Deep Learning Course Project- Gesture Recognition

# Problem Statement

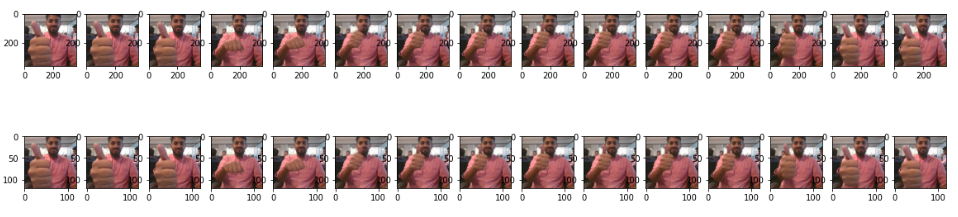
As a data scientist at a home electronics company which manufactures state of the art smart televisions. We want to develop a new feature in the smart-TV that can recognize five different gestures performed by the user which will help users control the TV without using a remote.

* Thumbs up : Increase the volume.
* Thumbs down : Decrease the volume.
* Left swipe : 'Jump'’s backwards 10 seconds.
* Right swipe : 'Jump'’s forward 10 seconds.
* Stop : Pause the movie.

**Here’s the data:** [**https://drive.google.com/drive/u/0/folders/1NDZfU-RkyE09hHW4oNKCfpRFPyGU2VDa**](https://drive.google.com/drive/u/0/folders/1NDZfU-RkyE09hHW4oNKCfpRFPyGU2VDa)

# 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 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.

## 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).

The following table represents the results obtained during the model building.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Exp # | Model | Result | Decision + Explanation | |
| 1 | Conv3D |  | Total params: 12,904,581  Trainable params: 12,904,581  Non-trainable params: 0  **Accuracy:**  Train acc = 0.3768  Validation acc = 0.3333  **Loss:**  Train loss = 9.0786  Validation loss = 79.4799  Batch size = 30  Epochs = 30 | Model seems to be less accurate and the loss between the train and validation is huge. |
| 3 | Conv3D | Negative Dimension Error. | The new CNN kernel sizes are not compatible with the output | |
| 4 | Conv3D |  | Total params: 13,781  Trainable params: 13,589  Non-trainable params: 192  **Accuracy:**  Train acc = 0.866  Validation acc = 0.633  **Loss:**  Train loss = 0.3273  Validation loss = 0.6520  Batch size = 30  Epochs = 30 | The model has very losses, but the accuracy declined after 20th epoch. |
| 5 | Conv3D |  | Total params: 8,317,701  Trainable params: 8,314,757  Non-trainable params: 2,944  **Accuracy:**  Train acc = 0.9551  Validation acc = 0.5667  **Loss:**  Train loss = 0.0221  Validation loss = 1.0179  Batch size = 30  Epochs = 30 | Model seems to be overfitted, as the gap between the train and validation is more.  Can increase the epoch to look for the improvement in accuracy.  In the next trail, the number of epochs is increased to 50 before we opt for dropout. |
| 6 | Conv3D |  | Total params: 8,317,701  Trainable params: 8,314,757  Non-trainable params: 2,944  **Accuracy:**  Train acc = 0.3768  Validation acc = 0.3333  **Loss:**  Train loss = 9.0786  Validation loss = 79.4799  Batch size = 30  Epochs = 50 | Model doesn’t seems to improve and the loss between the train and validation is huge.  In the next experiment, dropout would used. |
| 7 | Conv3D |  | Total params: 8,317,701  Trainable params: 8,314,757  Non-trainable params: 2,944  **Accuracy:**  Train acc = 0.9609  Validation acc = 0.6250  **Loss:**  Train loss = 0.0300  Validation loss = 0.6250  Batch size = 30  Epochs = 30 | WE still find overfitting with the dropout value of 0.2.  Let’s change the drop out from 0.2 to 0.5 in the next experiment. |
| 8 | Conv3D |  | Total params: 8,317,701  Trainable params: 8,314,757  Non-trainable params: 2,944  **Accuracy:**  Train acc = 0.9522  Validation acc = 0.7583  **Loss:**  Train loss = 0.0713  Validation loss = 1.0839  Batch size = 30  Epochs = 30 | Model seems to improve in accuracy inline amongst the train and validation results.  However, the loss (gap) is more between the train and validation set.  Therefore the previous ‘drop-out = 0.2’ is further used to remove CNN layer. |
| 9 | Conv3D |  | Total params: 22,732,549  Trainable params: 22,730,629  Non-trainable params: 1,920  **Accuracy:**  Train acc = 0.9522  Validation acc = 0.7583  **Loss:**  Train loss = 0.0713  Validation loss = 1.0839  Batch size = 30  Epochs = 30 | The model is still over fitted. Let us opt for Global average pooling instead of flatten. |
| 10 | Conv3D |  | Total params: 712,453  Trainable params: 710,533  Non-trainable params: 1,920  **Accuracy:**  Train acc = 0.9347  Validation acc = 0.7621  **Loss:**  Train loss = 0.022  Validation loss = 0.812  Batch size = 30  Epochs = 30 | The model is slightly acceptable and the training and validation scores are good. The model has 710,533 trainable parameters. Let’s try architectures too. |
| 11 | Time distributed  +GRU |  | Total params: 99,845  Trainable params: 99,269  Non-trainable params: 576  **Accuracy:**  Train acc = 0.9522  Validation acc = 0.7583  **Loss:**  Train loss = 0.0713  Validation loss = 1.0839  Batch size = 30  Epochs = 30 | The model is working quite well on validation dataset with less trainable parameters (99,269), Lets add some drop outs after each layer, so that both train and validation accuracies will be closure. |
| 12 | Time distributed  +GRU |  | Total params: 129,477  Trainable params: 128,517  Non-trainable params: 960  **Accuracy:**  Train acc = 0.884  Validation acc = 0.5417  **Loss:**  Train loss = 0.2302  Validation loss = 1.0701  Batch size = 30  Epochs = 30 | The model accuracy could further be enhanced by GRU with a plain Dense Layer Network and some Global Avg Pooling |
| 13 | Time distributed  +Dense |  | Total params: 129,477  Trainable params: 128,517  Non-trainable params: 960  **Accuracy:**  Train acc = 0.9043  Validation acc = 0.7833  **Loss:**  Train loss = 0.1737  Validation loss = 0.5595  Batch size = 30  Epochs = 30 | More number of trainable parameters are available. To check for the betterment in the results, let us use time distribution with ConvLSTM2D |
| 14 | Time distributed  +ConvLSTM2D |  | Total params: 13,781  Trainable params: 13,589  Non-trainable params: 192  **Accuracy:**  Train acc = 0.8580  Validation acc = 0.7833  **Loss:**  Train loss = 0.2873  Validation loss = 0.5764  Batch size = 30  Epochs = 30 | This model seems to be the best of the experiments performed with closing gap between the train and validation set for both accuracy and loss. |