Gesture Recognition Project

By: Siddhant Pattanaik

Problem Statement

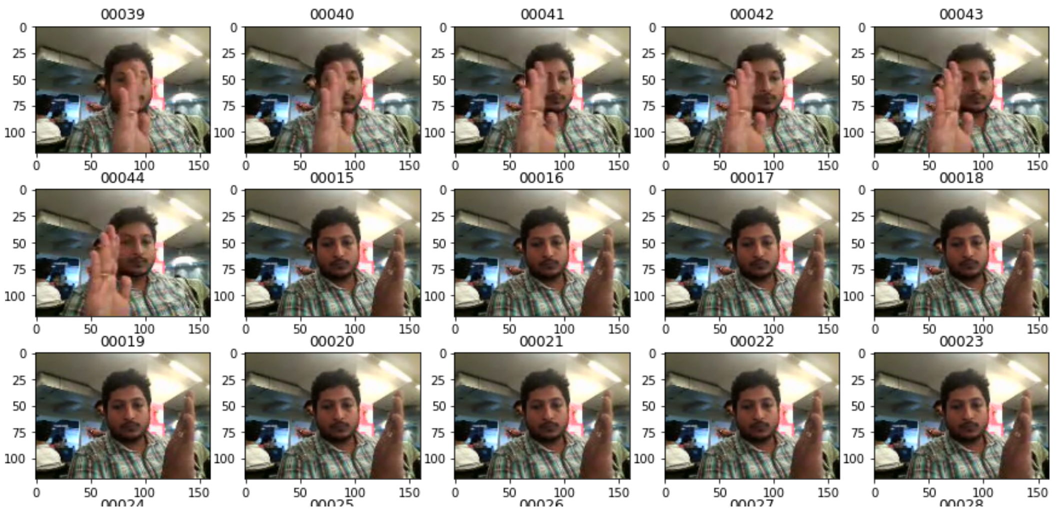
Build a model which can recognize five different hand gestures to control a smart TV. The five gestures are below:

Gesture	Action
Thumbs Up	Increase the volume
Thumbs Down	Decrease the volume
Left Swipe	‘Jump’ backward 10 seconds
Right Swipe	‘Jump’ forward 10 seconds
Stop	Pause the movie

Learnings from Exploratory Data Analysis

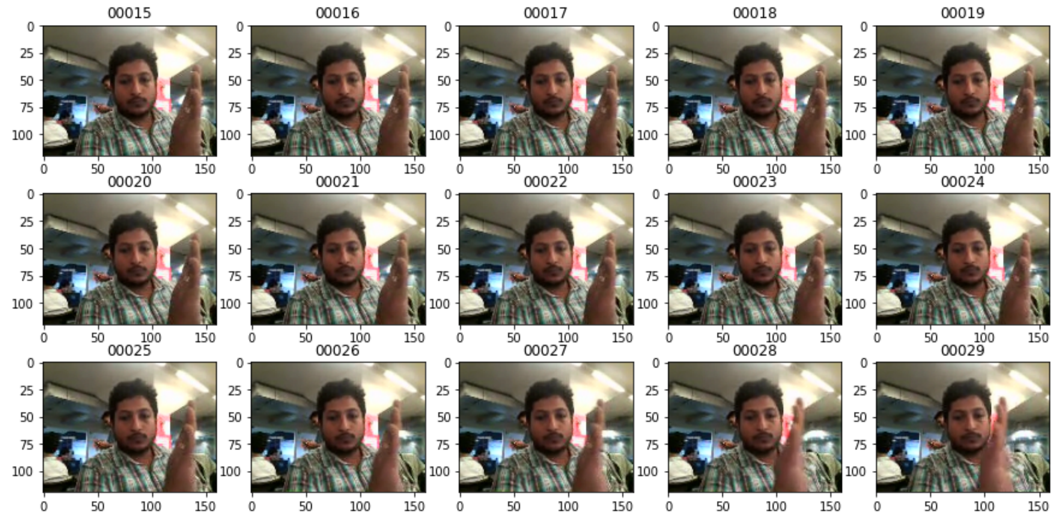
1. **Frames from each sequence, when read in, are not strictly sequential, i.e., the frames are not read sequentially from the beginning.**
   From the abrupt change in the gesture in the 2nd row below as well as the file numbers on top of each image, it can be concluded that the last 6 frames of each sequence are being read in first, followed by the rest of the video in sequence. **Hence, the frames need to be sorted using their filenames (frame number) prior to input of a video in our generator function.**

Example Sequence as it was read in originally (first 15 frames displayed). Frame no. Jumps from 44 to 15 in the 2nd row, due to improper sorting.



It is important that the data being fed into the network(s) is sorted sequentially, since that is the very essence of the training of a sequential model. This needs to be incorporated in the generator function defined later in the notebook.

Sorted sequence:



2. There are 2 possible dimensions for all the videos in the dataset:
    - **(120, 160, 3)**
    - **(360, 360, 3)**

We will need to account for this and resize one / both of the dimensions to bring them on the same footing for Convolutional layers to work properly.

3. **Resizing / Cropping inferences:**
    - Resizing the images from (120,160,3) to a square size of (120,120,3) squishes the image horizontally, but otherwise retains all/most of the features from the original image. On the other hand, cropping the images loses significant amount of information.
    - A judgement call needs to be made on whether cropping would be beneficial in any way. Common logic would dictate that we should try to resize the image instead of cropping them to retain as much information as possible.
    - Resizing the images from (360,360,3) to a lower size of (120,120,3) does not noticeably deteriorate the image, at least visually. All/most of the features we want to capture is more or less intact. Hence, we should go ahead with the resizing to significantly reduce the model training time.
    - We use an input image shape of (120,120,3) as our starting point with the option of increasing or decreasing the preferred shape of input images as required.

Generator function(s)

A custom generator function is useful when we are dealing with large datasets, since it will not be possible to load all data in memory simultaneously.
Hence, the generator function can supply one batch of data at a time to the model when the model requires it. This is made possible by using the ‘yield’ command in the generator function instead of a ‘return’ statement.

The generator function(s) defined in the attached Jupyter notebook can adapt to changes in preferences for the following parameters:

- Batch sizes
- No. of frames per sequence to be included.
- Preferred input shape of the images. As seen in EDA above, the images in the data can be of 2 shapes and may need to be resized/cropped.
- Option to crop the images, if so desired.

They perform the following functions:

- Creating a framework where the indexes of the images generated are based on the number of frames chosen and are centred around the centre of the complete sequence. The logic for doing this is that most of the valuable information in a sequence is likely to be in the middle of the sequence rather than in the first few or last few frames.
    - If all 30 frames are to be included, index would be from 0 to 29.
    - If 18 frames per sequence are chosen, indexes would be from 6 to 23.
- Sort the images from each video in sequential order as mentioned in the EDA section earlier.
- Generate batches of data when requested during model training, including accounting for non-full size batches at the end of each epoch.
- Resize and/or crop the image as required.
- Normalize the images to aid in the training process.

The generator function has been split into 2 parts for ease of use and understanding.

- **"generator"**: Primary function called prior to model training.
- **"obtain_batch"**: Secondary function which actually does the heavy lifting of generating a batch of data. It is invoked by the primary 'generator' function.

Functions for model training, callbacks definition, plotting model metrics.

Function: "train_model"

While calling this function, most of the arguments remain the same from model to model. The only arguments which need to be updated are:

- "model": The model architecture defined immediately before this function is called
- "batch_size": Self- explanatory
- "num_epochs": No. of epochs of training to be run.
- "num_frames": No. of frames per sequence to be considered for training. Can stay mostly constant unless results are poor
- "pref_shape": Preferred shape of the images input to the model. The images are then resized accordingly in the generator function.
- "crop": True or False, If true, the non-square images would be cropped by 20 pixels each from left and right to bring them to a square shape.

The function also has callbacks defined for later use.

- Checkpoint for saving the best weights and architecture of the model as an '.h5' file.
- Reducing the learning rate if a plateau is encountered in evaluation metrics (val_loss)
- Early Stopping after 10 continuous epochs of very little improvement in val_loss.

The function finally 'fits' the training data as required.
It avoids the need of re-running all of the steps within this function for each model and can simply be invoked whenever a model needs to be trained.

Function: "plot_results"

There is also a separate function defined to plot the results of training and validation metrics.

The above functions significantly reduce the amount of repetitive code that need to be run while training and evaluating each model.

Modelling Process

The modelling process can be split into 4 sections enumerated below. Generally, the inferences from previous models have been incorporated into the architecture of latter models.

1. Memory / GPU RAM Utilization Tests
   This is done to obtain optimal batch sizes, number of frames per sequence etc. to efficiently use the GPU memory available for training.
2. 3D Convolution models
3. 2D Convolution + RNN models
4. Transfer Learning (instead of 2D conv layers) + GRU

The architecture, hyperparameters, results and inferences from each of the models tested have been tabulated below.

Memory / GPU RAM Utilization Tests						
Experiment No.	**Model No.**	**Batch Size**	**No. of frames per sequence**	**Architecture**	**Results**	**Inferences & Explanations**
1	0	**256**	**30**	• 4 x Conv3D layers (16,32,64 & 128 neurons respectively) • 2 x Fully-connected layers (128 neurons) • 1 x Softmax Output layer • Activation function for hidden layer: 'relu' • Optimizer: Adam (Default learning rate)	"Resource Exhausted Error: Out of Memory (OOM)"	• High batch size and inclusion of all frames per sequence leading to memory exhaustion. **Action: Reduce batch size to 128.**
2	0	**128**	**30**	Same as above	"Resource Exhausted Error: Out of Memory (OOM)"	• High batch size and inclusion of all frames per sequence leading to memory exhaustion. **Action: Reduce no. of frames per sequence to 20 from 30.**
3	0	**128**	**20**	Same as above	"Resource Exhausted Error: Out of Memory (OOM)"	• High batch size leading to memory exhaustion. **Action: Reduce Batch size to 64 and use all frames per sequence.**
4	0	**64**	**30**	Same as above	Model runs - Memory Usage: 15.7GB / 16GB	**Very good memory utilization.**
5	0	**64**	**20**	Same as above	Model runs - Memory Usage: 9.6GB / 16GB	**Good memory utilization.**
6	0	**32**	**30**	Same as above	Model runs - Memory Usage: 9.6GB	**Good memory utilization.**

					/ 16GB	

<u>Memory test conclusions:</u>

- Higher batch sizes of 256 & 128 resulted in Out of Memory (OOM) error, even with reduced no. of frames per sequence.
- There was a trade-off seen between batch-size and no. of frames per sequence used when it came to memory usage.
- **Ideally, we would like to include all 30 frames per sequence to <u>include maximum information</u> from the input data.**
- The following possible combinations of batch size may be considered for optimal usage:
    1. Batch size: 64, No. of frames: 30
    2. Batch size: 32, No. of frames: 30
    3. Batch size: 64, No. of frames: 20 (Only use if further memory issues encountered.)
    4. Batch Size: 16, No. of frames: 30 (Only use if further memory issues encountered.)

3D Convolution models					
Experiment No.	**Model No.**	**Batch Size**	**Architecture or Change in Architecture**	**Best Metrics / Results**	**Inferences & Explanations**
7	1	64	• 4 x Conv3D layers (16,32,64 & 128 neurons respectively) Filter: (3,3,3), MaxPooling3D: (2,2,2) • 2 x Fully-connected layers (128 neurons) Dropout: 0.25 for each FC layer • 1 x Softmax Output layer • Activation function for hidden layers: 'relu' • Optimizer: Adam (Default learning rate) • **No cropping** of input images. Only re size.	# parameters: 1,111,941 After 20 epochs: Training acc: **0.9548** Validation accuracy: **0.91**	• Excellent first model results obtained. • Including all 30 frames in each sequence seems to provide good results, as does an input shape of (120,120,3). We shall stick to these parameters unless there is a significant drop in accuracy, which would warrant the change. • Batch size of 64 just about able to fit in memory as well throughout training process without any errors. • There is also very little overfitting observed, which is good.
8	2	64	• **Cropping applied to input images** with original shape of (120,160,3) - 20 pixels from each side cropped to bring the input images down to (120,120,3)	# parameters: 1,111,941 After 20 epochs: Training acc: **0.9382** Validation accuracy: **0.88**	• Not much difference between model 1 and 2 with both models providing similar results. If anything, cropping some of the images slightly deteriorates the results. This is expected as we are losing some information by cropping. • **Hence, we shall avoid cropping any of the images in future models and would stick to simply resizing both the shapes of images to our preferred input shape for retention of all/most image information.**
9	3	64	• **Doubling the number of conv3D layers from model 1 to see the results.** • 8 x Conv3D layers (Creating a replica of each Conv 3D layer from Model 1) • 2 x Fully-connected layers (256 neurons) Dropout: 0.25 for each FC layer • 1 x Softmax Output layer	# parameters: 2,552,565 After 21 epochs: Training acc: **0.8477** Validation	• Addition of more convolutional layers actually deteriorates the results as compared to Model 1. • This leads me to believe that model 1 with a simpler architecture and lesser number of parameters would be able to perform equally better.

			• Activation function for hidden layers: ‘relu’ • Optimizer: Adam (Default learning rate)	accuracy: **0.83**	
10	4	64	• **Reducing the width of the fully-connected layers** to 64 neurons as compared to model 1. This significantly reduces the number of trainable parameters in the network.	# parameters: 697,797 <u>After 30 epochs:</u> Training acc: **0.9261** Validation accuracy: **0.89**	• Reducing the number of parameters actually provides quite similar results as compared to model 1. However, the training time is longer due to this model requiring more epochs to achieve similar results. • Hence, we are probably better off continuing with the original architecture from model 1, which has provided with the best results thus far.
11	5	**32**	• **Reducing the Batch size to 32 and using the same architecture as model 1.**	# parameters: 1,111,941 <u>After 20 epochs:</u> Training acc: **0.9774** Validation accuracy: **0.93**	• Reducing the batch size helps quicken the training process. High accuracies are obtained quicker (in terms of no. of epochs) than earlier models. • **This may be attributed to the fact that weight updates happen more frequently with a smaller batch size.** • The results with a smaller batch size improve the results compared to our previous best model 1.

<u>3D Convolution models final comments:</u>

- The very first model attempted provided with surprisingly good results. Hence, it was a matter of tweaking the parameters to see if slight improvements can be made.
- This was obtained by simply reducing the batch size to 32 from the original 64 without any change in the model architecture from the first model. The improvement may be attributed to the fact that weight updates happen more frequently with a smaller batch size.
- We shall consider Model 5 as our best model using Conv3D layers. Based on the results, this is certainly a very good model with little to no overfitting.

2D Convolution + RNN models

Experiment No.	Model No.	Batch Size	Architecture or Change in Architecture	Best Metrics / Results	Inferences & Explanations
12	6	32	• 4 x Conv2D layers in a Time Distributed manner (16,32,64 & 128 neurons respectively) Kernel size: (3,3). MaxPooling2D: (2,2) • **1 x LSTM layer (128 cells)** Dropout: 0.3 • 1 x Fully-connected layers (128 neurons) Dropout: 0.3 • 1 x Softmax Output layer • Activation function for hidden layers: 'relu' • Optimizer: Adam (Default learning rate)	# parameters: 3,391,909 <u>After 20 epochs:</u> Training acc: **0.8582** Validation accuracy: **0.74**	• Definite overfitting observed for this model. More stringent regularization recommended. • Training accuracy is almost 100% towards the latter epochs, while validation accuracy is languishing around 75%. • Number of parameters significantly larger compared to Conv3D models because of the inclusion of LSTM layer.
13	7	32	• Same architecture as model 6 above with dropouts increased to 0.5.	# parameters: 3,391,909 <u>After 20 epochs:</u> Training acc: **0.8597** Validation accuracy: **0.77**	• The overfitting seen in Model 6 has been reduced somewhat, but not significantly. • Perhaps even more regularization is required.
14	8	32	Changes from model 7: • Additional dropouts of 0.25 after each Conv2D layer • No. of LSTM cells reduced to 64.	# parameters: 1,728,677 <u>After 20 epochs:</u> Training acc: **0.8069** Validation	• Model is clearly still overfitting. Adding dropout layers after Conv2D layers does not seem to have helped in any way and has in fact worsened the results. • Perhaps a change in architecture might help.

				accuracy: **0.63**	
15	9	32	• Replacing the LSTM layer from model 7 with GRU layer (128 cells) to see how this performs. • 4 x Conv2D layers in a Time Distributed manner (16,32,64 & 128 neurons respectively) Kernel size: (3,3). MaxPooling2D: (2,2) • **1 x GRU layer (128 cells)** Dropout: 0.5 • 1 x Fully-connected layers (128 neurons) Dropout: 0.5 • 1 x Softmax Output layer • Activation function for hidden layers: 'relu' • Optimizer: **Adam (Learning rate: 0.0001)**	# parameters: 2,572,965 <u>After 30 epochs:</u> Training acc: **0.8658** Validation accuracy: **0.73**	• With a reduced learning rate, the training process is expectedly slower. • However, the results are fairly similar to what was seen using LSTM on the similar architecture of model 7. In hindsight, this is expected since LSTM and GRU networks usually have comparable performances. • As with the LSTM models, there is still significant overfitting observed.
16	10	32	• Reducing the number of neurons in GRU and the fully-connected layers to 64. • Learning rate increased to 0.0003. • All other aspects of model 9 remain the same. • The number of parameters is significantly less compared to model 9.	# parameters: 1,318,821 <u>After 30 epochs:</u> Training acc: **0.9246** Validation accuracy: **0.82**	• Though the accuracy numbers increased from the previous model by reducing the network width, it did not resolve the overfitting issues in any way. • Perhaps adding another GRU layer may help.
17	11	32	• Adding an additional GRU layer. • 4 x Conv2D layers in a Time Distributed manner (16,32,64 & 128 neurons respectively) Kernel size: (3,3). MaxPooling2D: (2,2) • **2 x GRU layers (64 cells)** Dropout: 0.3	# parameters: 1,343,781 <u>After 20 epochs:</u> Training	• While the training accuracy has improved to >97% using 2 GRU layers, overfitting is a bigger issue now with validation accuracy at about 80%. • The results seen in models 6 – 11 may be due to **feature extraction in the Conv2D layers being inadequate.**

			• 1 x Fully-connected layers (64 neurons) Dropout: 0.4 • 1 x Softmax Output layer • Activation function for hidden layers: ‘relu’ • Optimizer: Adam (Default learning rate)	acc: **0.9713** Validation accuracy: **0.81**	Applying transfer learning with some pre-defined models may be more suitable.

<u>2D Convolution + RNN models comments:</u>

- As seen in the table above, none of the models (6 – 11) were able to provide satisfactorily good results with validation data, though training accuracies go up to the high nineties.
- Overfitting remains a concern.
- While more/better regularization might help with these issues, it may be that the Conv2D architecture being used is inadequate in extracting features. This may call for a more refined architecture for feature extraction, which may be provided by a tried and tested pre-defined model. This transfer learning approach followed by RNN layer(s) has been tried next.

Transfer Learning (instead of 2D conv layers) + GRU

Experiment No.	Model No.	Batch Size	Architecture or Change in Architecture	Best Metrics / Results	Inferences & Explanations
18	12	32	• Using Mobilenet v2 as our pre-defined model since it is a relatively light weight model. • Weights from the pre-defined model **not** being re-trained, they are used as is. This would keep the number of trainable parameters relatively low. • 1 x GRU layer: 128 GRU cells, Dropout: 0.4 • 1 x Fully-connected layer: 128 Dense neurons, Dropout: 0.4, Activation fn: relu • Output softmax layer • Optimizer: Adam (with default learning rate)	# parameters: 4,291,141 # trainable params: 2,033,157 After 20 epochs: Training acc: **0.9774** Validation accuracy: **0.89**	• The weights from the pre-defined model do a good job in ramping up training accuracy fairly quickly. This indicates that the pre-defined model is able to extract features decently. • However, after keeping up with training accuracy for a while, the validation accuracy stagnates around 85%. • There is overfitting which needs to be addressed, potentially by re-training the weights from the pre-defined model. • **The less-than-optimal results may have been because the pre-defined model was trained on different images with different input shapes.**
19	13	32	Changes from model 12: • Weights of the pre-defined model are being re-trained for our use-case. This is because the original Mobilenet model would have been trained on different data which would have different features than our data. • The number of trainable parameters increases significantly. • Dropout after GRU and FC layers increased to 0.5 to address overfitting. • Learning rate reduced to 0.0003	“Resource Exhausted Error: Out of Memory (OOM)” # parameters: 4,291,141 # trainable params: 4,257,029	• Due to us training all the weights in the Mobilenet architecture, the number of trainable parameters has increased significantly. This has resulted in the memory exhaustion error using a batch size of 32. • There are 2 obvious options available o Reduce no. of frames per sequence o Reduce batch size • We shall use the latter since we want to retain as much information in the data as possible. This is the same logic as we followed regarding the decision to not crop the images after model 2.
20	13	16	• Reducing batch size to 16. • Keeping number of frames per sequence at 30 since we want to train with the maximum amount of training	# parameters: 4,291,141 # trainable	• The results are quite promising, but there is some overfitting still present. • The validation accuracy plateaus out at about 91%, while training accuracy is up at

			data possible.	params: 4,257,029 <u>After 20 epochs:</u> Training acc: **0.9970** Validation accuracy: **0.92**	99%. • Possible remedies may be to o Increase regularization o Increase width of the network. o Reducing the learning rate further.
21	**14**	16	Changes from model 13: • Increasing the width of the full connected layer to 256 to see how it performs. • Reducing the learning rate to 0.0001 Architecture: • Using MobilenetV2 as our pre-defined architecture for 2D convolutions. Re-training the weights of this model for our dataset. • 1 x GRU layer: 128 GRU cells, Dropout: 0.5 • 1 x Fully-connected layer: 256 Dense neurons, Dropout: 0.5, Activation fn: relu • Output softmax layer • Optimizer: Adam (Learning rate: 0.0001)	# parameters: 4,308,293 # trainable params: 4,274,181 <u>After 20 epochs:</u> Training acc: **0.9864** Validation accuracy: **0.96**	• This is a very good result as indicated by the metrics, with almost no overfitting. • The training and validation curves also moved together during training, which is always a good sign. • The wider network seems to be able to make better interpretations of the sequences and thereby results in much better validation accuracy values as compared to all of the earlier models. • As mentioned earlier, the pre-defined architecture, once re-trained with our training dataset is able to extract much more information from the input data, compared to Conv2D layers used earlier. • We shall conclude our model testing here having achieved high accuracies which was our primary objective.

<u>Transfer Learning + RNN final comments:</u>

- The inclusion of transfer learning for our feature extraction process (instead of Conv2D layers) provides significantly better results. This highlights the importance of proper feature extraction during the training process.
- The original weights from the pre-trained model are not directly transferable to our use-case, primarily because the training of these weights would have occurred on a different set of data. However, the architecture is able to perform very well once we re-train the network with our data.
- The drawback of this re-training is that the number of trainable parameters increases significantly, resulting in a fairly heavy model. However, the benefits of this are evident from the significant improvement in the accuracy metrics. As such the training time was the same as any of the Conv3D models with lesser parameters.

Conclusions and Best Model(s)

- The results prove that for deep learning models, more data during training usually leads to better training. There was a case to use a subset of the total frames available per sequence to reduce training times, but high accuracy values indicate otherwise. Including the complete sequence allows us to extract more information during training.
- Transfer Learning, using pre-trained models which have gone through robust training and architecture optimization can provide better results, especially for feature extraction, as compared to using multiple Conv2D layers.

Best Model

After going through several types of architectures, we have two models which may be considered as the most suitable for our use.

- **Model 14 (Transfer Learning + GRU)**
  - By sheer accuracy numbers, this is by far the best model. (Training accuracy: 0.9864, Validation accuracy: 0.96)
  - However, due to retraining of weights of the pre-trained MobilenetV2 model, the number of trainable parameters are a bit high (4,274,181)
- **Model 5 (Conv3D)**
  - This is a reaonable second choice model for the task at hand. (Training accuracy: 0.9744, Validation accuracy: 0.93)
  - This model has relatively less number of trainable parameters (1,111,941). So, if memory and time are a consideration, this would pip model 14 for efficiency.

Hence, it essentially boils down to whether we want higher accuracy or better efficiency with slightly compromised accuracy values.

For now, we shall consider **Model 14** as our winner, and if we face any performance lags, we can always revert to Model 5.