**Icon

Description automatically generated**

Contents

[Problem Statement 3](#_Toc88288710)

[Understanding the Dataset 3](#_Toc88288711)

[Objective 4](#_Toc88288712)

[Approach to solve the problem 5](#_Toc88288713)

[Articulation of Experiments 6](#_Toc88288714)

# Problem Statement

As a data scientist at a home electronics company which manufactures state of the art **smart televisions**. We 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:

* 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.

# Understanding the Dataset

The training data consists of a few hundred videos categorized 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.

Table

Description automatically generated

The zipped folder given to us contains a 'train' and a 'val' folder with two CSV files for the two folders. These folders are in turn divided into subfolders where each subfolder represents a video of a particular gesture. Each subfolder, i.e. a video, contains 30 frames (or images). Note that all images in a particular video subfolder have the same dimensions but different videos may have different dimensions. Specifically, videos have two types of dimensions - either 360x360 or 120x160 (depending on the webcam used to record the videos). Hence, you will need to do some pre-processing to standardise the videos.

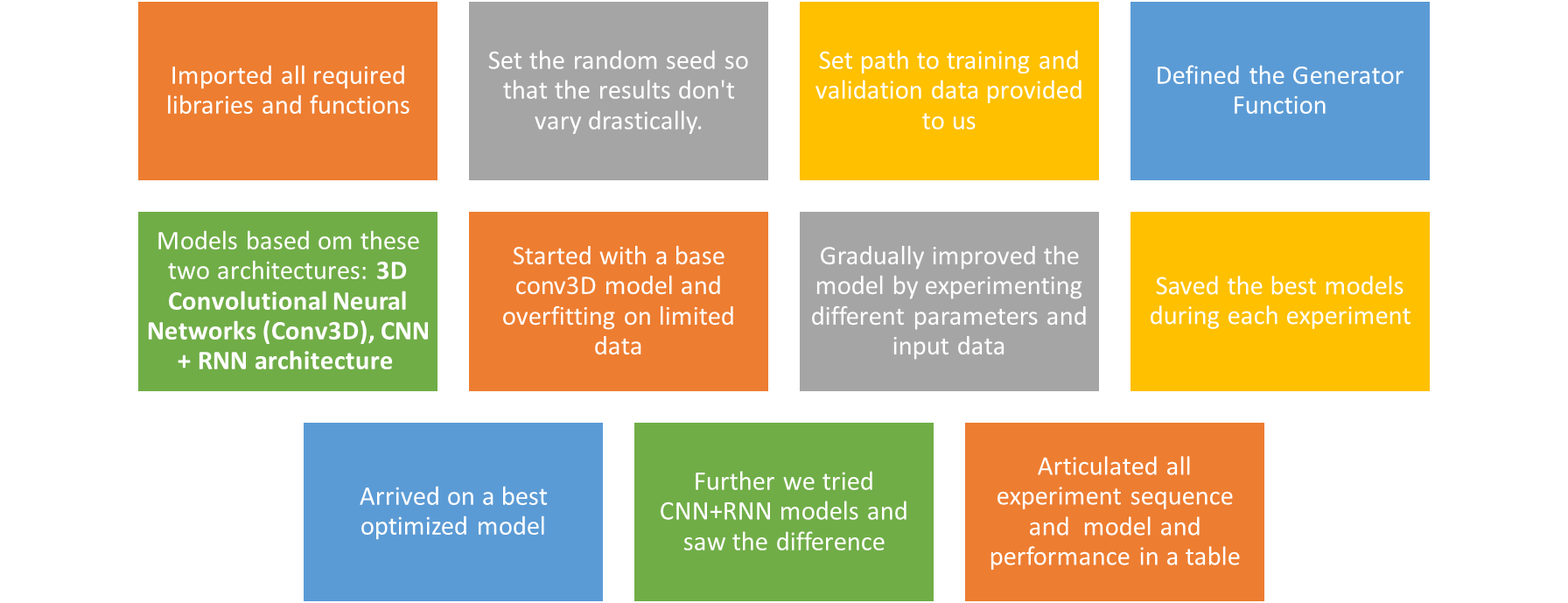
Each row of the CSV file represents one video and contains three main pieces of information - the name of the subfolder containing the 30 images of the video, the name of the gesture and the numeric label (between 0-4) of the video.

# 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.

# Approach to solve the problem

We have gone step by step with given sequence to find an optimal solution for this problem:



* Generator Function:
  + This is probably the most important part of building a training pipeline. Keras and other libraries provide built in generator functionalities, but they are restricted in scope and here we write own generator. In this problem we need to create a batch of videos
  + Passed img\_idx, image height and width as parameter to generatoe function to experiment various combinations for these variables
  + Generator function will provide us batch wise data in each call based on parameters (data path, batch size, img\_idx, imae height and width) passed to it
  + It runs in an infinite while loop
  + We are cropping the images to give a standard size
  + Since images are in different sizes, we are resizing them to make them homogeneous for input to our model
  + Normalizing the images for faster convergance
  + In case number of batches are not integer we are writing a section of code to handle the left out videos and provide one more round of data for our model
* Models based on these two architectures:
  + **3D Convolutional Neural Networks (Conv3D)**
  + 3D convolutions are a natural extension to the 2D convolutions. Just like in 2D conv, we 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).
  + **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).
    - Base Model and Gradually improved the model by experimenting different parameters and input data

# Articulation of Experiments

Refer below table for all experiments in sequence, results we got with each model, number of parameters and our thought process along with explanation around that.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Experiment No.** | **Model** | **Model variables** | **Result** | **Decision + Explanation** | **Number of Parameters** |
| 1 | Conv3D | Batch Size: 128 No. of images per video: 15  Image Size: 160x160 filter size = (3,3,3) Epochs: 15 | Kernel died/ resources exhausted | Started with a Conv3D base model having:  4 Conv3D layers,  Flatten, Dense and  output with softmax.   Idea was to have a basic model to start with and overfit on limited data where we chose to have 15 images per video.   We also used Max pooling and batch normalization in base model itself.  Kernel died as we chose a very high batch size. |  |
| 2 | Conv3D | Batch Size: 64 No. of images per video: 15  Image Size: 160x160 filter size = (3,3,3) Epochs: 15 | Epoch 15: Training Accuracy--100% Validatio Accuracy--40% | We reduced Batch size to 64 and ran the model and observed the model clearly overfitting with the given results after running 15 epochs | Total params: 1,932,485 Trainable params: 1,931,749 Non-trainable params: 736 |
| 3 | Conv3D | Batch Size: 64 No. of images per video: 15  Image Size: 160x160 filter size = (3,3,3) Epochs: 15 Dropouts-25% after each Conv3D and 50% at Dense Layer | Epoch 15: Training Accuracy--97.6% Validation Accuracy--31% | We added dropouts in this layer to control overfitting. Dropouts didn't help much to reduce the overfitting | Total params: 1,932,485 Trainable params: 1,931,749 Non-trainable params: 736 |
| 4 | Conv3D | Batch Size: 64 No. of images per video: 22  Image Size: 160x160 filter size = (3,3,3) Epochs: 15 Dropouts-25% after each Conv3D and 50% at Dense Layer | Kernel died/ resources exhausted | We tried to use more image per video to improve results, but Kernel died |  |
| 5 | Conv3D | Batch Size: 40 No. of images per video: 22  Image Size: 160x160 filter size = (3,3,3) Epochs: 15 Dropouts-25% after each Conv3D and 50% at Dense Layer | Epoch 15: Training Accuracy--99.06% Validation Accuracy--44% | We further reduced the batch size to 40 to adjust with increased images per video and compute resources we had.  Though we saw improvement in validation accuracy in this model but still we have overfit model | Total params: 3,570,885 Trainable params: 3,570,149 Non-trainable params: 736 |
| 6 | Conv3D | Batch Size: 40 No. of images per video: 22  Image Size: 160x160 filter size = (3,3,3) Epochs: 15 Dense nodes reduced from 128 to 64 Dropouts-25% after each Conv3D and 50% at Dense Layer | Epoch 11: Training Accuracy--95.87% Validation Accuracy--40% | To further improve the model we simplified dence layer and reduced nodes from 128 to 64.  This approach didn't help much | Total params: 1,931,845 Trainable params: 1,931,237 Non-trainable params: 608 |
| 7 | Conv3D | Batch Size: 20 No. of images per video: 22  Image Size: 160x160 filter size = (3,3,3) Epochs: 25 Two Dense layers with 64 nodes each Dropouts-25% after each Conv3D and 50% at Dense Layers LR reduced from 0.001 to 0.0002 | Epoch 25: Training Accuracy--65.75% Validation Accuracy--48% | Since reducing node in dense layer didn't help, we thought of adding one more dense layer. Along with this we further reduced batch size to 20 and LR to 0.0002  We are not able to improve validation accuracy | Total params: 1,936,261 Trainable params: 1,935,525 Non-trainable params: 736 |
| **8** | **Conv3D** | **Batch Size: 20 No. of images per video: 22  Image Size: 120x120 filter size = (2,2,2) Epochs: 40 Two Dense layers with 128 nodes each Dropouts-10% after each Conv3D and 50% at Dense Layers LR 0.0002** | **Epoch 29: Training Accuracy--80.12% Validation Accuracy--74%** | To further improve model we did multiple experiments in between but with these variables where we reduced filter size, increased nodes on dense layers and reduced image size, reduced dropouts to 10% after Conv3D layers and increased to 50% after Dense Layers.  **We got a pretty good model which was not overfitting/ generalizing well on validation data.** | **Total params: 908,725 Trainable params: 907,733 Non-trainable params: 992** |
| 9 | Conv3D | Batch Size: 20 No. of images per video: 22  Image Size: 100x100 filter size = (2,2,2) Epochs: 40 Two Dense layers with 128 nodes each Dropouts-10% after each Conv3D and 50% at Dense Layers LR 0.0002 | Epoch 31: Training Accuracy--75.73% Validation Accuracy--67% | We further tried to reduce the parameters by reducing image size to 100x100 but didn't get good results.  We stopped here as we had pretty good results with previous model and optimized parameters | Total params: 695,733 Trainable params: 694,741 Non-trainable params: 992 |
| 10 | CNN+RNN(Transfer Learning with Mobile Net) | Batch Size: 15 No. of images per video: 16  Image Size: 224x224 Base Model =Mobilenet Epochs: 25  Dropouts-25% after GRU layer  LR 0.0002 | Epoch 25: Training Accuracy--90.31% Validation Accuracy--92% | We tried with MobileNet as base model for CNN due to its high-speed performance with lightweight architecture.  Because of input size parameters are very high.  We used GRU instead of LSTM because GRU is less complex.  We got very good results, but number of parameters are very high in this model | Total params: 5,638,261  Trainable params: 5,616,373  Non-trainable params: 21,888 |
| **11** | **CNN+RNN (without Transfer Learning)** | **Batch Size: 20 No. of images per video: 22  Image Size: 120x120 filter size = (2,2) Epochs: 25  Dropouts-10% after each Conv2D and 25% after GRU & Dense Layers LR 0.001** | **Epoch 18: Training Accuracy--86.57% Validation Accuracy--81%** | **Based on the previous experiment experience we now tried CNN+RNN. Here again we chose GRU for the same reason. We made CNN layers based on previous experience and tried to control the parameters.**  **Though we tried quite a few variables here also but kept this model in notebook giving pretty good results.** | **Total params: 1,270,261**  **Trainable params: 1,269,781**  **Non-trainable params: 480** |

After doing all the experiments, we finalized **Mode l 1 (Epoch 18)– CNN+RNN**, which performed well with optimized parameters and pretty good performance.

* (Training Accuracy: 86.57%, Validation Accuracy: 81%)
* Number of Trainable params: 1,269,781