**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

Each video is a sequence of 30 frames (or images)

**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.

Architectures used :

**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).

1. Base model without Transfer Learning
2. CNN-LSTM model with Transfer learning

**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.

Below are details of all the experiments performed using different models and hyperparameter tuning. A short summary is provided for each experiment to explain the rationale behind it.

**Note :**

**Please note that below two architectures has been finalised after lot of experimentation and as multiple models (40+ models were tried out) only 11 have been mentioned but have tried to cover all sorts of hyperparameter tuning with respect to architecture and explained relevant ones in this document.**

**CNN-RNN Stack**



**3D CNN stack**



|  |  |
| --- | --- |
| Experiment no | 01 (Base Model) |
| Batch size considered | 16 |
| Image size (in pixels) considered | 100\*100 |
| Frame count considered | 30 |
| Type of RNN Used | LSTM |
| No of epochs | 15 |
| Base model train\_accuracy | 18 |
| Base model val\_accuracy | 23 |

**Observation/findings** : Base model did not give good accuracy as extracting features from images would need a robust CNN model with multiple layers. We tried to see what is output of base model and we will utilise ResNet50 transfer learning in following experiments to see the improvements.

|  |  |
| --- | --- |
| Experiment no | 02 |
| Batch size considered | 16 |
| Image size considered | 100\*100 |
| Frame count considered | 30 |
| Image normalisation technique | Basic Feed normalisation |
| Type of RNN Used | LSTM |
| No of epochs | 15 |
| CNN-RNN stack train\_accuracy(Transfer Learning) | 20 |
| CNN-RNN stack val\_accuracy(Transfer Learning) | 21 |
| 3D CNN stack train\_accuracy | 79 |
| 3D CNN stack val\_accuracy | 22 |

**Observation/findings** : In this case we used Transfer Learning (ResNet50) in CNN-RNN stack. We used normal image normalisation (dividing each image RGB channel by 255). We also used all available 30 frames from video for training. It gave poor accuracy in CNN-RNN stack and overfitted with high train accuracy in 3D CNN stack.

|  |  |
| --- | --- |
| Experiment no | 03 |
| Batch size considered | 16 |
| Image size considered | 100\*100 |
| Frame count considered | 30 |
| Image normalisation technique | Basic Feed normalisation (Different type ) |
| Type of RNN Used | LSTM |
| No of epochs | 15 |
| CNN-RNN stack train\_accuracy(Transfer Learning) | 38 |
| CNN-RNN stack val\_accuracy(Transfer Learning) | 39 |
| 3D CNN stack train\_accuracy | 88 |
| 3D CNN stack val\_accuracy | 49 |

**Observation/findings** : In this case we used Transfer Learning (ResNet50) in CNN-RNN stack. We used normal image normalisation (dividing each image RGB channel by 100 (as pixels are resized to 100\*100). We also used all available 30 frames from video for training.It gave mediocre accuracy in CNN-RNN stack and overfitted with high train accuracy in 3D CNN stack.

|  |  |
| --- | --- |
| Experiment no | 04 |
| Batch size considered | 16 |
| Image size considered | 100\*100 |
| Frame count considered | 30 |
| Image normalisation technique | Optimised |
| Type of RNN Used | LSTM |
| No of epochs | 20 |
| CNN-RNN stack train\_accuracy(Transfer Learning) | 61 |
| CNN-RNN stack val\_accuracy(Transfer Learning) | 57 |
| 3D CNN stack train\_accuracy | 93 |
| 3D CNN stack val\_accuracy | 84 |

**Observation/findings** : In this case we used Transfer Learning (ResNet50) in CNN-RNN stack. We used optimised image normalisation (by some research and post online relevant reading articles). We also used all available 30 frames from video for training.It gave decent accuracy in CNN-RNN stack and very good accuracy in 3D CNN stack.

|  |  |
| --- | --- |
| Experiment no | 05 |
| Batch size considered | 16 |
| Image size considered | 100\*100 |
| Frame count considered | 22 (First 4 and last 4 frames excluded) |
| Image normalisation technique | Optimised |
| Type of RNN Used | LSTM |
| No of epochs | 15 |
| CNN-RNN stack train\_accuracy(Transfer Learning) | 55 |
| CNN-RNN stack val\_accuracy(Transfer Learning) | 60 |
| 3D CNN stack train\_accuracy | 80 |
| 3D CNN stack val\_accuracy | 57 |

**Observation/findings** : In this case we used Transfer Learning (ResNet50) in CNN-RNN stack. We used optimised image normalisation. We used total 22 frames from video for training (reason – Observed that first 4 images and last 4 images did not have significant hand gestures and most of gestures are captured in the middle 22 frames). It gave decent accuracy in CNN-RNN stack and good accuracy in 3D CNN stack but overfitted.

|  |  |
| --- | --- |
| Experiment no | 06 |
| Batch size considered | 64 |
| Image size considered | 84\*84 |
| Frame count considered | 18 (alternate frames) |
| Image normalisation technique | Optimised |
| Type of RNN Used | LSTM |
| No of epochs | 15 |
| CNN-RNN stack train\_accuracy(Transfer Learning) | 80 |
| CNN-RNN stack val\_accuracy(Transfer Learning) | 81 |
| 3D CNN stack train\_accuracy | 63 |
| 3D CNN stack val\_accuracy | 52 |

**Observation/findings** : In this case we used Transfer Learning (ResNet50) in CNN-RNN stack. We used optimised image normalisation and tried to experiment with batch size of 64 to see if changing batch size brings any improvements. We used total 18 frames from video for training (reason – Observed that consecutive frames in middle 24 frames were bit redundant and two consecutive frames did not have lot of information difference, hence we chose 1 out of 2 consecutive frames). We also tried to play around with image size and kept it as 84\*84. It gave very good accuracy in CNN-RNN stack and average accuracy in 3D CNN stack with bit of overfitting.

|  |  |
| --- | --- |
| Experiment no | 07 |
| Batch size considered | 16 |
| Image size considered | 84\*84 |
| Frame count considered | 18 (alternate frames) |
| Image normalisation technique | Optimised |
| Type of RNN Used | LSTM |
| No of epochs | 15 |
| CNN-RNN stack train\_accuracy(Transfer Learning) | 72 |
| CNN-RNN stack val\_accuracy(Transfer Learning) | 75 |
| 3D CNN stack train\_accuracy | 81 |
| 3D CNN stack val\_accuracy | 73 |

**Observation/findings** : In this case we used Transfer Learning (ResNet50) in CNN-RNN stack. We used optimised image normalisation. We used total 18 frames from video for training (reason – Observed that consecutive frames in middle 24 frames were bit redundant and two consecutive frames did not have lot of information difference, hence we chose 1 out of 2 consecutive frames). We also tried to play around with image size and kept it as 84\*84. It gave very good accuracy in CNN-RNN stack and average accuracy in 3D CNN stack with bit of overfitting.

|  |  |
| --- | --- |
| Experiment no | 08 |
| Batch size considered | 16 |
| Image size considered | 64\*64 |
| Frame count considered | 18 (alternate frames) |
| Image normalisation technique | Optimised |
| Type of RNN Used | LSTM |
| No of epochs | 20 |
| CNN-RNN stack train\_accuracy(Transfer Learning) | 60 |
| CNN-RNN stack val\_accuracy(Transfer Learning) | 53 |
| 3D CNN stack train\_accuracy | 87 |
| 3D CNN stack val\_accuracy | 73 |

**Observation/findings** : In this case we used Transfer Learning (ResNet50) in CNN-RNN stack. We used optimised image normalisation. We used total 18 frames from video for training (reason – Observed that consecutive frames in middle 24 frames were bit redundant and two consecutive frames did not have lot of information difference, hence we chose 1 out of 2 consecutive frames). We also tried to play around with image size and kept it as 64\*64. It gave good accuracy in CNN-RNN stack and high accuracy in 3D CNN stack with bit of overfitting.

|  |  |
| --- | --- |
| Experiment no | 09 |
| Batch size considered | 16 |
| Image size considered | 100\*100 |
| Frame count considered | 18 (alternate frames) |
| Image normalisation technique | Optimised |
| Type of RNN Used | LSTM |
| No of epochs | 15 |
| CNN-RNN stack train\_accuracy(Transfer Learning) | 71 |
| CNN-RNN stack val\_accuracy(Transfer Learning) | 59 |
| 3D CNN stack train\_accuracy | 76 |
| 3D CNN stack val\_accuracy | 55 |

**Observation/findings** : In this case we used Transfer Learning (ResNet50) in CNN-RNN stack. We used optimised image normalisation. We used total 18 frames from video for training (reason – Observed that consecutive frames in middle 24 frames were bit redundant and two consecutive frames did not have lot of information difference, hence we chose 1 out of 2 consecutive frames). We also tried to play around with image size and kept it as 100\*100. It gave good accuracy in CNN-RNN stack (using LSTM) and good accuracy in 3D CNN stack as well but with overfitting.

|  |  |
| --- | --- |
| Experiment no | 10 |
| Batch size considered | 16 |
| Image size considered | 100\*100 |
| Frame count considered | 30 |
| Image normalisation technique | Optimised |
| Type of RNN Used | GRU |
| No of epochs | 20 |
| CNN-RNN stack train\_accuracy(Transfer Learning) | 98 |
| CNN-RNN stack val\_accuracy(Transfer Learning) | 93 |
| 3D CNN stack train\_accuracy | 88 |
| 3D CNN stack val\_accuracy | 60 |
| Filter size used in 3D CNN | 3\*3\*3 |

**Observation/findings** : In this case we used Transfer Learning (ResNet50) in CNN-RNN stack. We used optimised image normalisation. We used total 18 frames from video for training (reason – Observed that consecutive frames in middle 24 frames were bit redundant and two consecutive frames did not have lot of information difference, hence we chose 1 out of 2 consecutive frames). We also tried to play around with image size and kept it as 100\*100. It gave very high accuracy in CNN-RNN stack (using GRU) and decent accuracy in 3D CNN stack as well but with overfitting. Intention was to use both LSTM and GRU to see how each architecture behaves in this problem statement.

|  |  |
| --- | --- |
| Experiment no | 11 |
| Batch size considered | 16 |
| Image size considered | 100\*100 |
| Frame count considered | 30 |
| Image normalisation technique | Optimised |
| Type of RNN Used | GRU |
| No of epochs | 20 |
| CNN-RNN stack train\_accuracy(Transfer Learning) | 98 |
| CNN-RNN stack val\_accuracy(Transfer Learning) | 93 |
| 3D CNN stack train\_accuracy | 92 |
| 3D CNN stack val\_accuracy | 89 |
| Filter size used in 3D CNN | 4\*4\*4 |

**Observation/findings** :

Model in experiment 11 is similar to model in experiment 10 with only difference being 4\*4\*4 filter being used in 3D CNN. However it does seem to have improved validation accuracy of 3D CNN and removed overfitting.

**Summary/final model gist :**

1. Thus, after multiple round of experiments and hyperparameter tuning, we have observed that CNN-RNN stack with GRU has given the highest amount of accuracy on both train and validation set (even 3D CNN stack has given good results but we decide to go with CNN-RNN stack).
2. Additionally, the number of trainable params is also very less for GRU model compared to LSTM model which is a great plus in this situation (As LSTM model has 4 gates and GRU model uses 3 gates in its architecture/structure).

**Params in defined LSTM architecture**

lstm (LSTM) (None, 30, 4028) 97912624

lstm\_1 (LSTM) (None, 30, 2024) 49005088

lstm\_2 (LSTM) (None, 1012) 12293776

**Params in same defined GRU architecture**

gru (GRU) (None, 30, 4028) 73446552

gru\_1 (GRU) (None, 30, 2024) 36759888

gru\_2 (GRU) (None, 1012) 9223368