Neural Network Project

Gesture Recognition

# Problem Statement:

**Imagine you are working as a data scientist at a home electronics company which manufactures state of the art smart televisions. You 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:

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

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.

to train a model on the 'train' folder which performs well on the 'val' folder as well (as usually done in ML projects). We have withheld the test folder for evaluation purposes - your final model's performance will be tested on the 'test' set.

In order to get the data on the storage, perform the following steps in order

1. Open the terminal.
2. Go down https://drive.google.com/uc?id=1ehyrYBQ5rbQQe6yL4XbLWe3FMvuVUGiL
3. unzip Project\_data.zip

# Two Architectures: CNN-RNN Stack and 3D Convs

For analysing videos using neural networks, two types of architectures are used commonly. One is the standard CNN + RNN architecture in which you pass the images of a video through a CNN which extracts a feature vector for each image, and then pass the sequence of these feature vectors through an RNN. This is something you are already familiar with (in theory).

## **Convolutions + RNN**

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).

## **3D Convolutional Network, or 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 100x100x3, for example, the video becomes a 4-D tensor of shape 100x100x3x30 which can be written as (100x100x30)x3 where 3 is the number of channels. Hence, deriving the analogy from 2-D convolutions where a 2-D kernel/filter (a square filter) is represented as (fxf)xc where f is filter size and c is the number of channels, a 3-D kernel/filter (a 'cubic' filter) is represented as (fxfxf)xc (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 (100x100x30) tensor.

# Understanding Generators

In most deep learning projects, we need to feed data to the model in batches. This is done using the concept of generators.

Creating data generators is probably the most important part of building a training pipeline. Although libraries such as Keras provide built-in generator functionalities, they are often restricted in scope, and we have to write your own generators from scratch.

In this project, we have created our own python generator which is a python generator function. The generator yields a batch of data and 'pauses' until the fit\_generator calls next(). Writing custom generator function works with the yield statement. A custom generator would help you in creating a batch of any kind of data, for example, text data which is not readily available with keras.

We have experimented our model with the following points:

1. number of images to be taken per video/sequence
2. cropping the images
3. resizing the images
4. normalizing the images

# Models Experimented during the development of the Gesture Recognition project:

## Input Model

## First Model: 3D Conv Model

|  |  |  |
| --- | --- | --- |
| No of Images | : | 30 |
| Image Size | : | 160x160 |
| No of epoch | : | 30 |
| Batch Size | : | 40 |
| Observation | : | We had hit the limit on memory resources with image resolution of 160x160 with 30 frames and batch\_size of 40...we get the below error ResourceExhaustedError: OOM when allocating tensor with shape[40,16,30,160,160] |
| Next Action | : | We need to reduce the number of the layers and images as well size of the images. |

## Second Model: 3D Conv Model

|  |  |  |
| --- | --- | --- |
| No of Images | : | 18 |
| Image Size | : | 64x64 |
| No of epoch | : | 20 |
| Batch Size | : | 30 |
| No of params | : | 144421 |
| Train Time | : | 440 sec |
| Observation | : | train\_accuracy of 44 % and val\_accuracy of 44 % . |
| Next Action | : | As we see from the above experiments image resolution and number of frames in sequence have more impact on accuracy than batch\_size and epoch for Conv model. So experimentations are carried with batch size fixed around 15-40 and changing the resolution and number of image per sequence based on the device memory constraints. Still model is inderfitting,so increasing the dense nuerons to 128 from 64 because few neurons in the hidden layers to properly detect the signals in a complicated dataset causes this. Also we are reducing the cropping to 3 % from 10 % so we dont loose any data.Increase image size to 84 from 64. |

## Third Model: 3D Conv Model

|  |  |  |
| --- | --- | --- |
| No of Images | : | 18 |
| Image Size | : | 84x84 |
| No of epoch | : | 20 |
| Batch Size | : | 30 |
| No of params | : | 288101 |
| Train Time | : | 480 sec |
| Observation | : | train\_accuracy of 42 % and val\_accuracy of 41 %. |
| Next Action | : | Increase the amount of trainable data. We are further increasing image size to 100x100 for next model.Still model is underfitting,so increasing the dense nuerons to 256 from 128.Lets increase number of epoch and batch size as well |

## Fourth Model: 3D Conv Model

|  |  |  |
| --- | --- | --- |
| No of Images | : | 18 |
| Image Size | : | 100x100 |
| No of epoch | : | 30 |
| Batch Size | : | 50 |
| No of params | : | 651109 |
| Train Time | : | 750 sec |
| Observation | : | train\_accuracy of 58 % and val\_accuracy of 41 %. |
| Next Action | : | The model has started to overfitting which is evident from grap between the train and val accuracy.Model not trainable as a lot of parameters. Reduce the size of the image to 84 from 100 and reduce number of nuerons in the network to 64 from 256. Also the accuracy is still a problem here. Increase the amount of trainable data: increase input image sequence from existing 18 to full dataset 30. |

## Fifth Model: 3D Conv Model

|  |  |  |
| --- | --- | --- |
| No of Images | : | 30 |
| Image Size | : | 84x84 |
| No of epoch | : | 30 |
| Batch Size | : | 30 |
| No of params | : | 181285 |
| Train Time | : | 1420 sec |
| Observation | : | train\_accuracy of 79% and val\_accuracy of 66 %. |
| Next Action | : | The model is still overfitting but it has improved accuracy. Also the training time has increased to 1420 sec due to increase in number of input images to read.Try other models as Conv3D not giving desired accuracy for given data-set. |

## Sixth Model: LSTM Model

|  |  |  |
| --- | --- | --- |
| No of Images | : | 30 |
| Image Size | : | 120x120 |
| No of epoch | : | 20 |
| Batch Size | : | 20 |
| No of params | : | 1657445 |
| Train Time | : | 900 sec |
| Observation | : | train\_accuracy of 91% and val\_accuracy of 74 %. |
| Next Action | : | The base hybrid model has achieved the better accuracy than all the Conv models that we have experimented, although the model is overfitting here due to large number of parameters 1657445 in the model. |

## Seventh Model: with CNN-LSTM

|  |  |  |
| --- | --- | --- |
| No of Images | : | 30 |
| Image Size | : | 160x160 |
| No of epoch | : | 20 |
| Batch Size | : | 20 |
| No of params | : | 3754597 |
| Train Time | : | 1100 sec |
| Observation | : | train\_accuracy of 85% and val\_accuracy of 80%. |
| Next Action | : | The model is not overfitting anymore but its very complex model right now with too many paramaeter. Lets reduce the image size to 120 from 160 and number of neurons in the network for this.Also, the accuracy can be further improved. |

## Eight Model: with CNN-LSTM

|  |  |  |
| --- | --- | --- |
| No of Images | : | 30 |
| Image Size | : | 120x120 |
| No of epoch | : | 25 |
| Batch Size | : | 20 |
| No of params | : | 1287989 |
| Train Time | : | 2500 sec |
| Observation | : | train\_accuracy of 90% and val\_accuracy of 79%. |
| Next Action | : | The model has performance has improved marginally but it has started to overfit again with more than 10% difference between train and val accuracy. Lets increase the dense nuerons to see if increasing the network parameters helps. |

## Nineth Model: with CNN-LSTM

|  |  |  |
| --- | --- | --- |
| No of Images | : | 30 |
| Image Size | : | 120x120 |
| No of epoch | : | 25 |
| Batch Size | : | 20 |
| No of params | : | 1702645 |
| Train Time | : | 2500 sec |
| Observation | : | train\_accuracy of 87% and val\_accuracy of 82%. |
| Next Action | : | The models performance is improved and its not overfitting as training and val loss is comparable. But the network is heavy right now with these many network paramaeters and 2500 sec training time. Lets try experimenting with some other hybrid models. An LSTM has 4 gates, while GRU has 3 gates. Using GRU will significantly reduce the training times as it needs to compute values for 3 gates and its performance is at par with the LSTMs. |

## Model10: LSTM with GRU

|  |  |  |
| --- | --- | --- |
| No of Images | : | 30 |
| Image Size | : | 120x120 |
| No of epoch | : | 20 |
| Batch Size | : | 20 |
| No of params | : | 2573541 |
| Train Time | : | 900 sec |
| Observation | : | train\_accuracy of 94% and val\_accuracy of 79%. |
| Next Action | : | This model has decent val accuracy but it is clearly overfitting here. Lets try other to use pre-trained models for our own problem here using the Transfer Learning. |

## Model11: Transfer Learning

|  |  |  |
| --- | --- | --- |
| No of Images | : | 18 |
| Image Size | : | 120x120 |
| No of epoch | : | 5 |
| Batch Size | : | 20 |
| No of params | : | 3840453 |
| Train Time | : | 600 sec |
| Observation | : | train\_accuracy of 91% and val\_accuracy of 55%. |
| Next Action | : | The training time is less here. The model is again overfitting here with poor validation accuracy performance. We are not training the mobilenet weights and we see validation accuracy is very poor. Let's train them as well and observe if there is performance improvement. |

## Model12: Transfer Learning with GRU

|  |  |  |
| --- | --- | --- |
| No of Images | : | 18 |
| Image Size | : | 120x120 |
| No of epoch | : | 5 |
| Batch Size | : | 20 |
| No of params | : | 3840453 |
| Train Time | : | 990 sec |
| Observation | : | train\_accuracy of 98% and val\_accuracy of 95%. |
| Next Action | : | This mode is selected as the final model because it has very high accuracy of 95 % for validation data. |

As we have seen the last Model12 has given us the best training accuracy as well as the best test accuracy. Hence, marking the model model12-00020-0.01745-0.99548-0.01745-0.99548.h5 as the final one.

A computer code with numbers

Description automatically generated with medium confidence

A graph of loss and loss

Description automatically generated

**Finally, we have achieved the test accuracy of 99.55% and validation accuracy of 96.00% using the transfer learning of MobileNet.**