**Gesture Recognition on Video contents:**

**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 recognize five different gestures performed by the user which will help users control the TV without using a remote.

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

Data consists of video clips with 30 frames each.

**Data:**

Data consists of sample video recorded with 5 different gestures. Each video consists of 30 frames. Sample data is provided in two sets:

1. Training
2. Validation

Each training and validation set consists of several sub folders. Each subfolder consists of frames of one video (30 frames for a gesture). Also, we have been provided with two csv files which consist of the video metadata (i.e. path of the video and its label).

All images/frames of a video are of same dimension, there are two variants of the video image dimensions: 360x360 and 120x160

**Solution Approach**

To build model which can detect gesture in the video, we have followed following steps:

1. Data Understanding
2. Preprocessing/Cleaning
3. Model Architectures
4. Model performance comparison
5. Final model selection

1. **Data Understanding**: We have input data given in zip format. Data consists of Training and validation sets. Each set consists of several sub folders, each sub folder having 30 frames.

Input video frame dimensions:

360x360x3

120x160x3

2. **Preprocessing**: Following two steps are performed in preprocessing:

1. Resize: In the preprocessing step we have converted two different size of frames into one dimension of size 120x120.
2. Normalization: Image data is normalized by dividing pixel values by 255.

Along with the data normalization and resizing, we need to implement generator function. which will feed data to the model for each EPOC. We need to implement generator function which can feed equal size batches of sequences and also take care of remaining sequences. For example, with training size of 19 sequences and batch size of 5 leads to 4 batches as below-

Batch-1: 5 sequences

Batch-2: 5 sequences

Batch-3: 5 sequences

Batch-4: 4 sequences

3. **Model Architecture**: There are two architectures which can be used to build model for gesture recognition.

1. **CNN + RNN:** This is the standard architecture for processing videos. In this architecture, video frames are passed through a CNN layer which extracts features from the images and then these feature vectors are fed to a RNN network to simulate sequence behavior of the video. Output of RNN is regular SoftMax function/
   1. We can use transfer learning in 2D CNN layer instead of training own network.
   2. LSTM or GRU can be used in RNN
2. **3D Convolution 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.

**Model Comparison:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Experiment Number** | **Model Architecture** | **Result** | **Decision + Explanation** |
| 1 | Input image = 120 x 160  Conv3D (**16 -> 32**)  hidden layers = 2  Optimizer: Adam | Training accuracy: 20%  Validation accuracy: 21% | Model is not able to learn. Needs to change architecture |
| 2 | Input image = 120 x 160  Conv3D (**16 ->32 -> 64**)  hidden layers = 3  padding = same  Optimizer: Adam | Training accuracy: 18%  Validation accuracy: 23% | Loss is stuck at 1.6 and is not reducing  Needs to update architecture |
| 3 | Input image = 120 x 160  Conv3D (**8 -> 16 ->32**)  hidden layers = 3  padding = same  Optimizer: Adam | Training accuracy: 98%  Validation accuracy: 70% | Looks overfitting. We need to optimize network for validation accuracy > 80% |
| 4 | Input image = 120 x 160  Conv3D (**16 ->32**)  hidden layers = 2  padding = same  Optimizer: Adam | Training accuracy: 92%  Validation accuracy: 61% | Network not achieving expected accuracy. Need to further optimize network |
| 5 | Input image = 120 x 160  Conv3D **(8 -> 16 ->32)**  **BatchNormalization**  hidden layers = 3  padding = same  Optimizer: Adam | Training accuracy: 97%  Validation accuracy: 71% | Loss is not reducing from 0.76  Network needs to be improved |
| 6 | Input image = 120 x 160  Conv3D **(8 -> 16 ->16)**  **BatchNormalization**  hidden layers = 3  padding = same  Optimizer: Adam | Training accuracy: 81%  Validation accuracy: 67% | Network needs to be improved |
| 7 | Input image = 120 x 160  Conv3D **(8 -> 16 ->32)**  **BatchNormalization**  hidden layers = 3  padding = same  Optimizer: Adam | Training accuracy: 74%  Validation accuracy: 33% | Validation accuracy is not improving beyond 34%  Architecture needs to be updated |
| 8 | Input image = 120 x 160  Conv3D **(8 -> 16 -> 16 -> 32)**  **BatchNormalization**  hidden layers = 4  padding = same  Optimizer: Adam | Training accuracy: 90%  Validation accuracy: 73% | Model looks better. We need to try this with increased number of epochs from 20 to 40 |
| 9 | Input image = 120 x 160  Conv3D **(8 -> 16 -> 16 -> 32)**  **BatchNormalization**  hidden layers = 4  padding = same  Optimizer: Adam  **Epochs = 40**  **Reduce learning rate factor = 0.5** | Training accuracy: 91%  Validation accuracy: 83% | We have achieved validation accuracy more than 80% with this architecture.  In next model we will experiment with different image size as input |
| 10 | **Input image = 120 x 120**  Conv3D (8 -> 16 -> 16 -> 32)  **BatchNormalization**  hidden layers = 4  padding = same  Optimizer: Adam  **Epochs = 40** | Training accuracy: 90%  Validation accuracy: 73% | No improvement observed in validation accuracy. |
| 11 | Same as Model 9, except with a larger factor of 0.7 that is used to reduce the learning rate in case of a plateau. | Training accuracy: 96%  Validation accuracy: 81% | No improvement |
| 12 | **Conv2D + GRU**  Input image = 120 x 160  Conv2D **(8 -> 16 -> 16 -> 32)**  **BatchNormalization**  hidden layers = 4  padding = same  Optimizer: Adam  **GRU**  **Epochs = 15** | Training accuracy: 78%  Validation accuracy:64% | We have tried Conv2D with GRU and LSTM.  we got better accuracy with GRU model. |
| 13 | **Transfer Learning + GRU**  **GRU**  **Epochs = 20** | Training Accuracy: 99%  Validation Accuracy:70% | We have tried transfer learning using VGG19.  Model was highly overfitting. Validation Accuracy stuck at 70% |
| 14 | **Transfer Learning + GRU**  **Dropout + GlobalAveragePooling**  **Epochs = 15** | Training Accuracy: 85%  Validation Accuracy: 73% | To overcome with problem of overfitting we have used dropouts and Global Average  resulted increase in validation accuracy to 73% and reduced the overfitting. |

**Final Model Selection:**

Model 9 is our best model. It has provided an accuracy of 83% on the validation set consistently over multiple epochs. Also, it has not overfit the training data (unlike the previous models) with a training accuracy between 87-90%.

**The final model (Model 9) can be downloaded from here:**

<https://drive.google.com/drive/folders/1YDOavo0fBxQXc77tNTevpvRxqtanJeOy?usp=sharing>