**Gesture Recognition**

**Problem Statement**

A home electronics company which manufactures state of the art smart televisions 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.

**Here’s the data:** <https://drive.google.com/uc?id=1ehyrYBQ5rbQQe6yL4XbLWe3FMvuVUGiL>

**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.

Each row of the CSV file represents one video and contains three main pieces of information - the name of the subfolder containing the 30 images of the video, the name of the gesture and the numeric label (between 0-4) of the video.

**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.

**Data Generator**

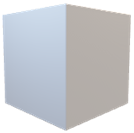
This is one of the most important part of the code. In the generator, we are going to pre-process the images as we have images of 2 different dimensions (*360 x 360* and *120 x 160*) as well as create a batch of video frames. The generator should be able to take a batch of videos as input without any error. Steps like cropping, resizing and normalization should be performed successfully.

**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

**RGB**



**Conv3D**

**Back**

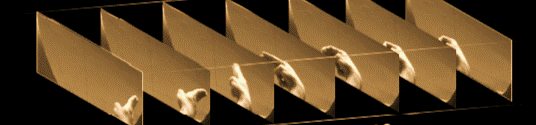
**Propagation**

**Update**

**Error**

**30 frames**

**Depth**



***e****.g****.*** *(100 x 100 x 3 x 30)*

**Figure 1: A simple representation of working of a 3D-CNN**

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).

A close up of a sign

Description automatically generated

**Figure 2: A simple representation of an ensembled CNN+LSTM Architecture**

**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.

**Note: It was taken into consideration that we don’t rotate images to a greater extent as this would change the meaning of the gestures completely**

**Experimented Procedure**

* 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 led to improvement in model’s accuracy by rectifying high variance in the model’s parameters.
* 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.

|  |  |  |  |
| --- | --- | --- | --- |
| **Experiment Number** | **Model** | **Result** | **Decision + Explanation** |
| 1 | Conv3D | Training Accuracy: 0.34  Validation Accuracy: 0.31 | No overfitting occurred, as we have used dropout regularization. |
| 2 | CNN-RNN | Training Accuracy: 0.84  Validation Accuracy: 0.54 | As training accuracy is high compared to validation accuracy, which led to overfitting  It can be overcome by data augmentation. |
| 3 | Conv3D with Augmentation | Training Accuracy: 0.25  Validation Accuracy: 0.22 | In our case the accuracy decreased slightly with data augmentation. |
| 4 | CNN-RNN with Augmentation | Training Accuracy: 0.60  Validation Accuracy: 0.44 | Overfitting is overcome with the help of data augmentation. |

# Conclusions

Following are the conclusions:

* Initially we build a simple **Conv-3D** model, we got an training accuracy of around 32% and validation accuracy of around 31%. There is no clear sign of overfitting.
* We have seen that, with the help of **data augmentation method**, there was no improvement in **Conv-3D** model compared to non-augmented data.
* Secondly, we build a **CNN-RNN** model, we got a training accuracy of around 80% and validation accuracy of around 54%. It is a clear sign of overfitting.
* We have seen that, with the help of **data augmentation method**, there was improvement in **CNN-RNN** model compared to non-augmented data. In our case it reduced the overfitting, but decreased our training accuracy (approx. 60%) and validation accuracy(approx. 43%)

We can say that in our casa the **CNN-RNN** model with data augmentation gives better results compare to other models.