## **Deep Learning Course Project- Gesture Recognition**

**Soundharya Sukumar – Group Facilitator**

**Nigama Veena**

**Problem Statement**

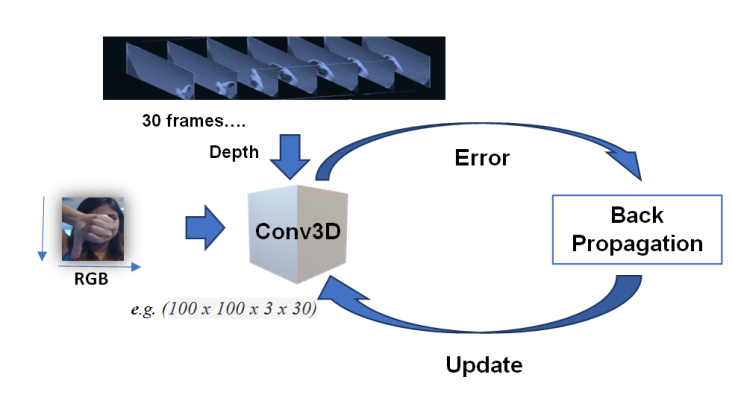
As a data scientist at a home electronics company which manufactures state of the art smart televisions. We want to develop a cool feature in the smart-TV that can recognise five different gestures performed by the user which will help users control the TV without using a remote.

* 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.

**Two types of architectures suggested for analysing videos using deep learning:**

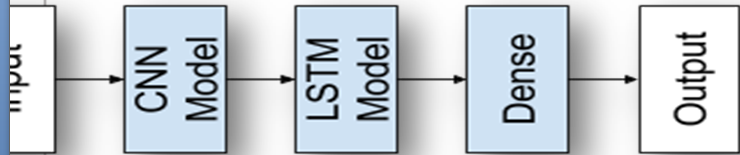
1. **3D Convolutional Neural Networks (Conv3D)**

*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



1. **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.

**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,* 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 Tesla K80 GPU with 12GB memory provided by nimblebox.ai platform.)
* 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.
* For detailed information on the Observations and Inference, please refer Below Table.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **MODEL** | **EXPERIMENT** | **RESULT** | **DECISION + EXPLANATION** | **PARAMETERS** | |
| **CONV3D** | **1** | **Training Accuracy : 0.52**  **Validation Accuracy : 0.32** | **The model is performing better on the validation data than on the training data. This indicates that the model is too simple and has lesser that required training parameters** | | **Total params: 317,669**  **Trainable params: 317,557** |
| **2** | **Training Accuracy : 0.45**  **Validation Accuracy : 0.55**  **Reducing a image size from 100 to 50** | **Scaling to smaller images work better** | | **Total params: 96,485**  **Trainable params: 96,373** |
| **3** | **Training Accuracy : 0.38**  **Validation Accuracy : 0.59**  **Reducing the image size from 50 to 20** | **Scaling from 50 to 20 reduced the training and validation accuracy** | | **Total params: 30,949**  **Trainable params: 30,837** |
| **4** | **Training Accuracy : 0.3012**  **Validation Accuracy : 0.3010**  **Decreasing the batch size from 8 to 4 Changing the optimizer to Adam** | **Reducing the batch size has reduced overfitting** | | **Total params: 96,485**  **Trainable params: 96,373** |
| **5** | **Training Accuracy : 0.22**  **Validation Accuracy : 0.21**  **Reducing the number of frames from 15 to 10** | **The training and validation accuracy has increased after reducing the number of frames from 15 to 10** | | **Total params: 96,485**  **Trainable params: 96,373** |
| **6** | **Training Accuracy : 0.58 Validation Accuracy : 0.71**  **Reducing the dropout rate from 0.5 to 0.25** | **A drop out rate of 0.25 gives up a better training and validation accuracy** | | **Total params: 96,485**  **Trainable params: 96,373** |
| **7** | **Training Accuracy : 0.56 Validation Accuracy : 0.69**  **Increasing the number of epochs from 25 to 35** | **After 35 epochs the training and validation accuracy changes to 56% and 69% respectively** | | **Total params: 96,485**  **Trainable params: 96,373** |