Gesture Recognition Project

- **Omkar Joshi – Group Facilitator**
- **Sameer Gadicherla**

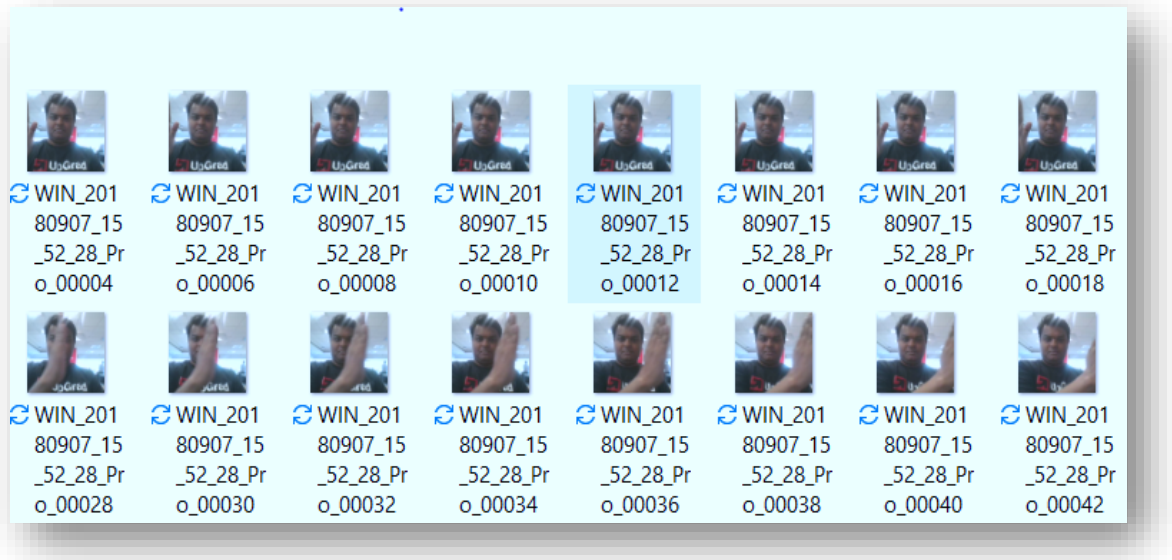
Problem Statement

1. This project involves processing videos of 30 frames each for doing Gesture recognition for smart TV.
2. Images are from two sources where the dimensions are (160,120,3) and (360,360,3)
3. Each Video is tagged with a gesture name amongst the five gesture names.
    - 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

Here's the data: https://drive.google.com/uc?id=1ehyrYBQ5rbQQe6yL4XbLWe3FMvuVUGiL

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 willuse.



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.

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 *100x100x3*, for example, the video becomes a 4D tensor of shape *100x100x3x30* which can be written as *(100x100x30)x3* 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 *(fxf)xc* where *f* is filter size and *c* is the number of channels, a 3D 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

.

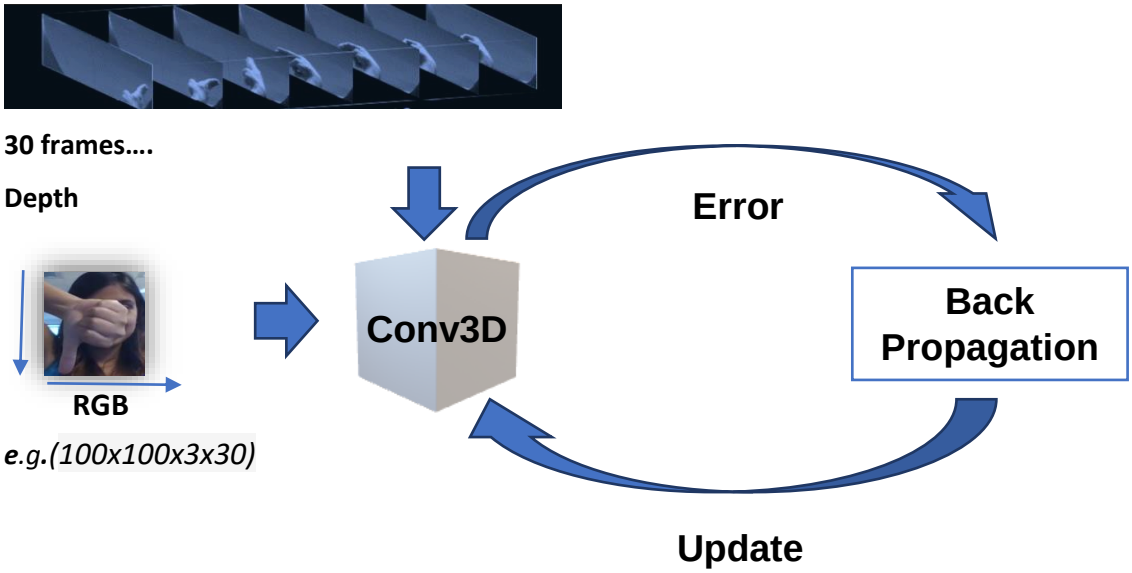


Figure 1: A simple representation of working of a 3D-CNN

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).
   We are going to use Conv2D + GRU architecture as well as transfer learning of mobile net model + GRU model.
   Mobilenet is used because it is lightweight model.

Data Generator

This is one of the most important part of the code. 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. Steps like cropping, resizing and normalization should be performed successfully.

Data Pre-processing

- ***Resizing* and *cropping* of the images.** This was mainly done to ensure that the NN only recognizes the gestures effectively rather than focusing on the other background noise present in the image.
- ***Normalization* of the images.** Normalizing the RGB values of an image can at times be a simple and effective way to get rid of distortions caused by lights and shadows in an image.

Observations

- It was observed that as the Number of trainable parameters increase, the model takes much more time for training.
- 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.
- In our best CONV3D model validation accuracy appears to be 1% more than training accuracy. This happens when the validation set is easier to interpret than the training set. This is NOT a negative sign and is much realistic as you see that the training and validation loss are very close by similar to categorical accuracies.
- *CNN+GRU* 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 below table.

Why did we not choose data Augmentations?

- We might give some wrong information to the labels if we rotate or flip the image as the order of images is crucial.

- Though we could apply few other methods, we have achieved a descent model without augmentation.

- Also there was no class imbalance problem.

MODEL	EXPERIMENT	RESULT	DECISION + EXPLANATION	PARAMETERS
CONV3D	1	Training Accuracy : 0.99 Validation Accuracy : 0.16	Overfitting ☹. Reduce the batch size from 32 to 20 and increase the epochs from 20 to 30	1,155,397 -
CONV3D	2	Training Accuracy : 0.99 Validation Accuracy : 0.81	Overfitting ☹ Let's add some Dropout Layers and decrease batch size more☺	1,155,397
CONV3D	3	Training Accuracy : 0.85 Validation Accuracy : 0.86 *(Best weight Accuracy,Epoch:25/30)*	Val_loss didn't improve from 0.46356. Best model so far for CONV 3D Increase filter size to 3X3X3 and check out results	1,155,397
CONV3D	4	Training Accuracy : 0.94 Validation Accuracy : 0.24	Overfitting☹ Let's reduce batch size from 32 to 20	1,359,685
CONV3D	5	Training Accuracy : 0.82 Validation Accuracy : 0.71	*Don't see much performance improvement. Let's add more epochs and check if model is improving*	1,359,685
CONV3D	6	Training Accuracy : 0.90 Validation Accuracy : 0.73	Model improved in terms of training accuracy but no improvement in validation accuracy. Let's add more convolutional layers.	1,359,685
CONV3D	7	Training Accuracy : 0.97 Validation Accuracy : 0.24	Overfitting☹ Let's reduce batch size from 32 to 20 and check performance	1,618,629
CONV3D	8	Training Accuracy : 0.79 Validation Accuracy : 0.52	Validation accuracy increase. More scope for increase. Let's increase more epochs and check	1,618,629
CONV3D	9	Training Accuracy : 0.84 Validation Accuracy : 0.67	Validation accuracy doesn't seem to increase after 30th epoch	1,618,629
CONV2D+GRU	10	Training Accuracy : 0.95 Validation Accuracy : 0.71	Overfitting☹ Lets increase dropout and increase batch size to see if it reduces overfitting.	1,346,405
CONV2D+GRU	11	Training Accuracy : 0.76 Validation Accuracy : 0.43	val_loss did not improve from 1.57095 Let's use transfer learning	1,346,405
MOBILENET + GRU	12	Training Accuracy : 0.88 Validation Accuracy : 0.81	lets push the accuravy by decreasing batch size & dropout and increasing frames to sample	3,693,253
MOBILENET + GRU	13	Training Accuracy : 0.95 Validation Accuracy : 0.76	Let's increase the batch size to 64 and decrease the frames to sample from 25 to 20 and see if it helps.	3,693,253
MOBILENET + GRU	14	Training Accuracy : 1.00 Validation Accuracy : 0.81	Lets reduce the number of parameters by reducing the gru cells and dense neurons to 64 from 128 each	3,693,253
MOBILENET + GRU	15	Training Accuracy : 0.99 Validation Accuracy : 0.80	lets reduce the batch size to 8 and below configuration with previous increased dense and gru units of 128 , reduce the frames to 15	3,446,725
MOBILENET + GRU	16	Training Accuracy : 0.96 Validation Accuracy : 0.81	Reduced Overfitting from 0.99 to 0.96. Lets try training on all layers of mobilenet model with the same parameters	3,693,253
MOBILENET + GRU	17	Training Accuracy : 0.99 Validation Accuracy : 0.94 *(Best weight Accuracy,Epoch:22/25)*	This is the best model till now with the highest validation accuracy of 94% and least validation loss of just 0.1122	3,693,253

Testing on our generated videos:

```
Use   imageio.imread   instead.
```

```
[ ] batch_labels ##true label is one hot at index 3
```

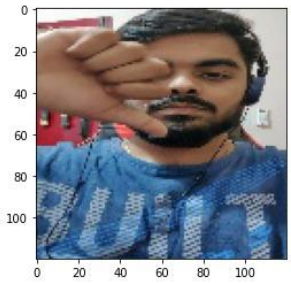
```
array([[0., 0., 0., 1., 0.]])
```

```
[ ] print(np.argmax(model.predict(batch_data[:,:,:,:,:]),axis=1)) ## predicted labels
```

```
[3]
```

```
## lets look at an image from this video
plt.imshow(batch_data[0][13])
```

```
<matplotlib.image.AxesImage at 0x7f6955144bd0>
```



As we can see that the index - 3 is for ThumbsDown_new and the video was actually thumbs down. Our final model has done a decent job in prediction!

Conclusion:

After many different experiments with architecture and hyper parameters we finally have two best models one with Conv3D and another with Mobilenet+GRU.

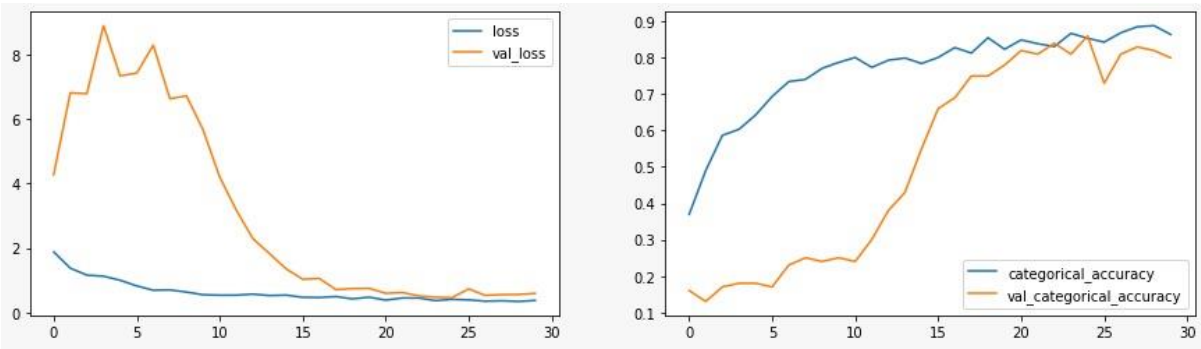
CONV3D Best Model Info:

Model weights size : ~14 MB

Training Accuracy : 0.85

Validation Accuracy : 0.86

Total Parameters : 1,155,397



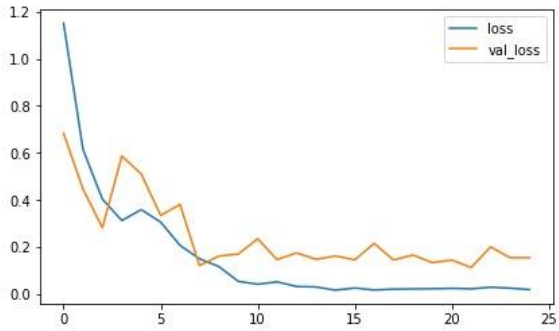
Mobilenet+GRU Best Model Info:

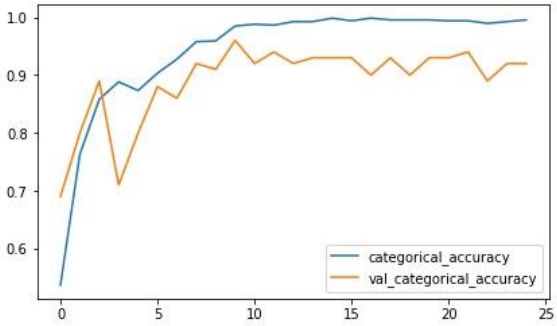
Model weights size :~44 MB

Training Accuracy : 0.99

Validation Accuracy : 0.94

Total Parameters : 3,693,253





Which Model to Choose??

We have got a tradeoff between two models:

1. Model3 -> Conv3D -> Epoch25th Model -> Validation accuracy 86% -> validation loss 0.46 -> Model Size 14MB -> Trainable Params 11,55,397

2. Model17 -> MobileNet(with complete training over all the layers) + GRU -> epoch 22nd Model -> Validation accuracy 94% -> validation loss 0.1122 -> Model Size 44MB -> 36,93,253

We select the second one which is the MobileNet+GRU for the following reasons:

- The validation loss is way lesser ~0.1122 than the conv3D Model with 0.46

- The accuracy is 94% which is remarkable than Conv3D model with 86%

- Trainable params are almost thrice but a 44MB model is not a huge model when compared to CNNs which end up in GBs.

- Models in MBs are easily deployable on smart devices.