Table of Contents

[Introduction 1](#_Toc78737371)

[Generator 2](#_Toc78737372)

[Image Processing 2](#_Toc78737373)

[Augmentation 2](#_Toc78737374)

[Models 3](#_Toc78737375)

[CNN3D 3](#_Toc78737376)

[CNN2D RNN 6](#_Toc78737377)

[Conclusion 7](#_Toc78737378)

# Introduction

**Problem Statement**

Imagine we 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:

| **Gesture** | **Corresponding Action** |
| --- | --- |
| 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. |

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

# Generator

In generator method, the images to be fed into the network is generated based on the batch size chosen. We have 663 unique folders in training data and 100 unique folders in validation data.

* The chosen batch size decides the number of batches of data. In each batch of data, we pick sample images (specified by the frames\_to\_sample) from “batch\_size” number of folders.

Eg: If the batch size is 10 and frame to sample is 20 each batch feeds 20x10 = 200 images into the network.

* We need to handle the case when the number of batches is not a multiple of the batch size chosen.

Eg: If the batch size is 10, then the number of batches is 663/10 = 66 and the remaining batch will have 3 folders (663 % 10 = 3)

* The smaller the batch\_size, the more the number of batches and hence more the number of feed forwards in the network.

## Image Processing

We do the following as part of the image processing in the generator

1. Cropping

Since the dataset is available in two shapes 360x360 and 120x160, we chose to crop 20 pixels from both the left and right side of the images which have the width 160 to make it a 120x120 image.

1. Image Resize

As mentioned, since we have images of dimensions 360\*360 and 120\*160, we have choice of resizing to at-least 120\*120 to make it uniform for Conv3D model to work. However, we tried to reduce to check if we get the right or better accuracy as we see that gesture is in the middle of the image.

1. Image Normalization

All the images are normalized using by dividing by 255 as they are all natural images.

## Augmentation

In the data generator, we had the following augmentation strategy implemented using OpenCV.

When augment flag is enabled, we generate two additional type of augmented dataset **for training data only**

1. We use a transformation matrix to do random location change in the range of [-30,30] for both X and Y axis on the original image. Then the image is converted to grayscale to help crop out any black borders using np.argwhere and finally the image is resized.
2. We rotate the original image by an of [-10,10] and resize the image.

This implies that with augmentation we end up having 3 times more images for training.

Eg. For a batch\_size of 10 and frames to sample as 20 without augmentation we will have

10x20 = 200 images.

But with augmentation, we will generate another 200 transformed images (from step 1 above) and another 200 rotated images (from step 2) above resulting in 600 images being fed into the network in every batch.

# Models

## CNN3D

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Experiment**  **Number** | **Model** | **Parameters** | **Result** | **Decision + Explanation** |
| 1 | Conv3D | 1736389 | Validation Accuracy: 0.16  Val Loss: 2.09  Training Accuracy: 0.98 | * The baseline model is established. Clearly the model is overfitting.   We will add more neurons to Dense layer and increase the dropout to 0.5 |
| 2 | Conv3D | 3434693 | Validation loss: 2.28 Validation Accuracy: 0.25  Training Accuracy: 0.96 | * The model is still not learning in spite of dropouts in fully connected layers * Accuracy has improved slightly possibly due to dense neurons   We will increase the number of samples to 30 |
| 3 | Conv3D | 3,434,693 | Validation loss: 2.94 Validation Accuracy: 0.19  Training Accuracy: 0.88 | * Adding more data didn’t improve the learning rate   We will reset the neurons at the dense layer to initial numbers.  We will now try to improve the learning by adding more layers. |
| 4 | Conv3D | 2,538,229 | Validation loss: 1.99 Validation Accuracy: 0.16  Training Accuracy: 0.58 | * Looks like the model is not learning even after adding multiple layers. So increasing epochs may not help.   We need to change the approach |
| 5 | Conv3D | 911,973 | Validation loss: 2.06 Validation Accuracy: 0.25  Training Accuracy: 0.78 | * There is slight improvement, and in the lower epochs looks like we are starting to learn * Complexity Reduction - We are able to reduce the parameters to less than 1M, with comparable accuracy(of the previous runs).   Increase the number of backpropagation and feedforward cycles.  We will try to decrease the batch size so there are more iterations of the model |
| 6 | Conv3D | 518,757 | Validation loss: 0.41 Validation Accuracy: 0.85  Training Accuracy: 0.85 | * Adding more batches results in more feedback loop and the model has optimum fit * With 10 batch size, we have around 66 Batches so 66 Feedforward loops which help in learning * Compare this with earlier batch size of 20, which creates 33 Feedforward loops only * By reducing the image size, we have also brought the no of parameters to 0.5 Million   We will start with Coarser filters initially and then extend to finer filters. |
| 7 | Conv3D | 579,877 | Validation loss: 0.24 Validation Accuracy: 0.95  Training Accuracy: 0.93 | * The results are even better * With little more coarse grained features, we are able to improve significantly by 10 percentage points * Also the Model is learning well and able to absorb most of the variations in the test data * Higher Epochs helped absorb more of the learning |
| 8 | Conv3D | 579877 | Validation loss: 0.18 Validation Accuracy: 0.94  Training Accuracy: 0.92 | Same model as 7 but with augmented data fed in. |

## CNN2D RNN

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Experiment**  **Number** | **Model** | **Parameters** | **Result** | **Decision + Explanation** |
| 1 | CNNRNN | 1,820,005 | Validation Accuracy: 0.92  Val Loss: 0.28  Training Accuracy: 0.96 | * CNNRNN Model learns quite well, and shows steady progress as the no of epochs increase * Historical State makes it easier to learn as compared to a plain CNN 3D model * The time taken to train an epoch is still high(Close to 90 Seconds which is 3 Times the 3D CNN Model), which we should try to reduce further * Augmented Data is causing significant increase in training time (Close to 30 minutes)     We will next create with GRU to reduce time |
| 2 | CNNRNN | 1,394,277 | Validation loss: 1.72 Validation Accuracy: 0.21  Training Accuracy: 0.42 | * The GRU model failed to learn * This means we have to tweak the hyperparameters or architecture * The time per epoch is now reduced to 79 seconds, which is expected.   We will increase the number of samples to 30  Rather than train a GRU from scratch, we will also try a new iteration with Transfer Learning. |
| 3 | CNNRNN | 3,840,453 | Validation loss: 0.248 Validation Accuracy: 0.91  Training Accuracy: 0.98 | * Transfer Learning has worked * In just 20 Epochs we have good results compared to the best model of CNN which took 50 Epochs. * By just retraining for a few more epochs we can have the best model * Time Taken is only 26 minutes, and model is also learning adequately   Next we will train the mobile net params too |
| 4 | CNNRNN | 3,693,253 | Validation loss: 0.028 Validation Accuracy: 0.99  Training Accuracy: 0.98 | * Using a pretrained architecture, we are able to save on multiple iterations for model hyperparameter tuning * Time Taken for Training is 26 min (3 M Parameters), which is very impressive, and due to GRU |

# Conclusion

Based on the above Readings, Model 8 is the best model for CNN3D, and Model 4 is the best model for CNN RNN