Gesture Recognition (Case Study)

PGDMLAI – IIITB – Upgrad

Submitted by:
Abhishek Rajan
Aishwarya Ramachandran
Anugraha Sinha
Vikash Sinha
Google Drive Location Model Code and model Files (h5) : https://drive.google.com/drive/folders/1hK5KwyRHqZGTZE6nEEoSo_-osL6wDO0-?usp=sharing

---

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.

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 channel (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 python's *skimage.transform.resize* 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 build 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

1) choiceoflist = 0
   In this, a list of all 30 frames is returned.
2) choiceoflist = 1
   In this, a list of only alternate number between 0,30 is returned.
3) choiceoflist = 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 FUNCTIONS)

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

This 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) transform_size - The image transformation size we (Default - (120,120)
5) frame_selection_list - frame_list obtained from frame_generator (Default - range(30))
6) process_input_func - To be provided in case CNN2D+RNN type modelling being done
7) base_model - To be provided in case CNN2D+RNN type 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

CNN3D (Convolution Neural Network) Type Networks

We have chosen an architecture that resembles a VGG type architecture. However, smaller in nature. Our 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

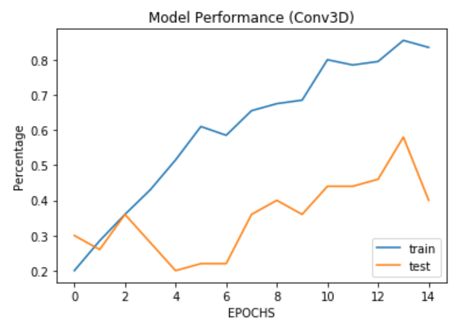
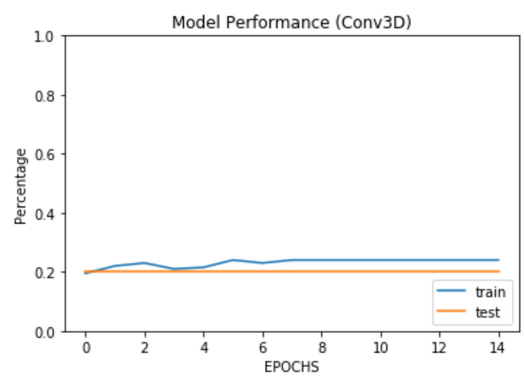
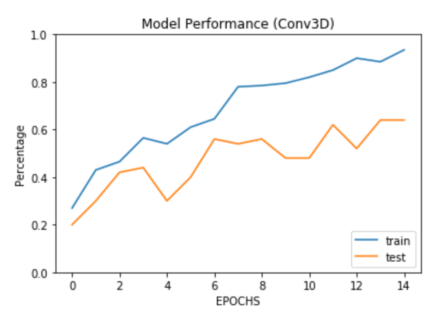
Convolution Layer Design

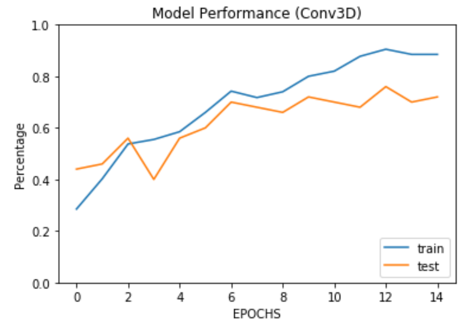
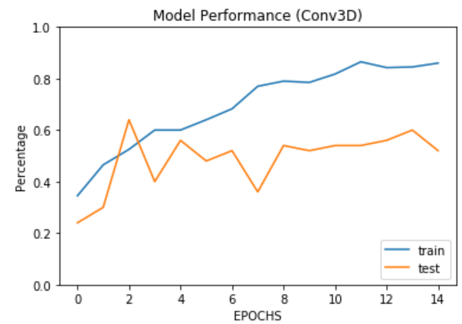
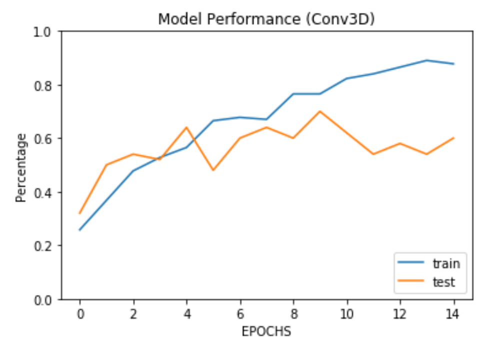
Conv3D-N (Kernel (3,3,3)) – Padding Same -> N means the number of feature maps being output at this layer
Activation Relu
Batch Normalize
Conv3D-N (Kernel (3,3,3))
Activation Relu
Batch Normalize
MaxPool3D (Kernel (2,2,2))
Dropout 0.25

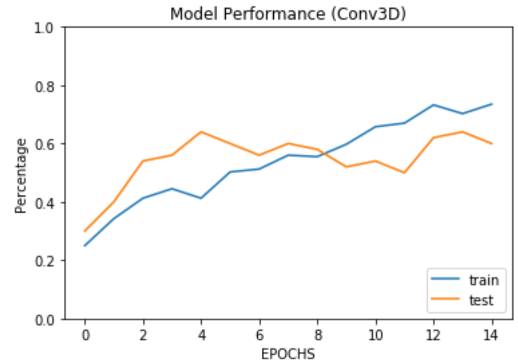
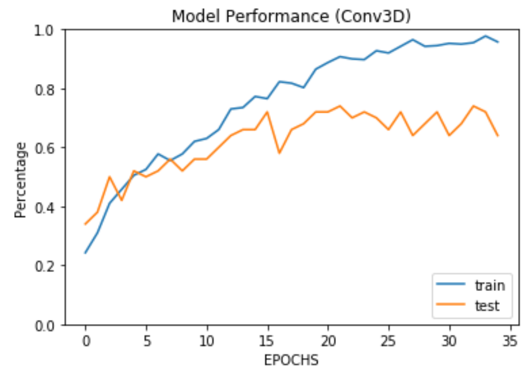
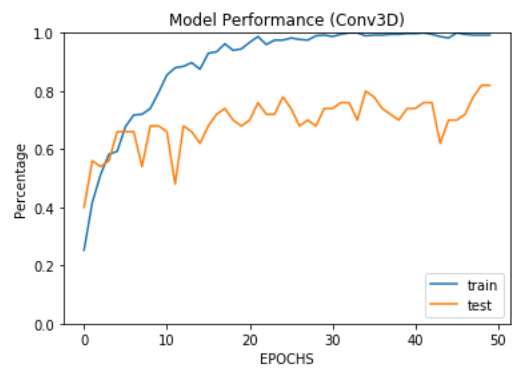
Fully Connected (FC) Layer Design

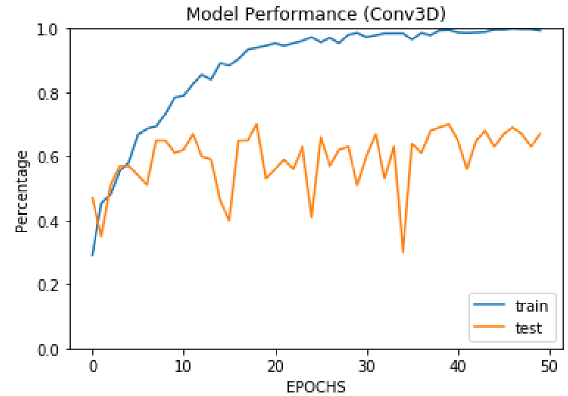
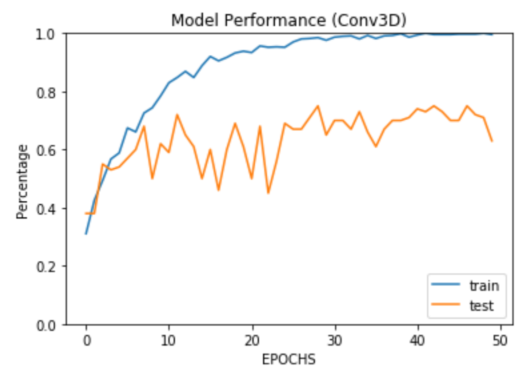
Dense(N) -> N means the number of Neurons in the FC layer
Activation Relu
Dropout

Refer to architecture descriptions mentioned in below table

CNN3D Modeling Experiments Results						
Exp SNo	**Frame Size**	**Framelist**	**Architecture**	**Trainsize**	**Graphs**	**Remarks**
1	(90,90)	All 30	(Convolution and FC layer architecture as above) Conv3D – 16 (Kernel (3,3,3)) + MaxPool(Kernel (2,2,2)) + Dropout(0.25) Conv3D – 32 (Kernel(3,3,3)) + MaxPool(Kernel (2,2,2)) + Dropout(0.25) FC(512) + Dropout(0.25) FC(256) + Dropout(0.25) FC(128) + Dropout(0.25) Dense(5) - Softmax	Train Data = 200 Test Data = 50 Batch Size = 16 Epochs = 15 ReduceLRPatience = 5		Training accuracy is leading up high, but there is high variance. Our first problem is to make training become perfect and make it reach 100% Decision: Increase the number of Conv Layers
2	(90,90)	All 30	(Convolution and FC layer architecture as above) Conv3D – 16 (Kernel (3,3,3)) + MaxPool(Kernel (2,2,2)) + Dropout(0.25) Conv3D – 32 (Kernel(3,3,3)) + MaxPool(Kernel (2,2,2)) + Dropout(0.25) Conv3D – 64 (Kernel (3,3,3)) + MaxPool(Kernel (2,2,2)) + Dropout(0.25) Conv3D – 128 (Kernel(3,3,3)) + MaxPool(Kernel (2,2,2)) + Dropout(0.25) FC(512) + Dropout(0.25) FC(256) + Dropout(0.25) FC(128) + Dropout(0.25) Dense(5) - Softmax	Train Data = 200 Test Data = 50 Batch Size = 16 Epochs = 15 ReduceLRPatience = 5		Increasing the model complexity to so high is very detrimental. There are very high number of parameters to be learnt. We have to reduce them Decision: 1. We increase MaxPooling Kernel Size to (3,3,3) 2. We keep only 2 Conv Layer (32 & 64). 3. We remove Dropout from Conv. 4. FC Relu L2 used
3	(90,90)	All 30	(Convolution and FC layer architecture as above) Conv3D – 32 (Kernel (3,3,3)) + MaxPool(Kernel (3,3,3)) + No Dropout Conv3D – 64 (Kernel(3,3,3)) + MaxPool(Kernel (3,3,3)) + No Dropout FC(512) + Dropout(0.25) + L2 (0.01) FC(256) + Dropout(0.25) + L2 (0.01) FC(128) + Dropout(0.25) + L2 (0.01)	Train Data = 200 Test Data = 50 Batch Size = 16 Epochs = 15 ReduceLRPatience = 5		This is helping a lot. We are reaching uptill 100% training accuracy. Lets increase the number of data points Decision: 1. Increase number of data points for training

			Dense(5) - Softmax			
4	(90,90)	All 30	(Convolution and FC layer architecture as above) Conv3D – 32 (Kernel (3,3,3)) + MaxPool(Kernel (3,3,3)) + No Dropout Conv3D – 64 (Kernel(3,3,3)) + MaxPool(Kernel (3,3,3)) + No Dropout FC(512) + Dropout(0.25) + L2 (0.01) FC(256) + Dropout(0.25) + L2 (0.01) FC(128) + Dropout(0.25) + L2 (0.01) Dense(5) - Softmax	Train Data = 400 Test Data = 50 Batch Size = 16 Epochs = 15 ReduceLRPatience = 5		Increasing number of training points is reducing variance. This is good. Lets try to increase batch size. Decision: 1. Increase batch size to 32
5	(90,90)	All 30	(Convolution and FC layer architecture as above) Conv3D – 32 (Kernel (3,3,3)) + MaxPool(Kernel (3,3,3)) + No Dropout Conv3D – 64 (Kernel(3,3,3)) + MaxPool(Kernel (3,3,3)) + No Dropout FC(512) + Dropout(0.25) + L2 (0.01) FC(256) + Dropout(0.25) + L2 (0.01) FC(128) + Dropout(0.25) + L2 (0.01) Dense(5) - Softmax	Train Data = 400 Test Data = 50 Batch Size = 32 Epochs = 15 ReduceLRPatience = 5		Batch size of 32 does not seem to be good for training and validation. Keeping batch size to 16 only. Decision: 1. Keep batch size = 16 2. Remove BatchNormalization from Conv Layer 3. Remove Dropout from Conv Layer 4. Remove Regu from FC 5. Remove Dropout from FC
6	(90,90)	All 30	(Convolution and FC layer architecture as above) Conv3D – 32 (Kernel (3,3,3)) + MaxPool(Kernel (3,3,3)) + No Dropout + No BatchNorm Conv3D – 64 (Kernel(3,3,3)) + MaxPool(Kernel (3,3,3)) + No Dropout + No BatchNorm FC(512) + No Dropout + no regularization FC(256) + No Dropout + no regularization FC(128) + No Dropout + no regularization Dense(5) - Softmax	Train Data = 400 Test Data = 50 Batch Size = 16 Epochs = 15 ReduceLRPatience = 5		Not much change. Lets increase 1 more FC layer before Softmax, and keep all dropout, BatchNormalization and Regu settings same as before Decision: 1. Increase 1 more FC(64) layer at end (before SoftMax)

7	(90,90)	All 30	(Convolution and FC layer architecture as above) Conv3D – 32 (Kernel (3,3,3)) + MaxPool(Kernel (3,3,3)) + No Dropout + No BatchNorm Conv3D – 64 (Kernel(3,3,3)) + MaxPool(Kernel (3,3,3)) + No Dropout + No BatchNorm FC(512) + No Dropout + no regularization FC(256) + No Dropout + no regularization FC(128) + No Dropout + no regularization FC(64) + No Dropout + no regularization Dense(5) - Softmax	Train Data = 400 Test Data = 50 Batch Size = 16 Epochs = 15 ReduceLRPatience = 5		Increasing FC layer is not helping. Lets keep things simple. Decision: 1. Have only 512->256->128 FC layers 2. Bring L2 regu back (FC) 3. Bring Dropout at FC back (FC) 4. No dropout at Conv. 5. Bing BatchNorm at Conv 5. Increase EPOCHs to 35
8	(90,90)	All 30	(Convolution and FC layer architecture as above) Conv3D – 32 (Kernel (3,3,3)) + MaxPool(Kernel (3,3,3)) + No Dropout + Yes BatchNorm Conv3D – 64 (Kernel(3,3,3)) + MaxPool(Kernel (3,3,3)) + No Dropout + Yes BatchNorm FC(512) + Dropout(0.25) + L2(0.01) FC(256) + Dropout(0.25) + L2(0.01) FC(128) + Dropout(0.25) + L2(0.01) Dense(5) - Softmax	Train Data = 400 Test Data = 50 Batch Size = 16 Epochs = 35 ReduceLRPatience = 5		Good one, we are reaching 100% training accuracy. Decision 1. Increase EPOCHS to 50 same architecture.
9	(90,90)	All 30	(Convolution and FC layer architecture as above) Conv3D – 32 (Kernel (3,3,3)) + MaxPool(Kernel (3,3,3)) + No Dropout + Yes BatchNorm Conv3D – 64 (Kernel(3,3,3)) + MaxPool(Kernel (3,3,3)) + No Dropout + Yes BatchNorm FC(512) + Dropout(0.25) + L2(0.01) FC(256) + Dropout(0.25) + L2(0.01) FC(128) + Dropout(0.25) + L2(0.01) Dense(5) - Softmax	Train Data = 400 Test Data = 50 Batch Size = 16 Epochs = 50 ReduceLRPatience = 5		Reached 100% Training accuracy and 82% Validation Accuracy. Decision: 1. Now bringing in all data points. 2. No change in architecture

10	(90,90)	All 30	(Convolution and FC layer architecture as above) Conv3D – 32 (Kernel (3,3,3)) + MaxPool(Kernel (3,3,3)) + No Dropout + **Yes** BatchNorm Conv3D – 64 (Kernel(3,3,3)) + MaxPool(Kernel (3,3,3)) + No Dropout + **Yes** BatchNorm FC(512) + Dropout(0.25) + L2(0.01) FC(256) + Dropout(0.25) + L2(0.01) FC(128) + Dropout(0.25) + L2(0.01) Dense(5) - Softmax	Train Data = All Test Data = All Batch Size = 16 Epochs = 50 ReduceLRPatience = 5		With more number of data points, variance has increased. Model is becoming unstable. Decision: 1) Including Dropouts at Conv Layers 2) Trying again with all data points and 50 epochs
11	(90,90)	All 30	(Convolution and FC layer architecture as above) Conv3D – 32 (Kernel (3,3,3)) + MaxPool(Kernel (3,3,3)) + Dropout(0.25) + **Yes** BatchNorm Conv3D – 64 (Kernel(3,3,3)) + MaxPool(Kernel (3,3,3)) + Dropout(0.25) + **Yes** BatchNorm FC(512) + Dropout(0.25) + L2(0.01) FC(256) + Dropout(0.25) + L2(0.01) FC(128) + Dropout(0.25) + L2(0.01) Dense(5) - Softmax	Train Data = All Test Data = All Batch Size = 16 Epochs = 50 ReduceLRPatience = 5		We reached 100% training accuracy and about 75% validation accuracy. **FINAL CNN3D MODEL** **(model-00047-9.74345-0.99095-10.55803-0.75000.h5)**

Final Validation Data Prediction Confusion Matrix (CNN3D Model)

	Predicted_Left	Predicted_Right	Predicted_Stop	Predicted_Down	Predicted_Up
Actual_Left	10	6	1	1	0
Actual_Right	0	23	0	0	0
Actual_Stop	3	2	12	3	2
Actual_Down	0	0	0	19	2
Actual_Up	1	0	2	2	11

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

CNN2D+GRU Modelling technique

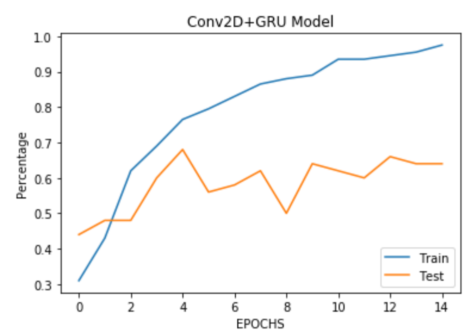
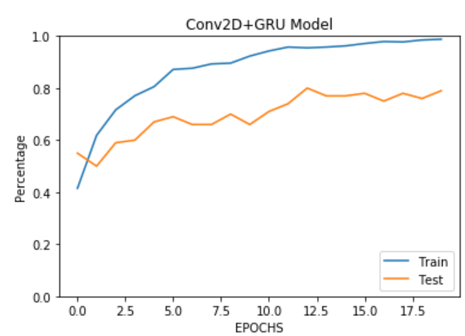
We use a CNN2D pre-learned model (VGG19) model to build features from it. These features are then fed to a GRU network for categorizing them into 1 of the 5 gestures.

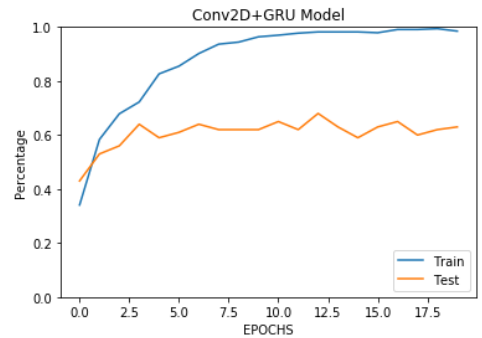
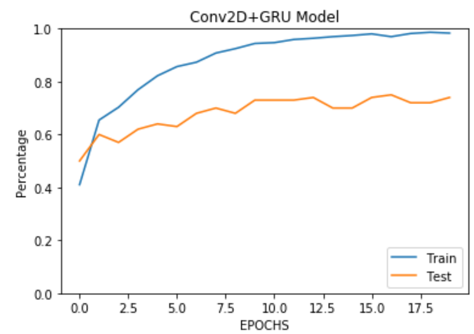
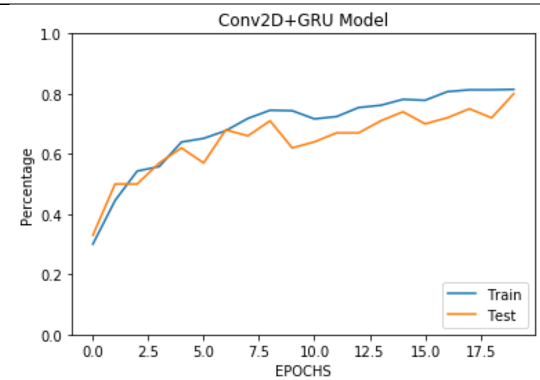
We will use (120,120,3) dimension of image data. Bigger than Conv3D type modelling technique. This is done because we are using 2 models here Conv2D and RNN, so having more features is helpful. Lesser number of points will reduce the number of features.

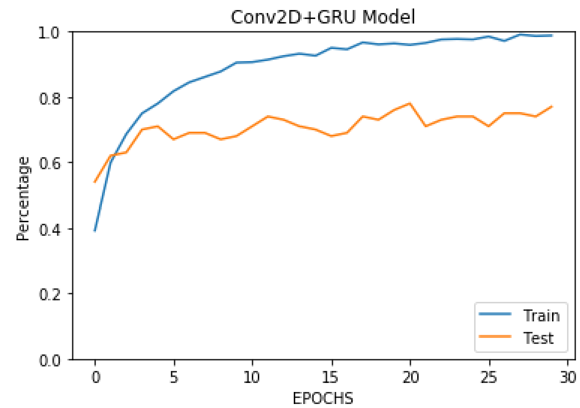
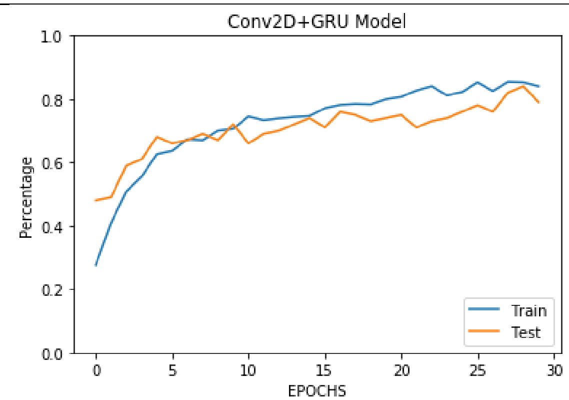
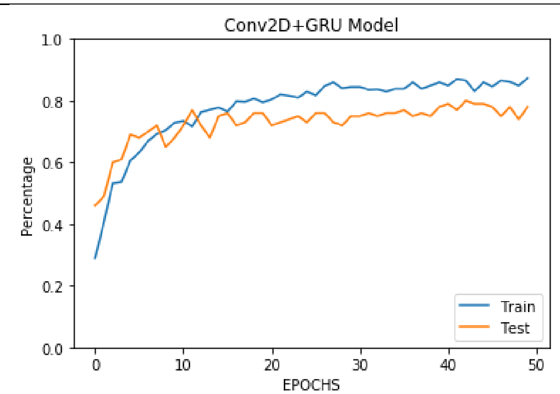
The method to obtain output from VGG19 has been embedded in GENERATOR functions as described above.

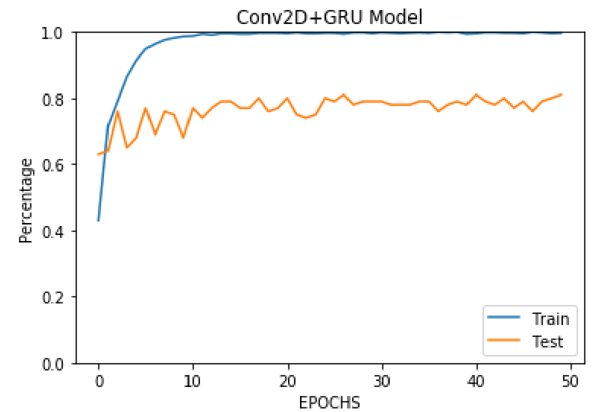
We will describe the different experiments that we did for GRU architecture below.

CNN2D(VGG19) + GRU Modelling Experiment Results

Exp SNo	Frame Size	Framelist	Architecture	Trainsize	Graphs	Remarks
1	(120,120)	All 30	GRU(64) Dense(5) + Softmax Activation	Train Data = 200 Test Data = 50 Batch Size = 16 Epochs = 15 ReduceLRPatience = 5		Training accuracy is reaching 100%, however validation accuracy is low. Increasing training data points. Decision: Increasing the number of training data points to ALL Increasing the number of EPOCHS to 20
2	(120,120)	All 30	GRU(64) Dense(5) + Softmax Activation	Train Data = All Test Data = All Batch Size = 16 Epochs = 20 ReduceLRPatience = 5		Things are getting better. Training accuracy is reaching 100% and validation accuracy is hitting 81%. Lets try and change batch size for improving speed of training (model building) Decision: 1) Increase batch size to 32

3	(120,120)	All 30	GRU(64) Dense(5) + Softmax Activation	Train Data = All Test Data = All Batch Size = 32 Epochs = 20 ReduceLRPatience = 5		Increasing batch size is not helping. Although we reach 100% faster, but overall it is not helpful, as validation accuracy has reduced to just ~ 60%. Decision 1) Keeping batchsize = 16 only. 2) Changing number of frames (reducing the number of frames per video)
4	(120,120)	[0, 1, 2, 3, 4, 5, 6, 9, 12, 15, 18, 21, 24, 25, 26, 27, 28, 29]	GRU(64) Dense(5) + Softmax Activation	Train Data = All Test Data = All Batch Size = 32 Epochs = 20 ReduceLRPatience = 5		Using limited number of frames per video, seem result in lower validation accuracy. (Max valid acc = 74%). Decision 1) Using all frames for training. 2) Adding dropout at GRU layer to reduce variance
5	(120,120)	Range(30)	GRU(64) + Dropout(0.5) Dense(5) + Softmax Activation	Train Data = All Test Data = All Batch Size = 32 Epochs = 20 ReduceLRPatience = 5		Variance has reduced to a large extent. Validation Accuracy is following training accuracy in a good manner. But it is not reading 100%. Lets keep this regularization at GRU and add another GRU layer. Decision: 1) Keep 0.5 Dropout at GRU64 2) Add another GRU 3) Increase epochs to 30

6	(120,120)	Range(30)	GRU(64) + Dropout(0.5) + Return Seq GRU(64) Dense(5) + Softmax Activation	Train Data = All Test Data = All Batch Size = 32 Epochs = 30 ReduceLRPatience = 5		Look a bit better. Adding GRU helped to increase training accuracy, but variance again came up. Decision: 1. Add Dropout(0.25) at 2nd GRU
7	(120,120)	Range(30)	GRU(64) + Dropout(0.5) + Return Seq GRU(64) + Dropout(0.25) Dense(5) + Softmax Activation	Train Data = All Test Data = All Batch Size = 32 Epochs = 30 ReduceLRPatience = 5		Dropout at 2nd GRU made training and validation accuracy become closer. Interesting. Increasing the number of epochs. Decision: 1) Increase epochs to 50
8	(120,120)	range(30)	GRU(64) + Dropout(0.5) + Return Seq GRU(64) + Dropout(0.25) Dense(5) + Softmax Activation	Train Data = All Test Data = All Batch Size = 32 Epochs = 50 ReduceLRPatience = 5		Increasing epochs is not helping. Lets keep things simple. Lets have 2 GRU, with no Dropouts or regularizations Decision 1) Have 2 GRUs but no dropouts

9	(120,120)	Range(30)	GRU(64) + Return Seq GRU(64) Dense(5) + Softmax Activation	Train Data = All Test Data = All Batch Size = 32 Epochs = 50 ReduceLRPatience = 5		Fair model, quite stable now. Training accuracy is 100% and max validation accuracy reached = 81% **FINAL CNN2D + GRU MODEL (model-00027-0.04369-0.99397-0.54823-0.81000.h5)**

Final Validation Data Prediction Confusion Matrix (CNN2D+GRU Model)

	Predicted_Left	Predicted_Right	Predicted_Stop	Predicted_Down	Predicted_Up
Actual_Left	10	6	2	0	0
Actual_Right	5	15	1	0	2
Actual_Stop	0	1	20	1	0
Actual_Down	0	0	0	21	0
Actual_Up	1	0	3	0	12

The problem here seems to be mainly prediction ***Swipe Left.***

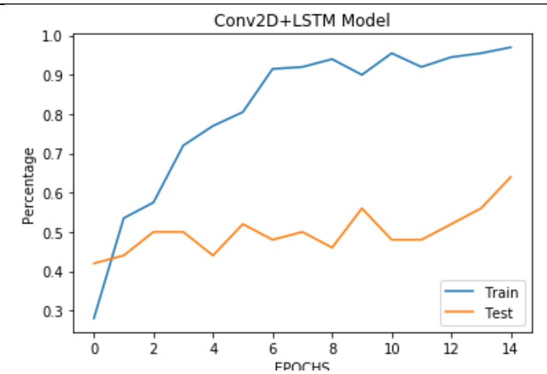
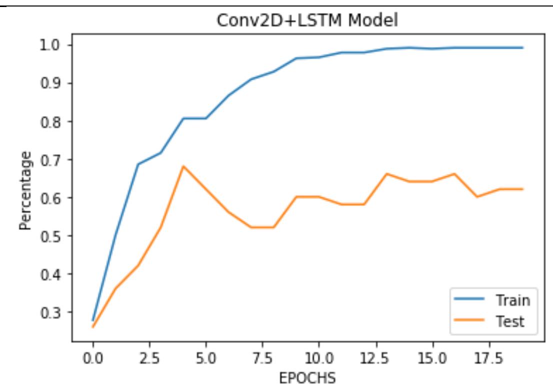
CNN2D + LSTM Network

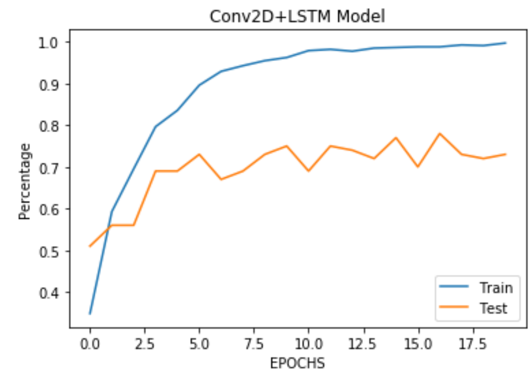
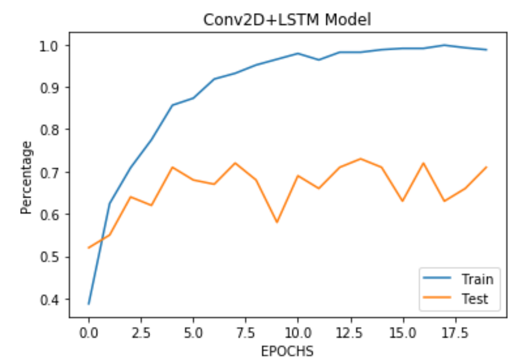
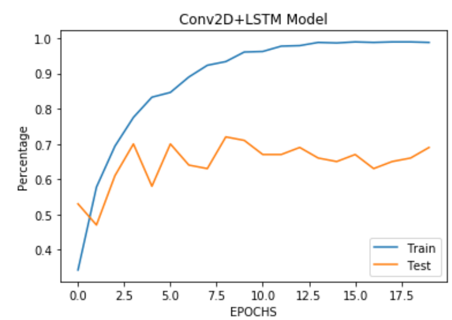
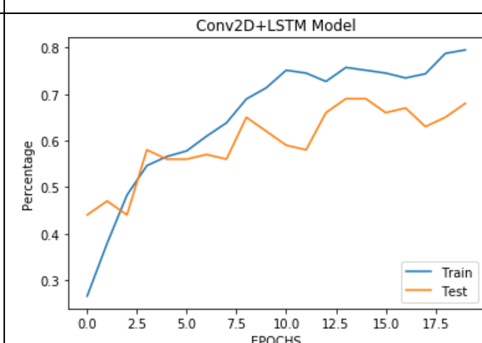
We use a CNN2D pre-learned model (VGG19) model to build features from it. These features are then fed to a LSTM network for categorizing them into 1 of the 5 gestures.

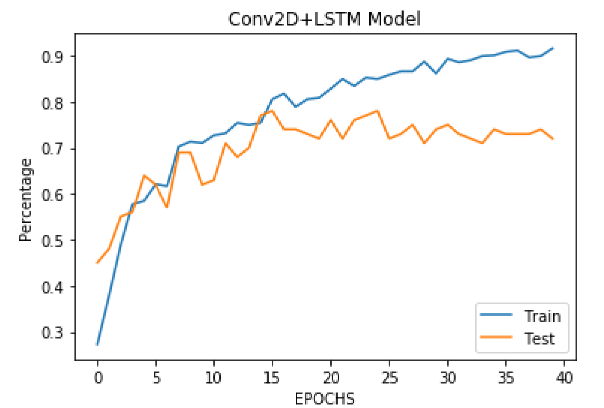
We will use (120,120,3) dimension of image data. Bigger than Conv3D type modelling technique. This is done because we are using 2 models here Conv2D and RNN, so having more features is helpful. Lesser number of points will reduce the number of features.

The method to obtain output from VGG19 has been embedded in GENERATOR functions as described above.

We will describe the different experiments that we did for LSTM architecture below.

CNN2D(VGG19) + LSTM Modelling Experiment Results						
Exp SNo	**Frame Size**	**Framelist**	**Architecture**	**Trainsize**	**Graphs**	**Remarks**
1	(120,120)	All 30	LSTM(64) Dense(5) + Softmax Activation	Train Data = 200 Test Data = 50 Batch Size = 16 Epochs = 15 ReduceLRPatience = 5		Training accuracy is reaching 100%, however validation accuracy is low. Increasing training data points. Decision: Increasing the number of training data points to 400 Increasing the number of EPOCHS to 20
2	(120,120)	All 30	LSTM(64) Dense(5) + Softmax Activation	Train Data = 400 Test Data = 50 Batch Size = 16 Epochs = 20 ReduceLRPatience = 5		Training accuracy is stabilizing. However, variance seems to be high. Lets try to increase training data points further. Decision: 1) Increase training data points to ALL.

3	(120,120)	All 30	LSTM(64) Dense(5) + Softmax Activation	Train Data = All Test Data = All Batch Size = 16 Epochs = 20 ReduceLRPatience = 5		Validation accuracy has stagnated and also training accuracy is not hitting complete 100% Decision: 1. Increasing LSTM layer
4	(120,120)	All 30	LSTM(64) + return_sequences LSTM(64) Dense(5) + Softmax Activation	Train Data = All Test Data = All Batch Size = 16 Epochs = 20 ReduceLRPatience = 5		Increasing LSTM layer is not helping. It is increasing the number of training data parameters (as this is LSTM type network) and validation accuracy is going for a toss. Decision: 1) Have only 1 LSTM layer 2) Bring L2 Regularization at Dense Layer
5	(120,120)	All 30	LSTM(64) Dense(5) + Softmax Activation + L2(0.01)	Train Data = All Test Data = All Batch Size = 16 Epochs = 20 ReduceLRPatience = 5		Not helping. Removing L2 Reg and adding dropouts at LSTM layer Decision: 1. Remove L2 Reg from Dense Layer 2. Add Dropout at LSTM layer
6	(120,120)	All 30	LSTM(64) + Dropout(0.5) Dense(5) + Softmax Activation	Train Data = All Test Data = All Batch Size = 16 Epochs = 20 ReduceLRPatience = 5		Things seem to be improving here. Increase EPOCH to 40 Decision: Increasing epochs = 40

7	(120,120)	All 30	LSTM(64) + Dropout(0.5) Dense(5) + Softmax Activation	Train Data = All Test Data = All Batch Size = 16 Epochs = 40 ReduceLRPatience = 5		Fair model. We reached 100% training accuracy and 77% validation accuracy. **FINAL CNN2D+LSTM MODEL** **(model-00040-0.49856-0.84615-0.67494-0.78000.h5)**

Final Validation Data Prediction Confusion Matrix (Conv2D + LSTM Model)

	Predicted_Left	Predicted_Right	Predicted_Stop	Predicted_Down	Predicted_Up
Actual_Left	11	5	1	0	1
Actual_Right	5	17	1	0	0
Actual_Stop	0	0	19	2	1
Actual_Down	0	0	0	20	1
Actual_Up	0	0	4	0	12

Again there are misses happening for ***Swipe Left*** and ***Swipe Right***.

CNN2D + SimpleRNN Network

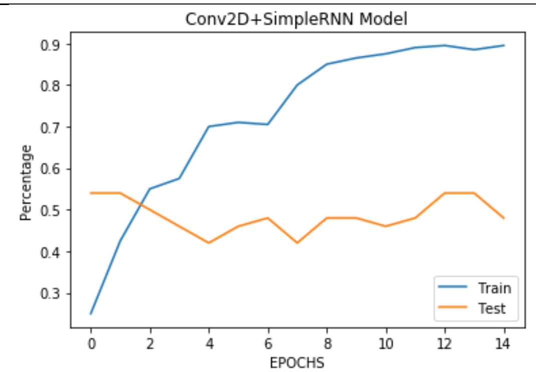
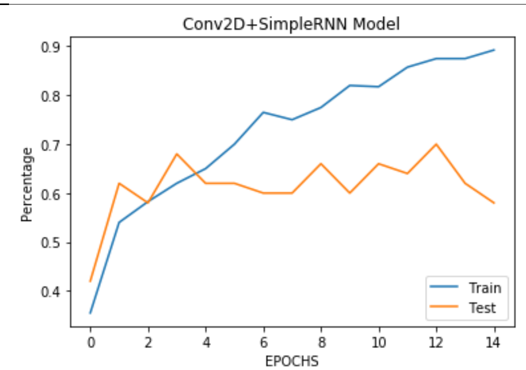
We use a CNN2D pre-learned model (VGG19) model to build features from it. These features are then fed to a SimpleRNN network for categorizing them into 1 of the 5 gestures.

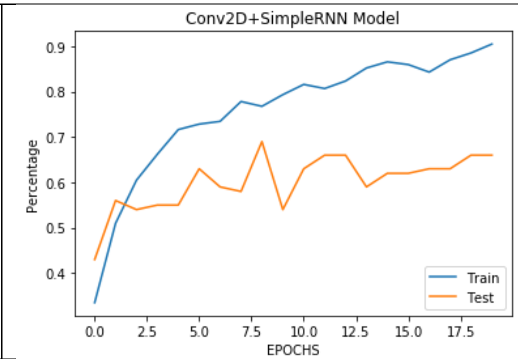
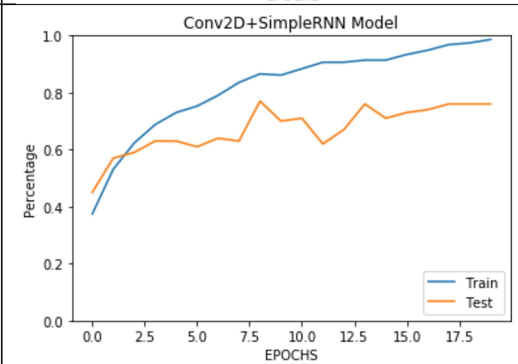
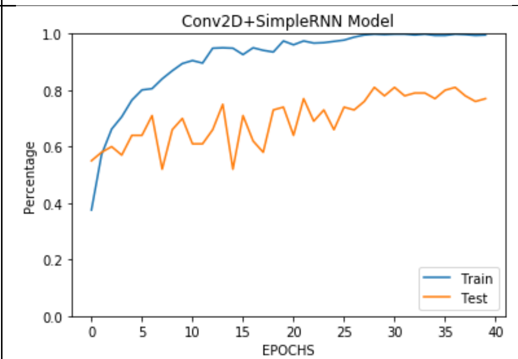
We will use (120,120,3) dimension of image data. Bigger than Conv3D type modelling technique. This is done because we are using 2 models here Conv2D and RNN, so having more features is helpful. Lesser number of points will reduce the number of features.

The method to obtain output from VGG19 has been embedded in GENERATOR functions as described above.

We will describe the different experiments that we did for SimpleRNN architecture below.

CNN2D(VGG19) + SimpleRNN Modelling Experiment Results

Exp SNo	Frame Size	Framelist	Architecture	Trainsize	Graphs	Remarks
1	(120,120)	All 30	SimpleRNN(64) Dense(5) + Softmax Activation	Train Data = 200 Test Data = 50 Batch Size = 16 Epochs = 15 ReduceLRPatience = 5		Training accuracy is reaching 100%, however validation accuracy is low. Increasing training data points. Decision: Increasing the number of training data points to 400
2	(120,120)	All 30	SimpleRNN(64) Dense(5) + Softmax Activation	Train Data = 400 Test Data = 50 Batch Size = 16 Epochs = 15 ReduceLRPatience = 5		Things are improving. Increasing training data points to ALL and EPOCHS to 20 Decision: 1) Increase all training and validation data points 2) Increase epochs to 20

3	(120,120)	All 30	SimpleRNN(64) Dense(5) + Softmax Activation	Train Data = All Test Data = All Batch Size = 16 Epochs = 20 ReduceLRPatience = 5		Not much help. Adding another SimpleRNN(64) layer Decision: Add another SimpleRNN(64) layer
4	(120,120)	All 30	SimpleRNN(64) + Return_sequences SimpleRNN(64) Dense(5) + Softmax Activation	Train Data = All Test Data = All Batch Size = 16 Epochs = 20 ReduceLRPatience = 5		Fair model, however variance seems to be pretty high. Lets try to increase EPOCHS
5	(120,120)	All 30	SimpleRNN(64) + Return_sequences SimpleRNN(64) Dense(5) + Softmax Activation	Train Data = All Test Data = All Batch Size = 16 Epochs = 40 ReduceLRPatience = 5		Good Model, we reached 100% training accuracy and 81% validation accuracy. **FINAL CNN2D+SimpleRNN Model** **model-00029-0.03407-0.99849-0.73980-0.81000.h5**

Final Validation Data Prediction Confusion Matrix (CNN2D + SimpleRNN Model)

	Predicted_Left	Predicted_Right	Predicted_Stop	Predicted_Down	Predicted_Up
Actual_Left	10	6	2	0	0
Actual_Right	5	17	0	1	0
Actual_Stop	1	0	20	0	1
Actual_Down	0	0	0	21	0
Actual_Up	0	0	5	0	11

Miss happening at ***Swipe Left & Swipe Right.***

CONCLUSION

Modelling technique	Training Accuracy	Validation Accuracy
Conv3D	100%	75%
Conv2D+GRU	100%	81%
Conv2D+LSTM	100%	78%
Conv2D+SimpleRNN	100%	81%

Important Points:

1. It is important to note that Conv3D and Conv2D+LSTM have very high number of parameters for learning. And we can see that there validation data accuracy is lesser than others. Conv2D + GRU seems to be doing best and also Conv2D + SimpleRNN.
2. 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.).
3. 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.
4. 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

IMPORTANT NOTE FROM TEAM:

Our team's modelling philosophy is based on following fundamentals

1) Model should attain ~ 100% training accuracy as primary objective.
2) After it has attained (Point 1) then we look at stabilizing Validation accuracy.
3) This is derived from the idea that our model should be able to do good (almost perfect) prediction on training data, which is fed to it for learning. Reducing variance is the 2nd objective.
4) Therefore, at multiple places in above explanation, it can be seen that we have tuned parameters even when variance between training and validation was very less. This was done because, even though variance was less, training accuracy was not reaching 100%.