**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:

* 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

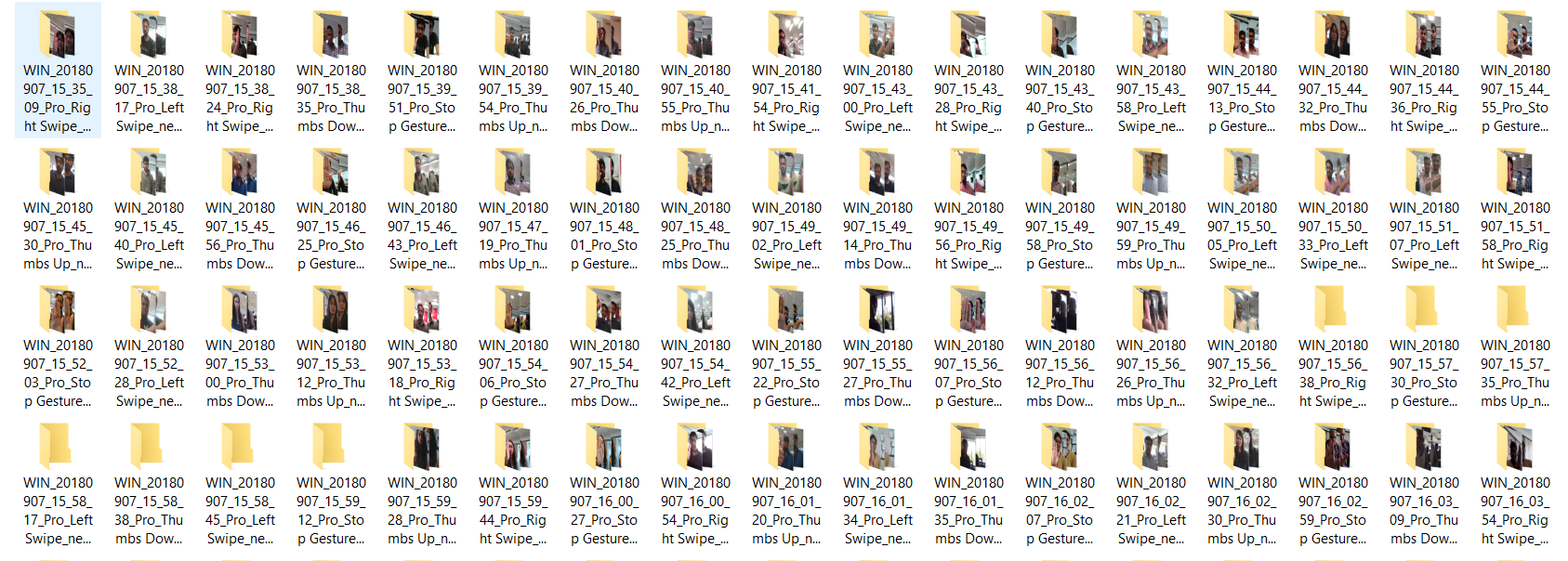
Each video is a sequence of 30 frames (or images)

## ****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.

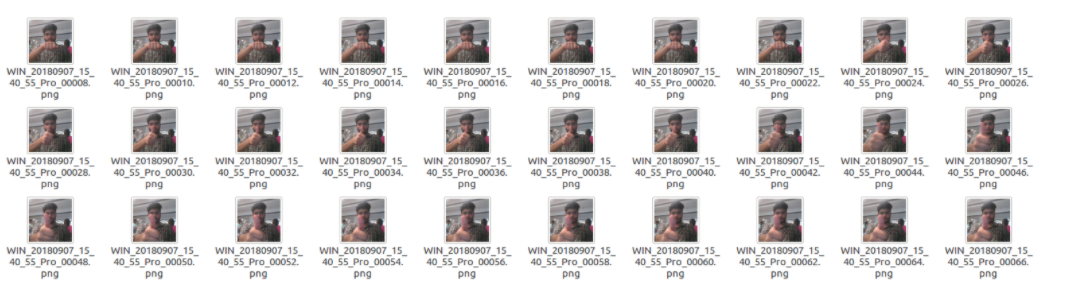
The data is in a zip file. The zip file contains a 'train' and a 'val' folder with two CSV files for the two folders.



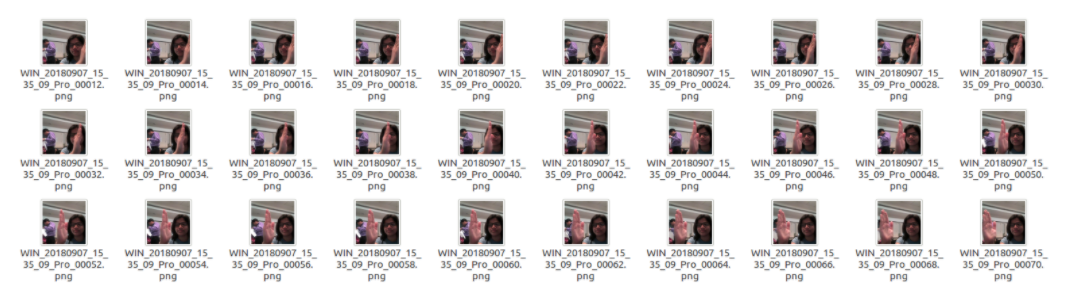
These folders are in turn divided into subfolders where each subfolder represents a video of a particular gesture.  


Each subfolder, i.e. a video, contains 30 frames (or images).

* Thumbs Up



* Right Swipe



**Two Architectures: 3D Convs and CNN-RNN Stack**

After understanding and acquiring the dataset, the next step is to try out different architectures to solve this problem.

For analysing videos using neural networks, two types of architectures are used commonly.

One is the standard CNN + RNN architecture in which you pass the images of a video through a CNN which extracts a feature vector for each image, and then pass the sequence of these feature vectors through an RNN.

Note:

* You can use transfer learning in the 2D CNN layer rather than training your own CNN
* GRU (Gated Recurrent Unit) or LSTM (Long Short Term Memory) can be used for the RNN

The other popular architecture used to process videos is a natural extension of CNNs - a 3D convolutional network. In this project, we will try both these architectures.

**Implementation steps:**

## Data Import: We are using google colab to build and train the model. We are downloading and extracting the zip folder in the /content folder of colab runtime.

## Generator: This is one of the most important part of the code. The overall structure of the generator has been given. In the generator, you are going to preprocess the images as you have images of 2 different dimensions as well as create a batch of video frames. You have to experiment with img\_idx, y,z and normalization such that you get high accuracy.

## Sample Model: Here you make the model using different functionalities that Keras provides. Remember to use Conv3D and MaxPooling3D and not Conv2D and Maxpooling2D for a 3D convolution model. You would want to use TimeDistributed while building a Conv2D + RNN model. Also remember that the last layer is the softmax. Design the network in such a way that the model is able to give good accuracy on the least number of parameters so that it can fit in the memory of the webcam.

#### **Experiment** - to see how training time is affected by image resolution, number of images in sequence and batch size

### **Observation**:

### The experiment shows that image resolution and number of frames in sequence have more impact on training time than batch size**.** So experimentations are carried with batch size fixed around 15-40 and changing the resolution and number of image per sequence based on the device memory constraints. Models are designed such that their memory foot print is less than 50 MB which corresponds to 4.3 million parameters assuming the datatype size of parameters to be 12 bytes

## Implementing various models

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| High level view | Model Name | Description | Total Parameters: | Validation Accuracy: | Training Accuracy | Conclusion |
| Basic model | Conv3D1\_model | Base Model - No Data Augmentation Batch Size40 and Epoch 15 | 1,117,061 | 19% | **97.29%** | Model is clearly overfitting. So we need to do data augmentation |
| Conv3D2\_model | Augment Data , (3,3,3) filter & 160x160 image resolution | 3,638,981 | 81% | **75%** | Model is not overfitting and we get a best validation accuracy of ~81% and training accuracy of ~75%. Next we will try to reduce the filter size and image resolution and see if get better results. Moreover, since we see minor oscillations in loss, let's try lowering the learning rate to 0.0002 |
| Conv3D3\_model | Reduce filter size to (2,2,2) and image res to 120 x 120 | 1,762,613 | 65% | **70.44%** | This Model has a best validation accuracy of 65% and training accuracy of 70% . Also we were able to reduce the parameter size by half the earlier model. However, the accuracy drops from the earlier model. |
| Conv3D4\_model | Adding more layers | 2,556,533 | 78% | 87% | With more layers we see some **performance improvement**. We get a best validation accuracy of 78% . |
| Conv3D5\_model | Adding dropout at convolution layers | 2,556,533 | 34% | 83.56% | Adding dropouts has further **reduced validation accuracy as its not to learn generalizable features** All models experimental models above have more than 1 million parameters. |
| Conv3D6\_model | reducing the number of parameters | 696,645 | 84% | 80.17% | For this low memory foot print model, the **best validation accuracy of 84%** |
| Conv3D7\_model | reducing the number of parameters | 504,709 | 74% | 76.32% | For this low memory foot print model, the **best validation accuracy of 74%** |
| Conv3D8\_model | reducing the number of parameters | 230,949 | 61% | 63.12% | For this low memory foot print model**, the best validation accuracy of 61%** |
| cnn\_rnn1\_model | CNN- LSTM Model | 1,657,445 | 82% | 89.59% | For CNN - LSTM model we get a **best validation accuracy of 82%** |
| Applying More Augmentation | conv\_3d10\_model | (3,3,3) Filter & 160x160 Image resolution - similar to Model 2 | 3,638,981 | 86% | 74.12% | The model **performs better** on the validation set with **higher accuracy of 86%** and the model is more generalizable. |
| conv\_3d11\_model | (2,2,2) Filter & 120x120 Image resolution - similar to Model 3 | 1,762,613 | 70% | 67.12% | The validation **accuracy drops** to 70%. However the model is **generalizable does not overfit.** |
| conv\_3d12\_model | Adding more layers - Similar to model 4 | 2,556,533 | 77% | 71% | The model generated with validation accuracy of 77% after adding more augmentation |
| conv\_3d13\_model | Adding dropouts - Similar to Model 5 | 2,556,533 | 65.91% | 28% | The model still **overfits**. Even after the augmentation. |
| conv\_3d14\_model | reducing network parameters - Similar to Model 6 | 696,645 | 73% | 74% | **Best validation accuracy with lower footprint 73%** |
| conv\_3d15\_model | reducing network parameters - Similar to model 7 | 504,709 | 72% | 73.6% | Best validation accuracy is 72% with **no overfitting.** |
| conv\_3d16\_model | reducing network parameters - Similar to Model 8 | 230,949 | 72% | 62% | The model performs better on the validation set. However accuracy is 72%. |
| rnn\_cnn2\_model | CNN LSTM with GRU - Similar to Model 9 | 2,573,925 | 68% | 93% | We can see the model is **overfitting** even after augmentation and **accuracy drops**. |
| Additional Models - Transfer Learning | rnn\_cnn\_tl\_model |  | 3,840,453 | 89% | 55% | We are not training the mobilenet weights and we see **validation** **accuracy is very poor** |
| Transfer Learning with GRU and training all weights | rnn\_cnn\_t2\_model |  | 3,840,453 | 99.47% | 94% | We get a **better accuracy** on training mobilenet layer’s weights as well. |

## Summary:

## Based on the stats mentioned above we would go with Augmented model conv\_3d10\_model, because it has high validation accuracy and it is more generalisable

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| High level view | Model Name | Description | Total Parameters: | Validation Accuracy: | Training Accuracy |
| Applying More Augmentation | conv\_3d10\_model | (3,3,3) Filter & 160x160 Image resolution - similar to Model 2 | 3,638,981 | 86% | 74.12% |

## Transfer learning: Transfer learning generally refers to a process where a model trained on one problem is used in some way on a second related problem

## 1.Transfer learning has the benefit of decreasing the training time for a neural network model and can result in lower generalization error.

## 2.Transfer learning without training the mobilenet weights would result in less accuracy. So this is improved by Transfer Learning with GRU and training all weights

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| High level view | Model Name | Total Parameters: | Validation Accuracy: | Training Accuracy |
| Transfer Learning with GRU and training all weights | rnn\_cnn\_t2\_model | 3,840,453 | 99.47% | 94% |

## Low memory footprint: The model performs better on the validation set with higher accuracy of 86% and the model is more generalizable.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Model Name | Total Parameters: | | Validation Accuracy: | | Training Accuracy | |
| Conv3D6\_model | | 696,645 | | 84% | | 80.17% | |

## Our final model is conv\_3d10\_model and the model file is model-00029-0.74428-0.72775-0.72484-0.78000.h5