# Neural Networks Project – Gesture Recognition

1. **Sanjay Belgaonkar – Group Facilitator**
2. **Yashraj Pathak**

## 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. |

## 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.

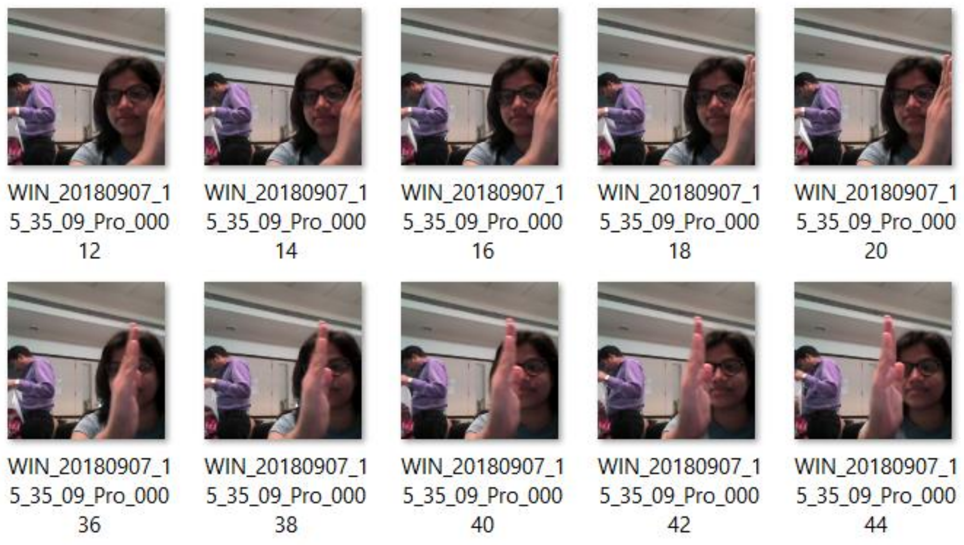
WIN_20180907_15_35_09_Pro_00012
WIN_20180907_15_35_09_Pro_00014
WIN_20180907_15_35_09_Pro_00016
WIN_20180907_15_35_09_Pro_00018
WIN_20180907_15_35_09_Pro_00020
WIN_20180907_15_35_09_Pro_00036
WIN_20180907_15_35_09_Pro_00038
WIN_20180907_15_35_09_Pro_00040
WIN_20180907_15_35_09_Pro_00042
WIN_20180907_15_35_09_Pro_00044

Fig.1: Sample of training data

## Goals of this Project

In this project, we will build a model to recognise 5 hand gestures.

Our task is to train a model on the 'train' folder which performs well on the 'val' folder as well (as usually done in ML projects). We have withheld the test folder for evaluation purposes - your final model's performance will be tested on the 'test' set.

## Two types of architectures suggested for analysing videos using deep learning:

1. **3D Convolutional Neural Networks (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 *100 x 100 x 3*, for example, the video becomes a 4D tensor of shape *100 x 100 x 3 x 30* which can be written as *(100 x 100 x 30) x 3* where *3* is the number of channels. Hence, deriving the analogy from 2D convolutions where a 2D kernel/filter (a square filter) is represented as *(f x f) x c* where *f* is filter size and *c* is the number of channels, a 3D kernel/filter (a *'cubic'* filter) is represented as *(f x f x f) x c* (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 *(100 x 100 x 30)* tensor.

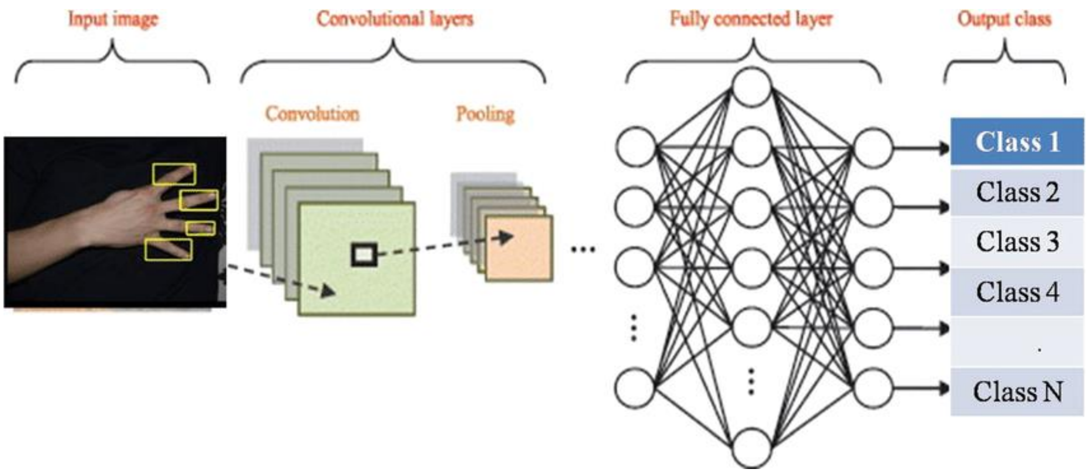

Fig.2: An example for Internal architecture of CNN for gesture recognition.

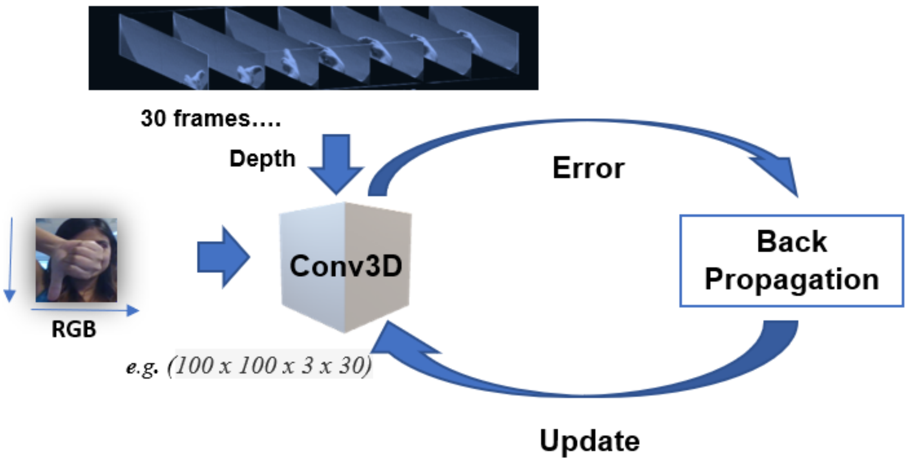

Fig.3: Representation of working of 3D-CNN model.

2. **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).

## Understanding Generators

Creating data generators is probably the most important part of building a training pipeline. Although libraries such as Keras provide builtin generator functionalities, they are often restricted in scope and we have to write our own generators from scratch. For example, in this problem, we are expected to feed *batches of videos*, not images.

In the generator, we are going to pre-process the images as we have images of 2 different dimensions (*360 x 360* and *120 x 160*) as well as create a batch of video frames. The generator should be able to take a batch of videos as input without any error.

## NN Architecture development and training

- Experimented with different model configurations and hyper-parameters and various iterations and combinations of batch sizes, image dimensions, filter sizes, padding and stride length were experimented with. We also played around with different learning rates and *ReduceLROnPlateau* was used to decrease the learning rate if the monitored metrics (*val_loss*) remains unchanged in between epochs.
- We experimented with *SGD()* and *Adam()* optimizers but went forward with *Adam()* as it lead to improvement in model's accuracy by rectifying high variance in the model's parameters. We were unsupportive of experimenting with *Adagrad()* and *Adadelta()* due to the limited computational capacity as these take a lot of time to converge because of their dynamic learning rate functionalities.
- We also made use of *Batch Normalization*, *pooling* and *dropout layers* when our model started to overfit, this could be easily witnessed when our model started giving poor validation accuracy inspite of having good training accuracy.
- *Early stopping* was used to put a halt at the training process when the *val_loss* would start to saturate / model's performance would stop improving.

## Observations

- It was observed that as the Number of trainable parameters increase, the model takes much more time for training.
- **Batch size is directly proportional to GPU memory.** A large batch size can throw *GPU Out of memory error*, and thus here we had to play around with the batch size till we were able to arrive at an optimal value of the batch size which our GPU could support.
- Increasing the batch size greatly reduces the training time but this also has a negative impact on the model accuracy. This made us realise that there is always a trade-off here on basis of priority -> If we want our model to be ready in a shorter time span, choose larger batch size else you should choose lower batch size if you want your model to be more accurate.

- *Data Augmentation* and *Early stopping* greatly helped in overcoming the problem of overfitting which our initial version of model was facing.
- *CNN+LSTM* based model with *GRU* cells had better performance than *Conv3D*. As per our understanding, this is something which depends on the kind of data we used, the architecture we developed and the hyper-parameters we chose.
- *Transfer learning* **boosted** the overall accuracy of the model. We made use of the *MobileNet* Architecture due to it's light weight design and high speed performance coupled with low maintenance as compared to other well-known architectures like VGG16, AlexNet, GoogleNet etc.
- For detailed information on the Observations and Inference, please refer the following Table.

| Architecture | Model No. | RESULTS | DECISION + Explanation | PARAMETERS |
|---|---|---|---|---|
| CNN – 3D | 1 | Training Accuracy : 0.81<br>Validation Accuracy : 0.85 | We started with Image size 64, # Channels 1, Batch size 30, # Gestures 30, Epochs 20, # Dense neurons 128. Statistically simple model with good validation accuracy. | 3,57,541 |
| | 2 | Training Accuracy : 0.85<br>Validation Accuracy : 0.85 | Same parameters except # dense neurons increased to 256 & hence increase in the # parameters and training accuracy. | 22,18,501 |
| | 3 | Training Accuracy : 0.49<br>Validation Accuracy : 0.60 | # Gestures were reduced to 16, dense neurons to 128 & hence accuracies decreased. Model underfit. | 22,18,501 |
| | 4 | Training Accuracy : 0.56<br>Validation Accuracy : 0.69 | # Dense neurons increased to 256 & # gestures kept same as Model 3. Thus, parameters got reduced but model still underfits. | 6,44,773 |
| CNN + LSTM | 5 | Training Accuracy : 0.84<br>Validation Accuracy : 0.74 | We introduced LSTM with CNN. Increase in batch size to 30 & epochs to 20. This showed good results. | 6,80,357 |
| | 6 | Training Accuracy : 0.87<br>Validation Accuracy : 0.85 | Batch size reduced to 30, epochs reduced to 25. Image size increased to 128. This also showed good results. | 25,72,677 |
| | 7 | Training Accuracy : 0.63<br>Validation Accuracy : 0.72 | # Dense neurons increased to 256 & we observed that validation accuracy is better than training accuracy. | 49,16,869 |
| | 8 | Training Accuracy : 0.65<br>Validation Accuracy : 0.74 | # Dense neurons reduced to 64 with negligible change in accuracies wrt model 7. Significant decrease in # parameters. | 18,38,501 |
| CNN + GRU | 9 | Training Accuracy : 0.98<br>Validation Accuracy : 0.92 | We introduced GRU with CNN & with less # parameters. Here we found that learning rate reduced from 0.001 to 0.0002 after 7th epoch. Thus, model overfits. | 8,52,501 |
| | 10 | Training Accuracy : 0.702<br>Validation Accuracy : 0.70 | Image size is now increased from 64 to 100. Dense neurons increased from 64 to 128. Thus, overfit has been removed with increase in parameters. | 20,49,413 |
| | 11 | Training Accuracy : 0.70<br>Validation Accuracy : 0.75 | Batch size increased from 30 to 40, # channels reduced to 1. Rest all parameters are same as model 10. This reduced total # parameters & validation accuracy got slightly increased. | 14,44,261 |
| | 12 | Training Accuracy : 0.96<br>Validation Accuracy : 0.85 | Same parameters as model 11. # Channels is 3. Dense neurons are 64. Again, model showed overfit on training data. | 13,95,589 |
| MobileNet + LSTM | 13 | Training Accuracy : 0.70<br>Validation Accuracy : 0.62 | Now we have considered transfer learning with LSTM with following parameters: Batch size 25, Epochs 20, Image size 100, Dense neurons 128, RNN cells 64. | 26,11,269 |
| | 14 | Training Accuracy : 0.69<br>Validation Accuracy : 0.66 | Batch size 30, epochs 25, Image size 140, dense neurons 256 & RNN cells 256.<br>This showed increment in validation accuracy & obviously # parameters. | 78,31,109 |
| | 15 | Training Accuracy : 0.72<br>Validation Accuracy : 0.76 | We changed the following parameters to reduce the total number of parameters.<br>Image size 64, dense neurons & RNN cells are now 128. Now we observed that model is working better. | 29,96,549 |
| | 16 | Training Accuracy : 0.61<br>Validation Accuracy : 0.69 | Batch size changed to 40, dense neurons changed to 256, image size changed to 128. Rest all remain same. This resulted in decrease in the accuracies & increase in # parameters. | 49,79,781 |
| MobileNet + GRU | 17 | Training Accuracy : 0.68<br>Validation Accuracy : 0.71 | Now we introduced GRU with MobileNet. And the parameters are: Batch size 30, epochs 25, gestures 30, image size 64, dense neurons 128, RNN cells 256. | 42,46,981 |

| | | | | |
|---|---|---|---|---|
| | **18** | Training Accuracy : 0.78<br>Validation Accuracy : 0.89 | Next, we experimented with following changes:<br>Batch size 35, epochs 25, image size 100, dense neurons 64, RNN cells 128 which showed better results in accuracies & decrease in # parameters. | **36,80,581** |
| | **19** | Training Accuracy : 0.87<br>Validation Accuracy : 0.88 | Now we aimed at reducing the # parameters. By changing the batch size 40, epochs 25, image size 64, dense neurons 128, RNN cells 64. And we observed the reduction in # parameters at the same time increment in accuracies. | **34,47,109** |
| | **20** | Training Accuracy : 0.87<br>Validation Accuracy : 0.90 | Here image size increased to 140, dense neurons to 512, RNN cells to 256 which showed increment in # parameters, which was expected, and we observed validation accuracy improved. | **67,06,885** |

## Conclusion:

After doing all the experiments, we would like to select **Model #5** which is **CNN + LSTM** for following reasons:

- Training Accuracy – 84% & Validation Accuracy – 74%
- Number of parameters are 6,80,357 which is less than other models' performance.
- And the model is statistically simple.