

Gesture Recognition (Case Study)

Submitted by:

- Megha Sahay
- Yash Mishra

Problem Statement (Business Understanding)

The problem involves the recognition of gestures (5 kind of gestures) using different architectures of Neural Network. Gestures can be treated as small video sample, which the network should evaluate and categorize them to be belonging to one of the 5 given gestures.

Working Python Environment

Major python package includes tensorflow_gpu 2.5.0 which comes with built-in keras wrapper. For more details on packages please refer to environment_package.txt file provided.

- To train model from scratch ensure that the **save_model** folder is empty and Jupyter notebook are in same folder.
- If model training is interrupted and Jupyter notebook is restarted from first command cell, then all training will start from last saved epoch in metrics_log.csv

Data Provided (Data Understanding)

The data provided consists of following parameters:

1. Video files: 30 frames per video file (Frames given in PNG format)
2. 1 Frame – 3 channels (RGB)
3. Training data frame size
 - a. (360,360,3)
 - b. (160,120,3)
4. Number of training video = 663
5. Number of validation videos = 100

Training Data Distribution

Movement_Type	Folder
Down	137
Left Swipe	136
Right Swipe	137
Stop	130
Up	123

Validation Data Distribution

Movement_Type	Folder
Down	21
Left Swipe	18
Right Swipe	23
Stop	22
Up	16

Data Preparation

Image Resizing Details

We have prepared a function called *image_processor()* function, which provides cropped frames in each video when reading data.

Case when original frame size – (360,360,3)

In such cases, we use *skimage.transform.resize()* a python function to resize the image as per user provided shapes. This has been done because this package internally provides resizing with reference to center.

Case when original frame size – (160,120,3)

In this case, we have built our own logic to trim the image appropriately as per user provided target image size.

Sequence List Generation

We have also built a specific sequence list generation function, which provides the option to select only a series of selected frame from each video.

This function provides 3 options

- choice_of_list=0
 - In this, a list of all 30 frames is returned.
- choice_of_list = 1
 - In this, a list of only alternate number between 0,30 is returned.
- choice_of_list = 2
 - In this, a customized frame list ([0,1,2,3,4,5,6,9,12,15,18,21,24,25,26,27,28,29]) is returned.

The idea is that we pick up all initial frames, jump over alternate frames in the middle, and then pick up all frames from the end. This is because, the frames in the middle, would generally have similar information and not help in model's learning process.

GENERATOR FUNCTION

We have built a specialized generator function, which is common when doing modelling for either Conv3D NN models, or Conv2D+RNN Models. The functions work as follows:

Generator function is heart of complete training process. It pumps batched data to network during learning and prediction both. The function description is given below:

Arguments:

- 1) Source Path - Directory path to be considered for reading video/images frames
- 2) folder_list - Lines from the train_doc we read above.
- 3) batch_size - The batch_size we want to select.
- 4) frame_selection - frame_list obtained from frame_generator (Default - range(30))
- 5) process_input_func - To be provided in case CNN2D+RNN type transfer learning modelling being done.
- 6) base_model - To be provided in case CNN2D+RNN type transfer learning modelling being done.

Working

▪ Case when CNN3D modelling being done

In this case, for each batch (according to batch size), we build:

1. **batch_data** = (batch_size, number_of_frames, image_size_x, image_size_y, n_channels)
2. We normalize each channel (RGB) by dividing the pixel value with 255.

▪ Case when CNN2D+RNN modelling being done (RNN can be any of SimpleRNN/LSTM/GRU)

In this case, for each batch (according to batch size), we build:

1. **batch_data** = (batch_size, number_of_frames, image_size_x, image_size_y, n_channels)
2. reshape batch data as **batch_data.reshape(batch_size * number_of_frames , image_size_x , image_size_y , n_channels)**.
3. Above reshaped NumPy array is sent to process_input_func() of the pre-learned CNN2D function. This will produce modified image vector as per pre-learned CNN2D function (like VGG19/VGG16/etc.)
4. After process_input_func we reshape again to (batch_size, number_of_frames, outputs from CNN2D vector)

Final Output

The final output of the function has a tuple which has the batch_data (processed) and one-hot-encoded Y variable. One-hot-encoded NumPy array will be of size (batch_size, 5) since we have 5 kind of gestures.

Model Building

In general, we have used a common approach for the model architectures. There are some strategies which has been adopted as mentioned below:

- We have used CSVLogger (*tensorflow.keras.callbacks.CSVLogger*) to log the information of epoch, loss, accuracy, val_loss and val_accuracy of each model.
- We have also written custom code to load the model from the last run epoch instead of running the training from epoch 0, in case of any error and for better resource utilization.
- Our callback takes care of saving the model along with model metrics parameter in folder
- We have tried with different batch size but in Conv2D+RNN models the batch size of 32 was giving OOM(out of memory error). Therefore, reduced the batch size to 16.

These are some methods to perform the above-mentioned tasks:

- **get_model_save_path(**args):**
 - This method is used to save model .h5 file in the specified path as save_model/{model_type}/{model_num} if this path does not exist then it will create one to save the .h5 file of the trained model.
- **get_model_callback_list(**args) :**
 - filepath is needed to be passed where the model and corresponding log will be saved.
 - This method will create the model check point which will monitor for the validation loss and will write the log to the metric_log.csv file for each model. Learning rate will be reduced to 0.001 when the model metrics stops improving in other words model stops learning.
- **get_latest_check_point(**args):**
 - This will return the latest .h5 file with its paths information so that when we start training a model and in case GPU crash or any other unforeseen error, we will be able to start training from that specific epoch without any need to train from initial epoch. This will save GPU and is less time taking.
- **get_best_check_point(**args):**
 - This method will give the file path of the model with maximum val_categorical_accuracy.
- **draw_evaluation_metric(**args):**
 - As we are writing the log in csv file this method will pick the metric from the csv log file and will plot the loss and accuracy of training and validation data.

CNN3D (Convolution Neural Network) Type Networks:

The architecture is very simple and small in nature. The basic principle was to have *Feature Extraction Layer(s)* and a *Dense Fully Connected(s)*. We have experimented extensively with different parameters and details are mentioned below:

Convolution Layer Design

Conv3D-N (Kernel (3,3,3)) – Padding Same -> N means the number of feature maps being output at this layer
Activation Relu
Batch Normalization
MaxPool3D (Kernel (2,2,2))

Fully Connected (FC) Layer Design

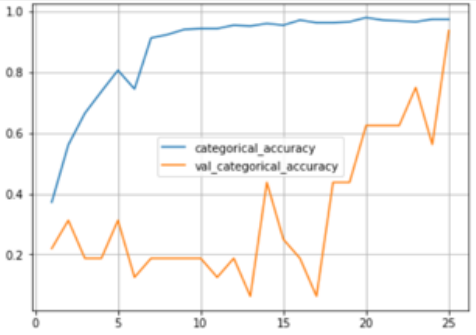
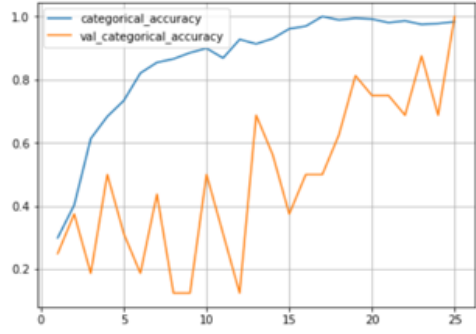

Dense(N) -> N means the number of Neurons in the FC layer
Activation Relu
Dropout
Dense (5) -> 5 number of classes in classification.
Activation Softmax

Design of Experiment for CNN3D

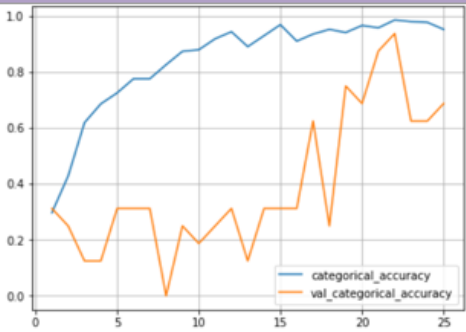
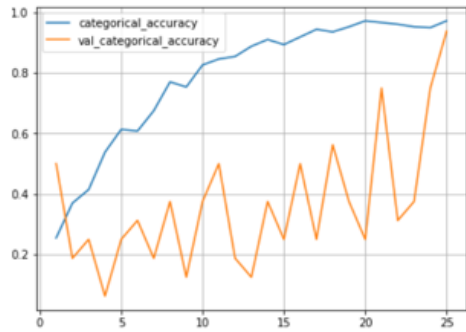
We have experimented the architecture by changing filter size, number of filters, layers, padding (same or valid), max pooling etc. We can see these in detail in subsequent section

CNN3D architecture descriptions and experiments are explained mentioned in subsequent table:

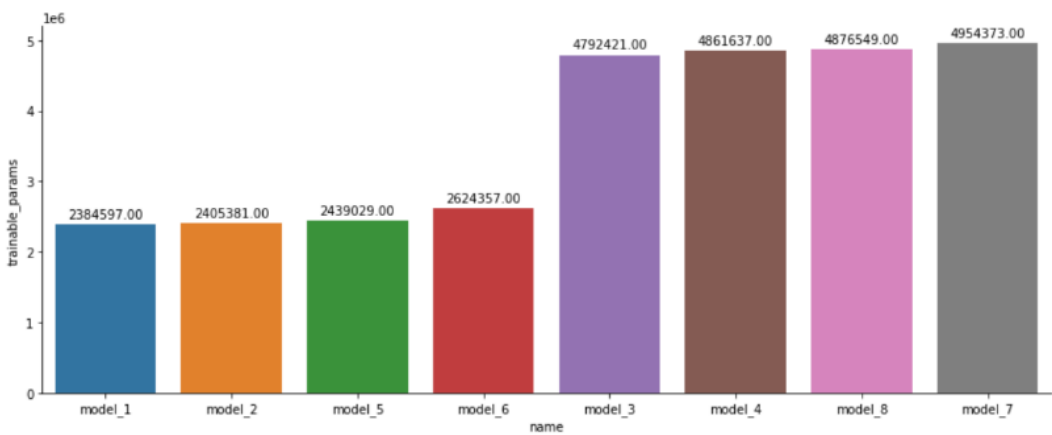
Design of Experiment for CNN3D-Table

Experiment Number	Architecture	Model	Batch size	Epochs	Graph/screenshots (loss,accuracy vs epoch)	Result (validation accuracy and train accuracy)	Decision + Explanation
1	<p>Model 1: Using 16,16,32 kernels in 3 consecutive conv3D layers</p> <ol style="list-style-type: none"> 1. Number of conv layers = 3 2. Number of filters = 16,16,32 3. Filter size = 3 4. padding type = same 5. Fully connected layers = 3 (1 input,1 hidden and 1 softmax layer) 	Conv3D	32	25		Val Accuracy : 0.9375 Train Accuracy : 0.975 Trainable params: 2,384,597	Model is very good now, let's try increasing the number of filters/kernels and see the results.
2	<p>Model 2: Increasing the number of filters/kernels.</p> <ol style="list-style-type: none"> 1. Number of conv layers = 3 2. Number of filters = 16,32,32 3. Filter size = 3 4. padding type = same 5. Fully connected layers = 3 (1 input,1 hidden and 1 softmax layer) 	Conv3D	32	25		Val Accuracy :1.0 Train Accuracy : 0.983 Trainable params: 2,405,381	This model is even better than the first one with very good validation accuracy.We will try increasing the number of filters and see what happens to the model accuracy.
3	<p>Model 3: Increasing the number of filters in third layer to 64.</p> <ol style="list-style-type: none"> 1. Number of conv layers = 3 2. Number of filters = 16,32,64 3. Filter size = 3 4. padding type = same 5. Fully connected layers = 3 (1 input,1 hidden and 1 softmax layer) 	Conv3D	32	25		Val Accuracy : 0.812 Train Accuracy : 1.0 Trainable params:4,792,421	This model is an overfitted model as seen the training accuracy is very high and validation accuracy is low.Number of training params is high. Lets try decreasing the number of filters in the next model.

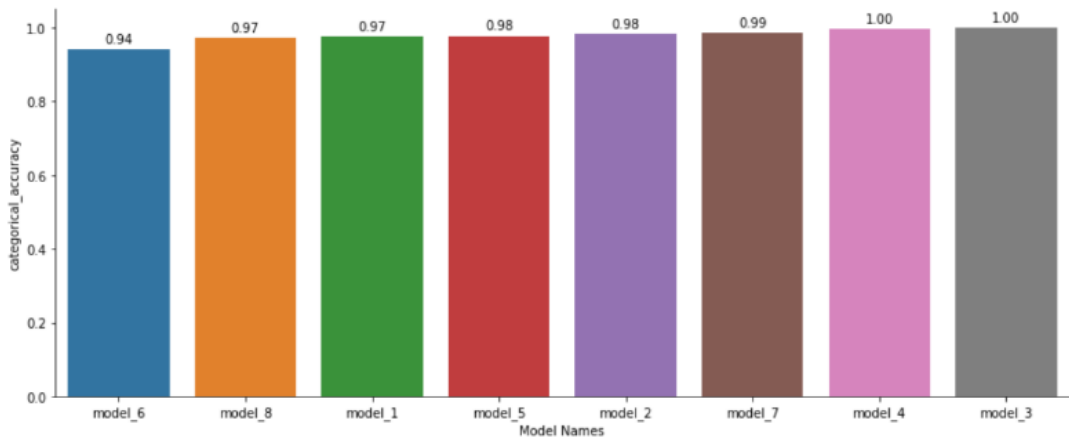
Experiment Number	Architecture	Model	Batch size	Epochs	Graph/screenshots (loss,accuracy vs epoch)	Result (validation accuracy and train accuracy)	Decision + Explanation
4	Model 4: Using 16,64,64 kernels/filters in all the 3 layers. 1. Number of conv layers = 3 2. Number of filters = 16,64,64 3. Filter size = 3 4. padding type = same 5. Fully connected layers = 3 (1 input,1 hidden and 1 softmax layer)	Conv3D	32	25		Val Accuracy : 0.812 Train Accuracy : 0.997 Trainable params: 4,861,637	Unable to overcome overfitting issue even after reducing the number of filters. This model is clearly an overfit as there is large difference between train and validation accuracy. Adding additional dense layer now to resolve overfitting issue.
5	Model 5 adding additional dense layer with 128 neurons. 1. Number of conv layers = 3 2. Number of filters = 16,32,32 3. Filter size = 2 4. padding type = same 5. Fully connected layers = 4 (1 input,2 hidden and 1 softmax layer)	Conv3D	32	25		Val Accuracy : 1.0 Train Accuracy : 0.977 Trainable params: 2,439,029	Reducing the size of filters and adding additional dense layer with 128 neurons have helped in overcoming the overfitting problem in the last model.
6	Model 6: Increasing the kernel/filter size to 5 x 5. 1. Number of conv layers = 3 2. Number of filters = 16,32,32 3. Filter size = 5 4. padding type = same 5. Fully connected layers = 4 (1 input,2 hidden and 1 softmax layer)	Conv3D	32	25		Val Accuracy : 0.937 Train Accuracy : 0.941 Trainable params: 2,624,357	Increasing the kernel size has significantly improved the overall model accuracy and loss. We will now try to build model with stride 1 and same padding type and see the results.

Experiment Number	Architecture	Model	Batch size	Epochs	Graph/screenshots (loss,accuracy vs epoch)	Result (validation accuracy and train accuracy)	Decision + Explanation
7	Model 7 : With stride of 1 and 'same' padding 1. Number of conv layers = 3 2. Number of filters = 32,64,64 3. Filter size = 3 4. padding type = same 5. Stride = 1 6. Fully connected layers = 4 (1 input,2 hidden and 1 softmax layer)	Conv3D	32	25		Val Accuracy : 0.937 Train Accuracy : 0.985 Trainable params : 4,954,373	Adding stride increases the number of training params but doesnot increase the validation accuracy from the last model.We will now build a model with stride of 2 and valid padding type.
8	Model 8: Added maxpooling with stride2 and valid padding caused error. 1. Number of conv layers = 3 2. Number of filters = 32,64,64 3. Filter size = 3 4. padding type = valid 5. Stride = 2 6. Fully connected layers = 4 (1 input,2 hidden and 1 softmax layer)	Conv3D	32	25	ValueError: Negative dimension size caused by subtracting 2 from 1 for '{{node max_pooling3d_23/MaxPool3D}} = MaxPool3D[T=DT_FLOAT, data_format="NDHWC", ksize=[1, 2, 2, 2, 1], padding="VALID", strides=[1, 2, 2, 2, 1]](activation_23/Relu)' with input shapes: [?,1,12,12,64].	InvalidArgumentError is thrown.	Adding stride of 2,valid padding type and then using MaxPooling layer resulted in reducing the input dimension to a neragtive size hence this error. This error is corrected in next model by removing maxpooling from 2nd CNN layer
9	Model 9: After correcting error in Model 8 by selecting stride of 2 and 'valid' (means no padding) padding. 1. Number of conv layers = 3 2. Number of filters = 32,64,64 3. Filter size = 3 4. padding type = valid in first layer while same in other layers. 5. Removing maxpooling from second layer. 6. Stride = 2 7. Fully connected layers = 4 (1 input,2 hidden and 1 softmax layer)	Conv3D	32	25		Val Accuracy : 0.937 Train Accuracy : 0.971 Trainable params: 4,876,549	This model doesnot improve from the last model in terms of validation accuracy. The number of trainable parameters is also high.

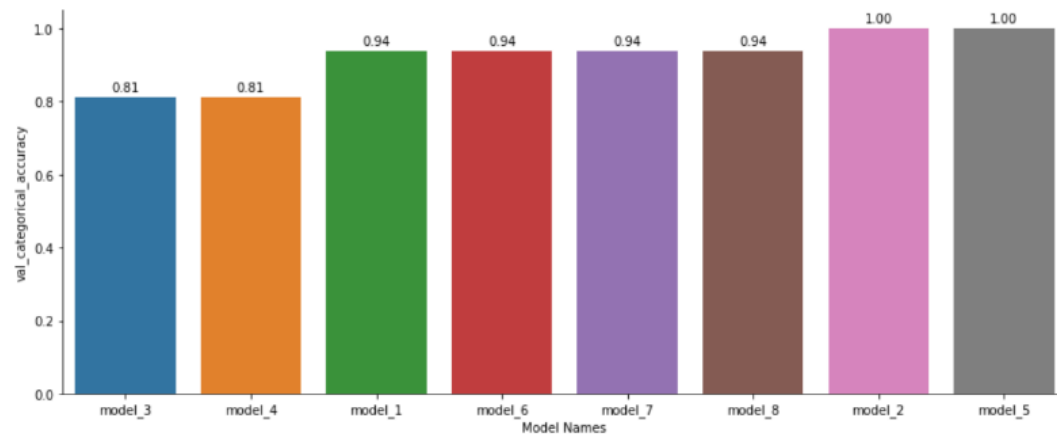
Graphical Comparisons of all the models



Graph: Trainable Parameters Vs Model



Graph: categorical accuracy vs Models



Graph: Validation accuracy Vs Models

Observations

- Model_2 and model_5 seems to be performing good with 100 % of validation accuracy given it has a smaller number of trainable parameters.
- Model_4 and Model_3 is clearly `overfitting` with high training accuracy and low validation accuracy.
- So, Model_2, Model_5, Model_8 is not very good model as compared to others as the difference in training and validation accuracy is high.
- Model_1, Model_5, Model_6, Model_7 is all good enough with less difference in training and validation accuracy and is having high validation accuracy.
- Overall, we can say Model_2 is one of the best having 100 % of validation accuracy given it has a smaller number of trainable parameters.

Final Validation Data Prediction Confusion Matrix (CNN3D Model)

	Predicted_Left	Predicted_Right	Predicted_Stop	Predicted_Down	Predicted_Up
Actual_Left	14	0	2	3	0
Actual_Right	0	20	1	1	0
Actual_Stop	1	0	19	2	0
Actual_Down	0	1	0	19	1
Actual_Up	0	0	2	4	10

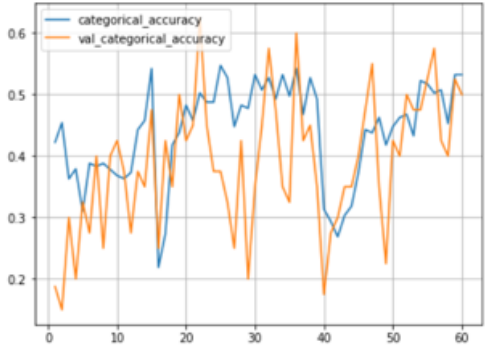
The misses are maximum in detecting *Left Swipe* and *Thumb Up* movement.

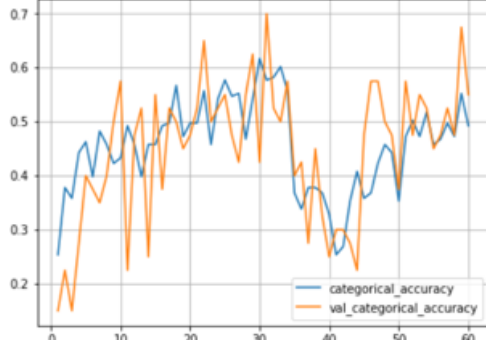
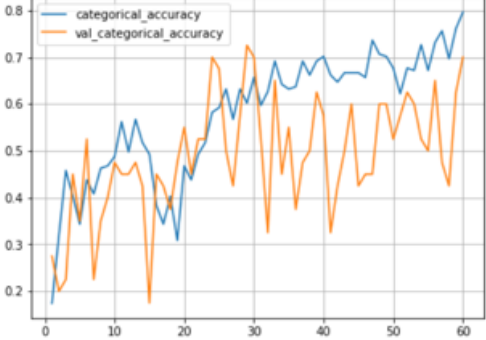
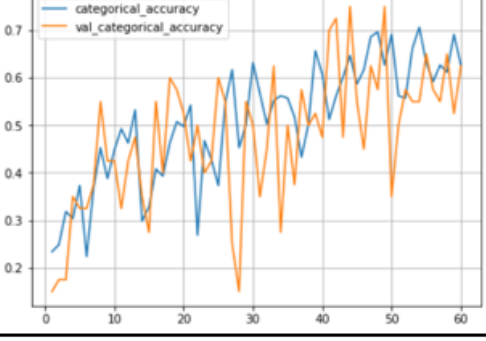
CNN2D+RNN Modelling technique

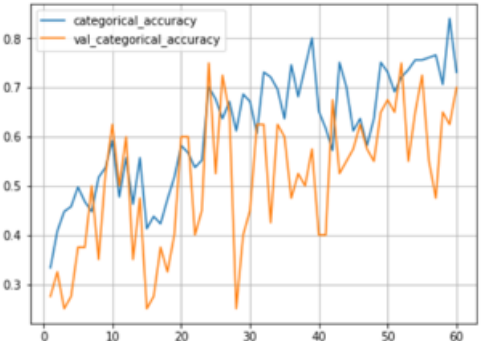
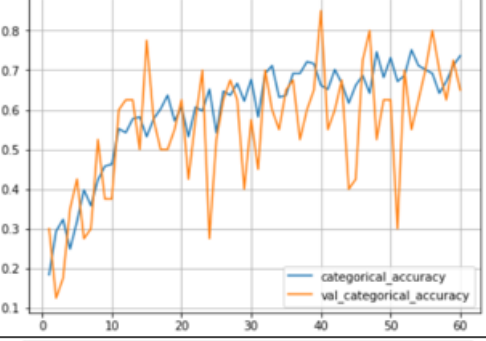
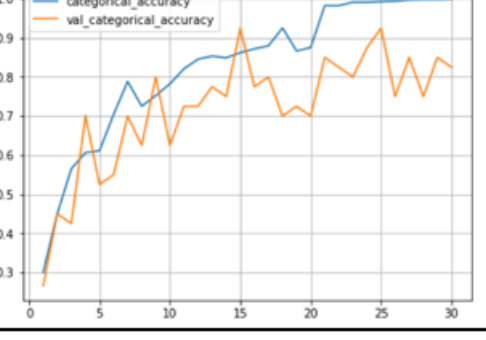
The overall approach for this architecture was to build conv2D with simple RNN model followed by LSTM, GRU, Bidirectional GRU check for trainable parameters, monitor loss and accuracy. The main goal was to minimize the loss and maximize the accuracy on the validation data.

It was seen that when we use transfer learning for base conv2d layer with Simple RNN in Fully Connected layer we could reach very good model accuracy. Following is the detailed information on experiment performed to get best model with high validation accuracy.

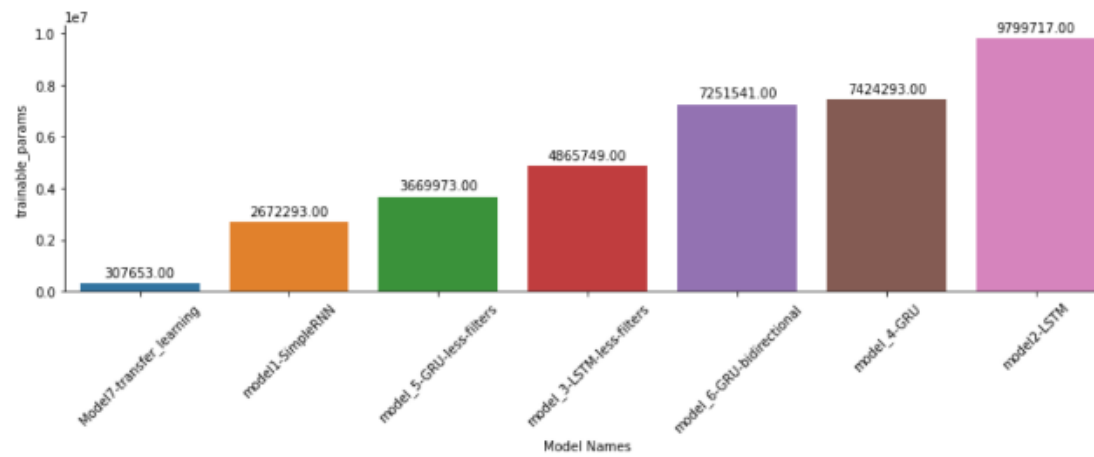
Design of Experiment for CNN2D with RNN/GRU/LSTM/Transfer Learning-Table

Experiment Number	Architecture	Model	Batch size	Epochs	Graph/screenshots (loss,accuracy vs epoch)	Result (validation accuracy and train accuracy)	Decision + Explanation
1	<u>Model 1:Conv2D + Simple RNN Model</u> 1. Number of conv layers = 3 2. Number of filters = 16,64,128 3. Filter size = 3 4. padding type = same 5. Fully connected layers = 3 (1 input,1 hidden SimpleRNN and 1 softmax layer)	Conv2D+RNN	32	60	<p>ResourceExhaustedError: OOM when allocating tensor with shape[32,100,15,100,100] and type float on /job:localhost/replica:0/task:0/device:GPU:0 by allocator GPU_0_bfc</p> <p>[[node gradient_tape/sequential_1/max_pooling3d/MaxPool3D/MaxPool3DGrad (defined at <ipython-input-13-f60efb873de8>:3)]]</p> <p>Hint: If you want to see a list of allocated tensors when OOM happens</p>	ResourceExhaustedError OOM error was seen.	To resolve OOM error the batch size was reduced to 16.
2	<u>Model 1:Conv2D + Simple RNN Model</u> 1. Number of conv layers = 3 2. Number of filters = 16,64,128 3. Filter size = 3 4. padding type = same 5. Fully connected layers = 3 (1 input,1 hidden SimpleRNN and 1 softmax layer)	Conv2D+RNN	16	60		Val Accuracy : 0.625 Train Accuracy : 0.50 Trainable params: 2,672,293	As seen the graph this model has very low validation accuracy. So we will try creating LSTM model and check if we can get better accuracy.

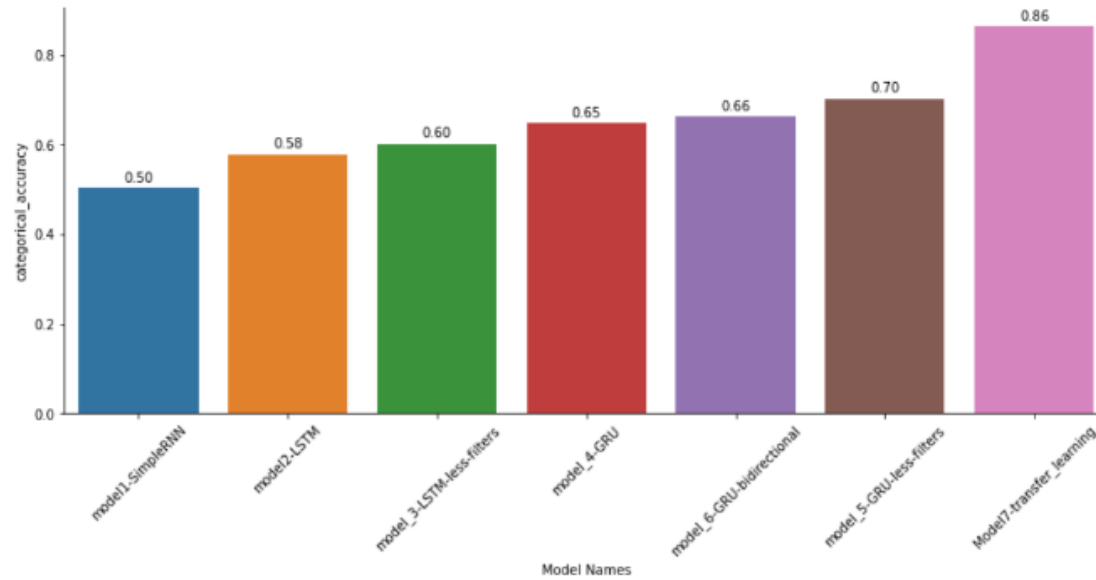
Experiment Number	Architecture	Model	Batch size	Epochs	Graph/screenshots (loss,accuracy vs epoch)	Result (validation accuracy and train accuracy)	Decision + Explanation
3	<u>Model 2:Conv2D + LSTM Model</u> 1. Number of conv layers = 3 2. Number of filters = 32,64,128 3. Filter size = 3 4. padding type = same 5. Fully connected layers = 3 (1 input,1 hidden LSTM and 1 softmax layer)	Conv2D+RNN	16	60		Val Accuracy : 0.699 Train Accuracy : 0.577 Trainable params: 9,799,717	The training and testing accuracy is better than the first model but still very low. So we will try to improve accuracy by decreasing the number of filters in the layers in next model.
4	<u>Model 3:Conv2D + LSTM Model with less filters</u> 1. Number of conv layers = 3 2. Number of filters = 16,32,64 3. Filter size = 3 4. padding type = same 5. Fully connected layers = 3 (1 input,1 hidden LSTM and 1 softmax layer)	Conv2D+RNN	16	60		Val Accuracy : 0.725 Train Accuracy : 0.601 Trainable params: 4,865,749	There seems to be increase in validation accuracy from the previous model but still not upto mark. We will try to build another model by increasing the number of kernels and using GRU.
5	<u>Model 4:Conv2D + GRU Model</u> 1. Number of conv layers = 3 2. Number of filters = 32,64,128 3. Filter size = 3 4. padding type = same 5. Fully connected layers = 3 (1 input,1 hidden GRU and 1 softmax layer)	Conv2D+RNN	16	60		Val Accuracy : 0.75 Train Accuracy : 0.646 Trainable params: 7,424,293	The validation accuracy has increased from the last model. So we will try reducing the number of filters to 16 and see if there is any improvement in validation accuracy.

Experiment Number	Architecture	Model	Batch size	Epochs	Graph/screenshots (loss,accuracy vs epoch)	Result (validation accuracy and train accuracy)	Decision + Explanation
6	Model 5:Conv2D + GRU Model with less filters 1. Number of conv layers = 3 2. Number of filters = 16,32,64 3. Filter size = 3 4. padding type = same 5. Fully connected layers = 3 (1 input,1 hidden GRU and 1 softmax layer)	Conv2D+RNN	16	60		Val Accuracy : 0.75 Train Accuracy : 0.70 Trainable params : 3,669,973	This model seems to be better than the last model as we have managed to increase the validation accuracy and training accuracy. Let us try using bidirectional GRU to get even better results.
7	Model 6 : Conv2D + Bidirectional GRU Model 1. Number of conv layers = 3 2. Number of filters = 16,32,64 3. Filter size = 3 4. padding type = same 5. Fully connected layers = 3 (1 input,1 hidden Bidirectional GRU and 1 softmax layer)	Conv2D+RNN	16	60		Val Accuracy : 0.850 Train Accuracy : 0.661 Trainable params : 7,251,541	This model is having good validation accuracy but low train accuracy. We will try to get better model by using transfer learning and using VGG-19 as pre trained model.
8	Model 7 : Transfer learning Model 1. Base conv2d model = VGG19 2. Fully connected 2 layers of SimpleRNN with 64 neurons and 1 softmax layer.	Conv2D+RNN	32	30		Val Accuracy : 0.925 Train Accuracy : 0.862 Trainable params : 307,653	This model build on a transfer learning approach seems to give good validation and training accuracy. It has better accuracy than SimpleRNN, LSTM, GRU or bidirectional GRU models.

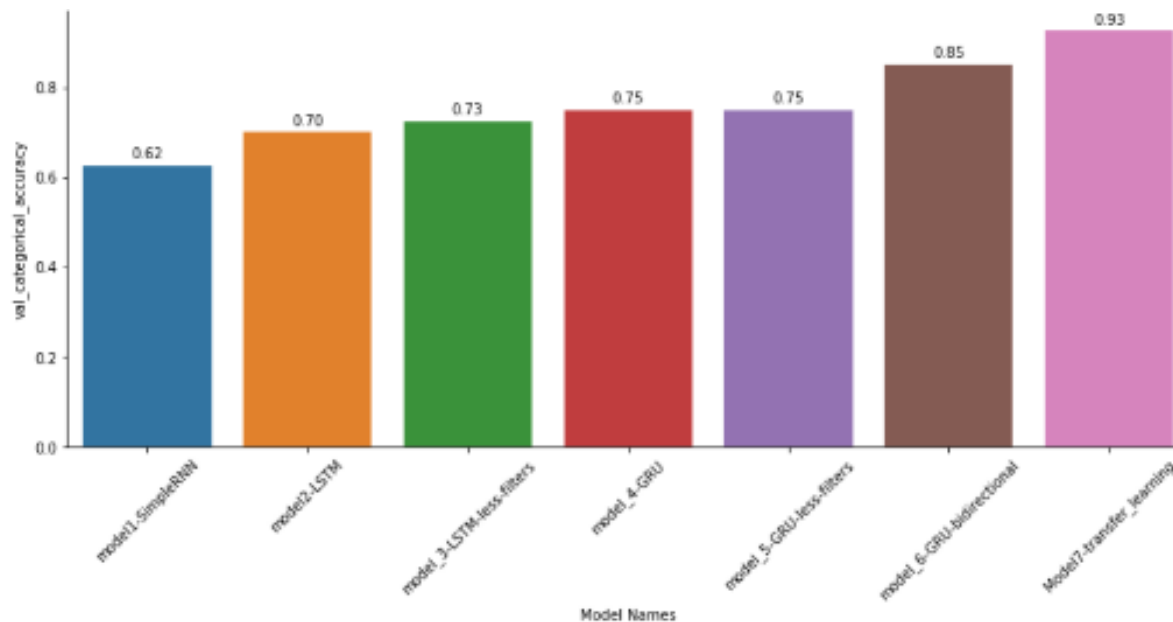
Graphical Comparisons of all the models



Graph: Trainable Parameters Vs Model Names



Graph: Categorical accuracy Vs Model Names



Graph: Validation categorical accuracy vs model names

Observations

- As seen in the above plots the model using transfer learning is performing the best.
- The training and validation accuracy for this model7 is reasonably good.
- The number of trainable parameters for model7 using transfer learning is also comparatively less.
- So, we can consider this model as best among others.
- Other models using LSTM, GRU, Bidirectional-GRU is not as good as Model7-transfer-learning.

Final Validation Data Prediction Confusion Matrix

	Predicted_Left	Predicted_Right	Predicted_Stop	Predicted_Down	Predicted_Up
Actual_Left	11	6	1	4	0
Actual_Right	0	27	0	0	0
Actual_Stop	1	1	25	0	0
Actual_Down	0	2	0	27	0
Actual_Up	0	2	0	3	14

Most of the misses are there in left swipe.

Conclusion

Modelling technique	Training Accuracy	Validation Accuracy
Conv3D	98.3%	100%
Conv2D+GRU	66.1%	85%
Conv2D+LSTM	60.1%	72.5%
Conv2D+SimpleRNN (Transfer learning)	86.2%	92.5%

Important Points:

- It is important to note that Conv2D+GRU and Conv2D+LSTM have very high number of parameters for learning. And we can see that their validation data accuracy is lesser than others. Conv3D and Conv2D+SimpleRNN (Transfer learning) seems to be doing best despite small number of trainable parameters.
- CNN3D model size (h5 file size) is also very large, because of high number parameters, weight values and network architecture details. From that perspective also CNN3D may not be an ideal modelling technique for smaller devices (like smartphones, IOT devices etc.).
- Looking at the confusion matrix, there seems to be a miss happening when differentiating between SWIPE LEFT and SWIPE RIGHT. Obviously these 2 movements are like mirror of each other, and so it can be a problem for model to differentiate them as we are doing time-series type analysis. Therefore, we should consider increasing training data for all 5 gestures, so that model is able to differentiate between such gestures.
- Another way to achieve better accuracy is to build 2 CNN3D (Conv3D) networks. One which works on low resolution images, and other which works on high resolution images. We read a research paper from NVIDIA where such modelling technique was tested for gesture detection using Conv3D networks.

https://research.nvidia.com/sites/default/files/pubs/2015-06_Hand-Gesture-Recognition/CVPRW2015-3DCNN.pdf