Model number	Model Type	Model comments	Parameters(BatchSize/frames/epochs/optimizer)	Number of training parameters	Results	Inference	Aprox.Runtime per epoch
1	Conv3D		12/30/10/adam	63,673,413	OOM	Out of memory	---
2	Conv3D		12/15/10/adam	63,673,413	OOM	Out of memory	---
3	Conv3D		8/30/10/adam	63,673,413	Trial Runs	Trial run to test generator (200 folders)	---
4	Conv3D		8/15/30/adam	63,673,413	full Runs	0.97-0.72 - Overfits and doesn't converge well	~100 sec/epoch
5	Conv3D-complex		8/15/30/adam	642,277	full Runs	0.86-0.78 - Doesn't converge well Plateau	~74 sec/epoch
6	Conv3D-complex		8/15/30/sgd	642,277	full Runs	0.79-0.64 - No overfitting, but low accuracy	~75 sec/epoch
7	Conv-GRU	Without dropout in conv	8/15/30/sgd	5,463,877	full runs	Validation accuracy plateau around Val. acc. 0.25 (No improv)	75 sec/epoch
8	Conv-GRU	with dropout in conv	8/15/30/sgd	5,463,877	full Runs	Overfits	75 sec/epoch
9	Mobinet-GRU	GRU few dropout	8/15/25/sgd	2,463,109	full Runs	0.99-0.87 | 0.0182-0.4712(loss) - Overfits	67sec/epoch
10	Mobinet-GRU	without crop	8/15/30/sgd	2,463,109	full Runs	0.98-0.84 | 0.0521-0.9975(loss) - Overfits comparitively	67 sec/epoch
11	Mobinet-GRU	More dropout with center crop	8/15/30/sgd	2,463,109	full Runs	0.96-0.87 | 0.1394-0.58(loss)	64 sec/epoch

As per the hint given, we set out to try and compare the Conv3D model and CNN-RNN models to figure out which of these performs better for the task of gesture recognition.

For both these models something that is common is the generator function which uses the datatype 'yield' to produce batch data for training the model. As the training data in this case is videos (30 frames of photos in chronological order), its strongly advisable not to store all the training batches at once in the memory, so we use 'yield'. The generator function provided, is modified as per the requirement. As per the requirement, the image is normalized, cropped and resized in the generator function itself (rather than having separate layers in the model).

Apart from picking the right model, the parameters that we can play around with are:

- Batch Size
- Number of frames
- Crop/resize/normalize
- Number of epochs
- Optimizer

<u>Choosing the memory params:</u>

The number of training data available in the beginning is 633, but all the data is not required for testing the generator and dependency of certain parameters. Hence the number of training folders was limited to 200.

We started with a initial batch-size of 12 and got and OOM (Out Of Memory) error, the batch size was gradually reduced until the model could be trained. The highest batch size without causing an OOM error was 8. This finding was consistent with two other experimental models.

Apart from this, the number of frames also plays a crucial role with the memory used to train the model. As it directly depicts the number of frames(images) to be used in the training. It is quite obvious that we do not need all the frames to label the gesture. So we could cut down the number of frames to half and yet convey the same gesture. This helps in speeding up the training process and as well saves enough space to increase the batch size.

Initially, the batch size could not be increased above 4, when all the frames was used. But by picking every alternative frame in the chronological order (15 frames), we were able increase the batch size to 8. The time taken for each epoch was also found to decrease drastically(ref. table for metrics).

Model 1.1: Conv3D

All this was tried out with Conv3D, but these results of the memory parameters are invariant of the model used. And also, this tuning was done not with all the training sample rather with a sub-batch (200 not 633).

The simple Conv3D had only a pair of conv3d layers with a depth of 3, which means whilst extracting the features, it will be looking at 3 frames at a time. This was just an extension of a Conv2D model(in the temporal direction). It was never intended to produce effective results. Rather was used to deduce the optimal img_idx(number of frames-15) and batch_size(8). Additionally it was used to test the functionality of the generator function. In the first couple of runs, as seen in the table, the cloud system ran out of memory, producing OOM error.

After capping the batch_size to 8, the number of frames to be used for training was played with. Using this same simple Conv3D, both the cases of 15 frames and 30 frames was tested using 30 epochs. The categorical accuracy was found to be constant(around 0.2), but the training time was found to drastically decrease (almost by 50% decrease).

It is always preferred to have a scheduled learning rate rather than a fixed one. i.e. A learning rate that dynamically varies w.r.t. time-steps. An advantage of this is, faster (~108 for SGD and ~100 sec/epoch for adam) and more reliable convergence. Hence, we started out with adam to be the optimizer. But unfortunately for this case it was out performed by SGD. This may be because of he explicit use of dynamic learning rate whilst fitting the model. Hence SGD was given more preference throughout.

Model 1.2: Conv3D Complex (for model summary, ref Ipython notebook)

This is quite expected when we use a simple Conv3D model. So, we set forth to add multiple dropout layers with a momentum of 0.3-0.4 in order to reduce the model from memorizing the data. This is what a dropout does, randomly sets certain weights to 0, hence introducing stochasticity and keeping the complexity of the model under control. Along to adding a pair of dropout layers, multiple Conv3D layers were added to improve the feature extraction and obviously these layers were followed by Batch layers and Maxpool3D layers to normalize the weights and consolidate the features respectively. The stride was as well set so as to avoid shape mismatch.

This complexConv3D was as well run using two Optimizers, namely adam and SGD. Though the runtimes were nearly equal (~75 sec/epoch), the accuracy was found to be way better with SGD(refer to table for metrics), for previously discussed reasons. Though this did overfit, it was not as bad as previous attempts. But nonetheless, a convnet-RNN model was as well done for improvisation of the overfitting issue and to reduce the runtime.

Model 2: ConvNet+RNN (For architecture ref. Ipython notebook)

Instead of building our own convolution model, we can also use pre-trained to extract features and boost model performance. As seen from the table, first we tried using a in house made CNN for the ConvNet part. But it was slow to train and it used to overfit quite frequently.

We can use MobileNet model which is pre-trained using a larger dataset, hence drastically fast forwards the training process. We set the number of trainable layers to a reasonable value of 9 in order to make the MobiNet suitable for this purpose. When the trainable layer(hyperparameter) was set to 6, the MobiNet did not perform better than the CNN version. This made sense as Mobinet was not fabricated exactly for this task, so adding more tunable layers, makes it acquainted to the current problem statement.

Started training the Mobinet + GRU model with less dropouts in the first experiment(ref. table), the model started to overfit.

Then, we had increased the dropouts to reduce the overfitting and improve validation accuracy.

At the end of 30 epochs with 15 frames, batch size as 8, 'sgd' as optimizer - we were able to achieve 96% Train accuracy, 87% Validation accuracy, 0.1394 train loss & 0.58 test loss.