**Gesture Recognition**

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:

1. Thumbs up: Increase the volume

2. Thumbs down: Decrease the volume

3. Left swipe: 'Jump' backwards 10 seconds

4. Right swipe: 'Jump' forward 10 seconds

5. Stop: Pause the movie

**Understanding the Dataset**

1. The training data consists of a few hundred videos categorised into one of the five classes.

2. Each video (typically 2-3 seconds long) is divided into a sequence of 30 frames(images).

3. 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.

**Model Description**

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 '3Dconvolve' 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 parts of the code. In the generator, we are going to pre-process the images as we have images of different dimensions (*50 x 50, 70 x 70, 100 x 100* and *120 x 120*) 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**: This was mainly done to ensure that the NN only recognizes the gestures effectively.
* **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.

**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. 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 SGDas it lead to improvement in model’s accuracy by rectifying high variance in the model’s parameters. Played with multiple parameters of the SGD like decay\_rate, starting learning rate.
* 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.
* **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**

|  |  |  |  |
| --- | --- | --- | --- |
| **Experiment Number** | **Model** | **Result** | **Decision + Explanation** |
| **1** | **Conv3D**  **Epochs = 10, batch\_size = 12, shape = (120,120), frames = 10** | **Training\_accuracy: 0.8989**  **Val\_accuracy: 0.8200** | **We got good accuracy. lets try to increase the batch\_size, no. of epocs and frames. We have resize the image.** |
| **2** | **Conv3D**  **Epochs = 15, batch\_size = 16, shape = (100,100), frames = 12** | **Training\_accuracy: 0.9849**  **Val\_accuracy: 0.8000** | **The model accuracy has increased but validation accuracy drops clear sign of overfitting. lets try to increase the batch\_size, no. of epocs and frames. We have resize the image.** |
| **3** | **Conv3D**  **Epochs = 20, batch\_size = 32, shape = (70,70), frames = 12** | **Training\_accuracy: 0.9683**  **Val\_accuracy: 0.7600** | **The model accuracy has dropped and validation accuracy also drops clear and overfitting is still there. lets try to increase the batch\_size, no. of epocs and frames. We have resize the image.** |
| **4** | **Conv3D**  **Epochs = 20, batch\_size = 32, shape = (70,70), frames = 18** | **Training\_accuracy: 0.9925**  **Val\_accuracy: 0.8200** | **The model accuracy and validation accuracy has increased but sign of overfitting is still there. lets try to increase the batch\_size, no. of epocs and frames. We have resize the image.** |
| **5** | **Conv3D**  **Epochs = 20, batch\_size = 64, shape = (50,50), frames = 12** | **Training\_accuracy: 0.9759**  **Val\_accuracy: 0.7400** | **There is drop in training and val accuracy and overfitting is still there.** |
| **6** | **CONV2D + GRU**  **Epochs = 20, batch\_size = 64, shape = (70,70), frames = 12** | **Training\_accuracy: 0.7768**  **Val\_accuracy: 0.6000** | **There is drop in training and val accuracy and overfitting is still there.** |