep Learning 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 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**

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 SGD() and Adam() optimizers but went forward with Adam() as it lead to improvement in model’s accuracy by rectifying high variance in the model’s parameters. We were unsupportive of experimenting with Adagrad() and Adadelta() due to the limited computational capacity as these take a lot of time to converge because of their dynamic learning rate functionalities.
* 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 inspite 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**

* As the number of trainable parameters increase, the model takes much more time for training
* Batch size is directly proportional to 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.
* Data Augmentation and Early stopping greatly helped in overcoming the problem of overfitting which our initial version of model was facing.
* Transfer learning boosted the overall accuracy of the model. We made use of the MobileNet Architecture due to it’s light weight design and high speed performance coupled with low maintenance as compared to other well-known architectures like VGG16, AlexNet, GoogleNet etc.

The detailed information is available in pybook :

<https://github.com/MrChetanDesai/Deep_Learning_Project_Gesture_Recognition>