Objective:

To develop a cool feature in the smart-TV for Home electronics 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. Following are the possible gesture classes:

1. Thumbs up : Increase the volume (index - 4)
2. Thumbs down : Decrease the volume (index - 3)
3. Left swipe : 'Jump' backwards 10 seconds ( index - 0)
4. Right swipe : 'Jump' forward 10 seconds ( index - 1)
5. Stop : Pause the movie (index - 2)

**Input:** Training data includes short videos 30 frames (images) for each gestures and labeled data as mentioned above.

**Image specification:**

There are 2 types of images available in the training dataset

* (360X360) pixels in height, width
* (120X120) height, width

**Possible step**: Make the images homogenous in size. (Image resizing and cropping)

Resize and cropping to: 120X120 (image in standard format - 28X28, 32X32, 64X64, and 128X128)

**Approached to case study:**

* Two types of architectures are used commonly. One is the standard CNN + RNN architecture in which we can pass the images of a video through a CNN architecture which extracts a feature vector for each image, and then pass the sequence of these feature vectors through an RNN.

RNN use lots of parameters (due to their recurrent nature) as compared to CNNs,

So we have to avoid creating lots of unnecessary parameters.

* Other one is natural extension of CNNs - a 3D convolution network. In this case, the input to a 3D convolution is a video (which is a sequence of 30 RGB images) .Assuming 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.
* Using generators we create batches of images , here we are taking size of 10 initially, Others are

Frame = 20, rows = 120, cols = 120 , Channel\_color = 3, num\_classes = 5. For Image processing

We also apply Normalization on the image using Open-CV Library methods.

|  |  |  |  |
| --- | --- | --- | --- |
| **Experiment Number** | **Model** | **Result** | **Decision + Explanation** |
| 1  Created Model using vanilla CNN architecture with input layer(16Neurons) , 2 hidden layer (1st layer : 64 Neurons 2nd layer : 128 )  Fully connected and output (dense layer).  Filter used (3X3X3). | Model-1 (vanilla)  Conv3D | Total parameters: 147,737,733  Trainable parameters: 147,737,285  Accuracy: 47.01  Valid accuracy : 33.00 | Tried to tune hyper the network and apply layers and other possible ways to get model performance and train and validation accuracy. |
| 2.  Reducing filter size to (2X2X2) and optimizer to ‘adam’.  Using Batch Normalization after each layer. Added dropout after each hidden layer. | Model-2  Reducing Filter (2,2,2) & Added Hyper Tuning Parameters  (Conv3D) | Total parameters are : 3,507,141  Trainable parameters: 3,506,661  Accuracy: 91.19  Valid accuracy: 43.00 | Reduced the size of the image and filter size did not produce and good accuracy hence getting back to same filter. We have also tried to hyper tuning of parameters through dropout and Batch normalization. |
| 3.  Resuming filter size to 3X3X3 and optimizer to ‘SGD’ and applying L2 regularization. | Model -3  changing optimizer to "SGD" & Added L2 Regularization  (Conv3D) | Total parameters are : 3,507,141  Trainable parameters: 3,506,661  Accuracy: 69.40  Valid accuracy: 47.00 | Increased the amount of trainable data/ reduce the filter size. We assume that using more hyper tuning will increase the accuracy hence we are trying different combinations of neural layers and trying to reduce over fitting while increasing the accuracy decreasing loss. |
| 4. Adding more hidden layers to increase the accuracy | Model -4  Adding More Layers  (Conv3D) | Total parameters: 2,481,701  Trainable parameters: 2,480,773  Accuracy : 81.49  Valid accuracy : 51.99 | We Added more hidden layers but it didn’t improved accuracy any further. We just doubled each of the hidden layer and performed max pooling on the output to produce better result and using dropout we are managing the over fitting of training data. |
| 5. Adding one more dense layer with filter size (2,2,2) | Model -5 Adding one more dense layer with filter size (2,2,2) | Total params: 1,827,637  Trainable params: 1,827,157  Accuracy : 68.65  Valid accuracy : 30.00 | We Added one more hidden layers with reduced filter size of (2,2,2), thought it will increase the accuracy. However it didn’t improved accuracy any further. fitting of training data. The training accuracy is 52% which is very less. |
| 6.  Using 2nd method of (CNN+RNN) architecture | Model – 6  (CNN + SIMPLE RNN NETWORK) | Total parameters: 1,836,901  Trainable parameters: 1,836,421  Accuracy : 88.50  Valid accuracy : 63.99 | We used first approach of using a combination of (CNN) for image processing and feature extraction and passing it to RNN for sequential feature analysis and producing output in one of the 5 category of gestures on the basis of RNN output. |
| 7.  Added more neurons and dropout as hyper tuning in (CNN+RNN) for better accuracy | Model – 7  (CNN+RNN) | Total parameters: 3,870,469  Trainable parameters: 3,869,893  Accuracy : 85.82  Valid Accuracy: 56.99 | We observed that vanilla architecture for CNN+RNN requires more hyper tuning due to over fitting hence we tried to add dropout and increase neurons to gather as much as input from image processing. |
| 8.  Added 3 hidden layers with batch normalization. (CNN + LSTM ) | Model – 8  (CNN + LSTM) | Total parameters: 3,392,869  Trainable parameters: 3,392,389  Accuracy : 80.00  Valid Accuracy: 60.00 | We assume that LSTM would provide better results than RNN as LSTM can handle over fitting which can be seen in the previous results and it helps in better learning of model as it stores the relevant information plus the current state. |
| 9.  CNN + LSTM Network(Added more cells, Removed Hidden Layer, Added Dropout) | Model – 9  (CNN + LSTM) | Total parameters: 60,392,389  Trainable parameters: 60,391,941  Accuracy : 77.76  Valid Accuracy: 63.00 | We tried to manage the output using LSTM and reduce the layer and added dropout for managing over fitting. But this resulted in increase in the number of parameters. |
| 10.  CNN + GRU  Network for better speed in computation | Model – 10  (CNN +GRU) | Total parameters: 5,180,517  Trainable parameters: 5,180,037  Accuracy : 90.14  Valid Accuracy: 73.00 | We also experimented with CNN+GRU architecture in order to check the accuracy over traditional RNN and LSTM models as GRU provides better and fast computation as compared to others. But we observed major gap between accuracy and validation accuracy. |

|  |  |  |  |
| --- | --- | --- | --- |
| 11.  Model 10 : CNN + GRU Network(Added more cells, Added Hidden Layer, Added Dropout) | MODEL-11  CNN +GRU | Total parameters: 3,123,077  Trainable parameters: 3,122,181  Accuracy : 69.55  Valid Accuracy: 38.99 | We tried a few variations in previous model to increase validation accuracy and tried to increase parameters for better learning while controlling the output with dropout. We saw this is not achieved by changes we made in the model. |
| Final Model:  We concluded from the above all models we created for this case study Model- 3 with CONV-3D has shown the closest accuracy with train accuracy and validation accuracy. | Model – 10  (CNN +GRU) | Total parameters: 5,180,517  Trainable parameters: 5,180,037  Accuracy : 90.14  Valid Accuracy: 73.00 | We also experimented with CNN+GRU architecture in order to check the accuracy over traditional RNN and LSTM models as GRU provides better and fast computation as compared to others. But we observed major gap between accuracy and validation accuracy.  Attached file  model-00010-0.30169-0.90149-0.74814-0.73000.h5 |
|  |  |  |  |