Hand Gesture Recognition using Deep Learning

1. Problem Statement:

This project is about developing a model which can recognize the five User's Hand Gesture that will help user to control the Video player in the Television without Remote. The Gesture is continuously monitored by the webcam mounted on the TV:

- Thumbs up: Increase the Volume
- Thumbs Down: Decrease the Volume
- Left Swipe: Jump Backwards 10 sec
- Right Swipe: Jump Forward 10 sec
- Stop: Pause the Movie

2. Training Data and Validation Data

The Training and Validation data consist of 663 and 100 videos respectively. Each videos is further divided into 30 frames each capturing the hand movements. The video frames images consists of two size (120, 160, 3) and (360, 360, 3). Three representing the RGB channels and rest two dimensions denoting the number of rows and columns.

3. Data Generators

In most of the Deep Learning projects, the data is feed into the model in batches. This is done using the concept of Data Generators to set up the data ingestion pipeline. Creating Data Generators is probably the most important part of building a training pipeline. Although libraries like Keras provide built-in generator functionalities, but they are often limited in scope and you have to write your own generators from scratch. For example, in this problem, you need to feed Batches of videos, not images.

The data generator yield batch of data at each iteration of shape (nFrames, nRows, nColumns, Channels). We have two image size, 120x160 and 360x360. While processing the image in the generators we cropped the image having 120x160 into 120x120 and then passed it to resize function to make it 80x80 image. The image with 360x360 shape is directly passed to resize function.

4. Model Architecture:

For Analysis videos using Neural Network, two types of architecture used commonly:

- 3D Convolution Network or Convo3D
  3D convolutions are the natural extension to the 2D convolutions. Just like 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).

- Convolutions + RNN
  The Convo 2D 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.

Note: For all the models we have used Image Size of (80, 80, 3) and 15 frames for each video.

5. Convolution Neural Network Modelling:

Model 1: (CONV3D)

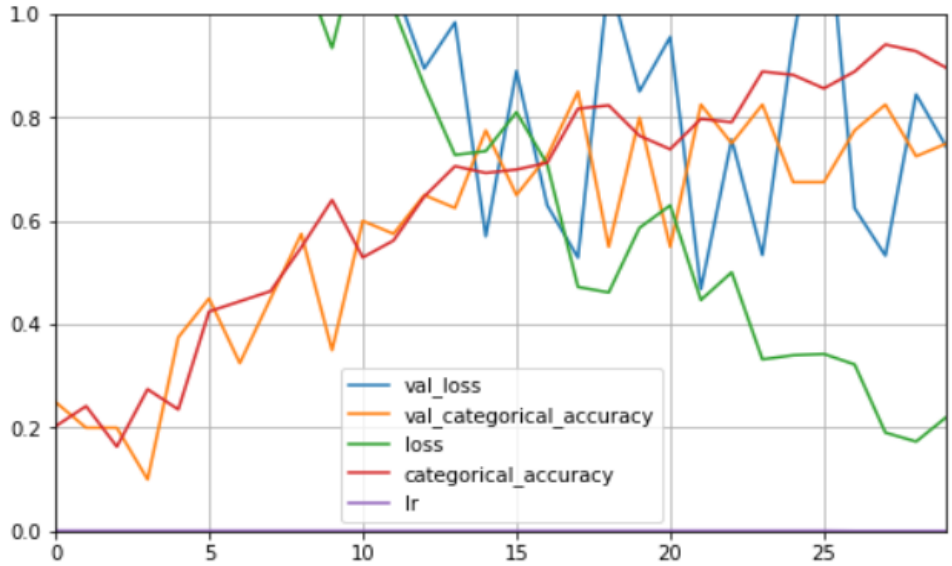
We started with basic model having MaxPooling in each hidden layer. “Relu” Activation was used. For all the models we used the image size of (80,80,3) and we selected 15 alternate frames out of 30 frames available for each video.

```
#Define model
model = Sequential()

#First convolution layer
model.add(Conv3D(16, kernel_size=(3, 3,3),padding='same', input_shape=input_shape))
model.add(Activation('relu'))

#First convolution layer
model.add(Conv3D(32, kernel_size=(3, 3,3),padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))

# second conv layer
model.add(Conv3D(64, kernel_size=(3, 3, 3),padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))

# third conv layer
model.add(Conv3D(128, kernel_size=(3, 3, 3),padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))

# flatten and put a fully connected layer
model.add(Flatten())
model.add(Dense(128, activation='relu')) # fully connected
#model.add(Dropout(0.5))

# softmax layer
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

We kept the Number of epoch to 30 and batch size of 16.

Training Accuracy: 89%
Validation Accuracy: 74%.

The below and all the following Graph is Number of epoch vs Training/Validation accuracy and loss and learning rate.



The Graph shows that the model has started overfitting somewhere around 22nd epoch as the validation accuracy started decreasing and Validation loss started increasing.

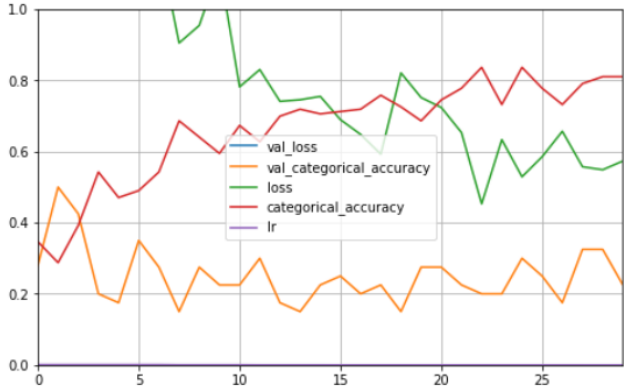
Model 2:(CONV3D)

To prevent the overfitting we added batch normalization and Dropouts.

```
#Define model
model = Sequential()

model.add(Conv3D(16, kernel_size=(3, 3,3),padding='same', input_shape=input_shape))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Conv3D(32, kernel_size=(3, 3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
model.add(Dropout(0.2))

model.add(Conv3D(64, kernel_size=(3, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
model.add(Dropout(0.2))

model.add(Conv3D(128, kernel_size=(3, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
model.add(Dropout(0.2))

# flatten and put a fully connected layer
model.add(Flatten())
model.add(Dense(128, activation='relu')) # fully connected
model.add(BatchNormalization())
model.add(Dropout(0.5))

# softmax layer
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

The model is overfitting still.

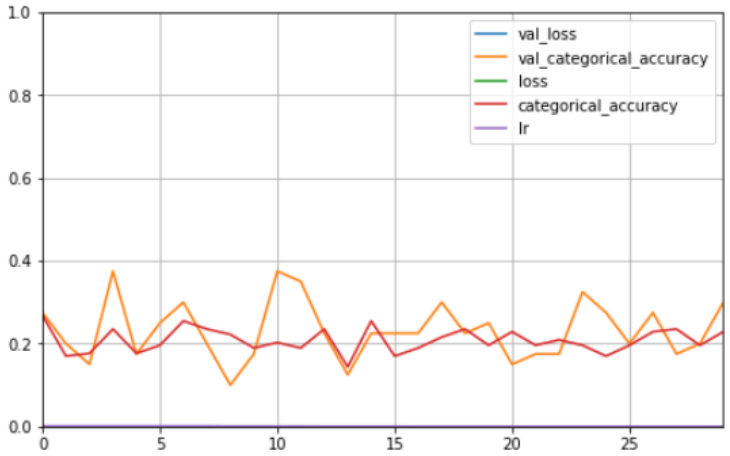
Training accuracy - 81% Validation accuracy - 32%



Model 3: (CONV3D)
We removed the dropouts from the FC layer.

```
#Define model
model = Sequential()

model.add(Conv3D(16, kernel_size=(3, 3,3),padding='same', input_shape=input_shape))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Conv3D(32, kernel_size=(3, 3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
model.add(Dropout(0.2))

model.add(Conv3D(64, kernel_size=(3, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
model.add(Dropout(0.2))

model.add(Conv3D(128, kernel_size=(3, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
model.add(Dropout(0.2))

# flatten and put a fully connected layer
model.add(Flatten())
model.add(Dense(128, activation='relu')) # fully connected


# softmax layer
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

Training Accuracy: 22%
Validation Accuraccy: 30%

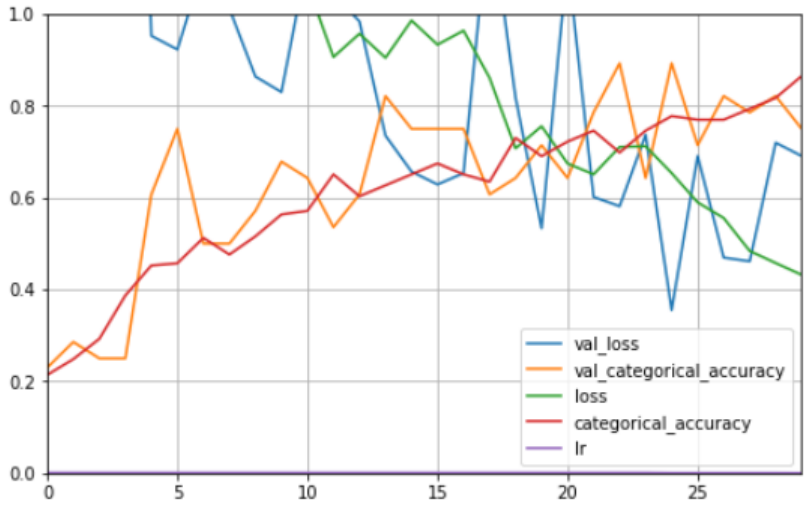


Model4: (CONV3D)

We added dropout in the FC layer and removed it from the Convolution layers.

```
#Define model
model = Sequential()

#First convolution layer
model.add(Conv3D(16, kernel_size=(3, 3,3),padding='same', input_shape=input_shape))
model.add(BatchNormalization())
model.add(Activation('relu'))
#model.add(Dropout(0.2))

#First convolution layer
model.add(Conv3D(32, kernel_size=(3, 3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
#model.add(Dropout(0.2))

# second conv layer
model.add(Conv3D(64, kernel_size=(3, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
#model.add(Dropout(0.2))

# third conv layer
model.add(Conv3D(128, kernel_size=(3, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
#model.add(Dropout(0.2))

# flatten and put a fully connected layer
model.add(Flatten())
model.add(Dense(128, activation='relu')) # fully connected
model.add(Dropout(0.5))

# softmax layer
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

Training Accuracy 86%
Validation Accuracy 75%

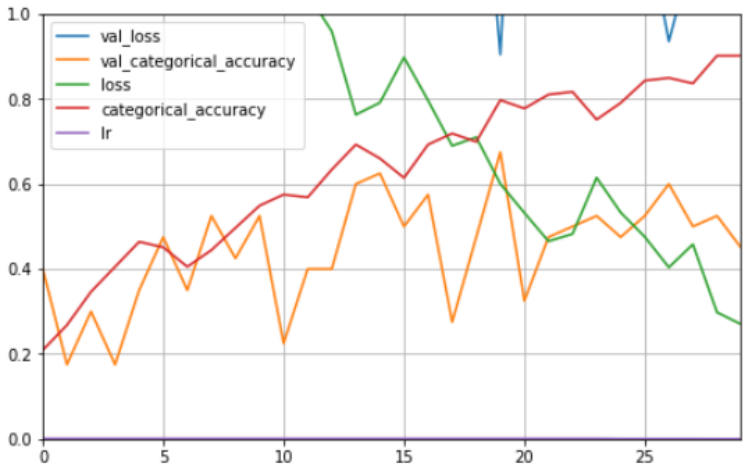


Model 5: (CONV3D)

We decreases the number of filters to 64 from 128 in the 4th convolution layer and added one FC layer with 64 neurons because the loss has increased in the previous model after epoch 25, could be sign of overfitting.

```
#Define model
model = Sequential()

model.add(Conv3D(16, kernel_size=(3, 3,3),padding='same', input_shape=input_shape))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Conv3D(32, kernel_size=(3, 3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
model.add(Dropout(0.2))

model.add(Conv3D(64, kernel_size=(3, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
model.add(Dropout(0.2))

model.add(Conv3D(64, kernel_size=(3, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
model.add(Dropout(0.2))

# flatten and put a fully connected layer
model.add(Flatten())
model.add(Dense(128, activation='relu')) # fully connected
model.add(Dense(64, activation='relu'))

# softmax layer
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

Training Accuracy: 90%
Validation Accuracy: 52%

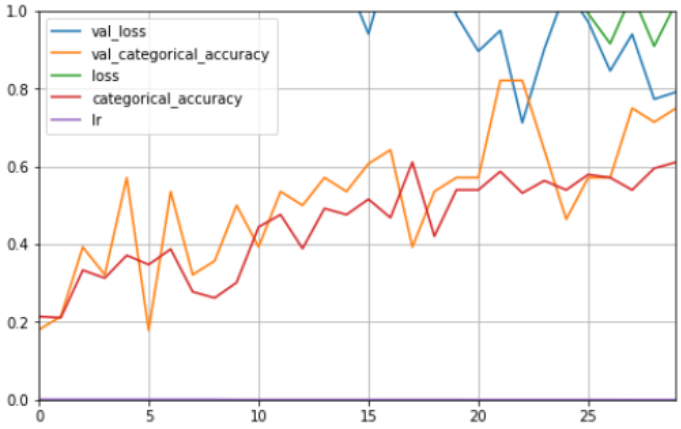


The model is still overfitting but it has started learning again.

Model 6: (CONV3D)

In this model we removed the BatchNormalization (kept only in the last convolution layer) and dropouts from the convolution layer and added dropouts in FC layer.

```
#Define model
model = Sequential()

#First convolution layer
model.add(Conv3D(16, kernel_size=(3, 3,3),padding='same', input_shape=input_shape))
model.add(Activation('relu'))
#model.add(MaxPooling3D(pool_size=(2, 2, 2)))

model.add(Conv3D(32, kernel_size=(3, 3, 3),padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))

# second conv layer
model.add(Conv3D(64, kernel_size=(3, 3, 3),padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))


# third conv layer
model.add(Conv3D(64, kernel_size=(3, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))


# flatten and put a fully connected layer
model.add(Flatten())
model.add(Dense(128, activation='relu')) # fully connected
model.add(Dropout(0.5))

model.add(Dense(64, activation='relu')) # fully connected
model.add(Dropout(0.5))

# softmax layer
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

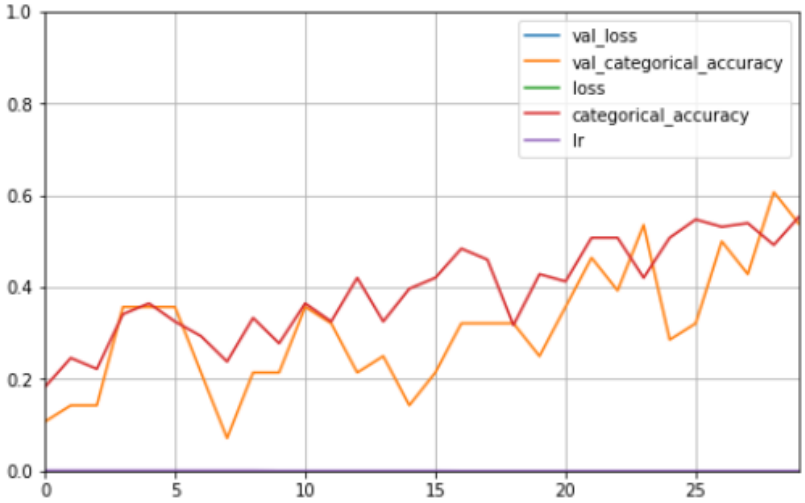


Training accuracy 59% Validation accuracy 71%
From the graph it shows that the model is under fitting as the validation accuracy is more than the training accuracy for most of the epoch.

Model 7: (CONV3D)
We replaced the two Fully Connected layer with one FC with 256 neurons. To avoid overfitting.

```
#Define model
model = Sequential()

#First convolution layer
model.add(Conv3D(16, kernel_size=(3, 3,3),padding='same', input_shape=input_shape))
model.add(Activation('relu'))

model.add(Conv3D(32, kernel_size=(3, 3, 3),padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))

# second conv layer
model.add(Conv3D(64, kernel_size=(3, 3, 3),padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))


# third conv layer
model.add(Conv3D(64, kernel_size=(3, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))


# flatten and put a fully connected layer
model.add(Flatten())
model.add(Dense(256, activation='relu')) # fully connected
model.add(Dropout(0.5))

# model.add(Dense(64, activation='relu')) # fully connected
# model.add(Dropout(0.5))

# softmax layer
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

The Under fitting is been avoided and the validation accuracy is less than the training accuracy. But both are low.
Training accuracy 55 %
Validation Accuracy 53%

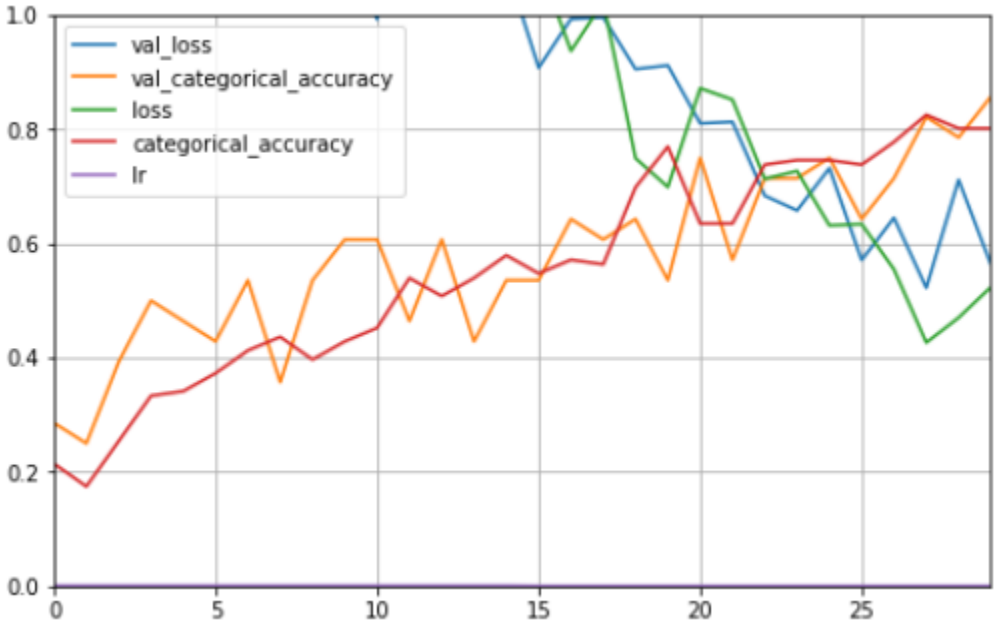


Model 8: (CONV3D)

We removed the 4th Convolution layer with 64 neurons and having batch normalization and decreased the depth of the kernel to two in the 2nd and to One in the 3rd Convolution layer. We have decrease the depth of the kernel as we go to next convolution layer they become more defining towards the differentiating between images, initial layers are for extracting edges and textures. So we thought, decreasing the depth will extract more features in later layers.

```
#Define model
model = Sequential()

#First convolution layer
model.add(Conv3D(16, kernel_size=(3, 3,3),padding='same', input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))

model.add(Conv3D(32, kernel_size=(2, 3, 3),padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))

# second conv layer
model.add(Conv3D(64, kernel_size=(1, 3, 3),padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))


# flatten and put a fully connected layer
model.add(Flatten())
model.add(Dense(256, activation='relu')) # fully connected
model.add(Dropout(0.5))



# softmax layer
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

Training Accuracy 82 %
Validation Accuracy 82%

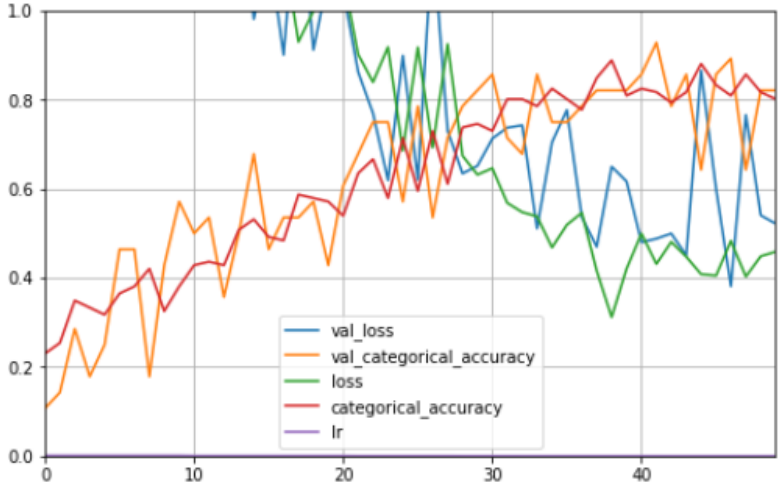


Model 9: (CONV3D)

We added one Batch Normalization in the last Convolution layer to improve the learning and also we changed the kernel size of the first layer to (4, 4, 4) because the first layers is for high level feature extraction so a bigger filter should do the job. Also we increased the number of Epochs to 50.

```
#Define model
model = Sequential()

model.add(Conv3D(16, kernel_size=(4, 4,4),padding='same', input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))

model.add(Conv3D(32, kernel_size=(3, 3, 3),padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))

model.add(Conv3D(64, kernel_size=(2, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
model.add(Dropout(0.20))


# flatten and put a fully connected layer
model.add(Flatten())
model.add(Dense(256, activation='relu')) # fully connected
model.add(Dropout(0.5))

# softmax layer
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

Training Accuracy 81%
Validation Accuracy 82%

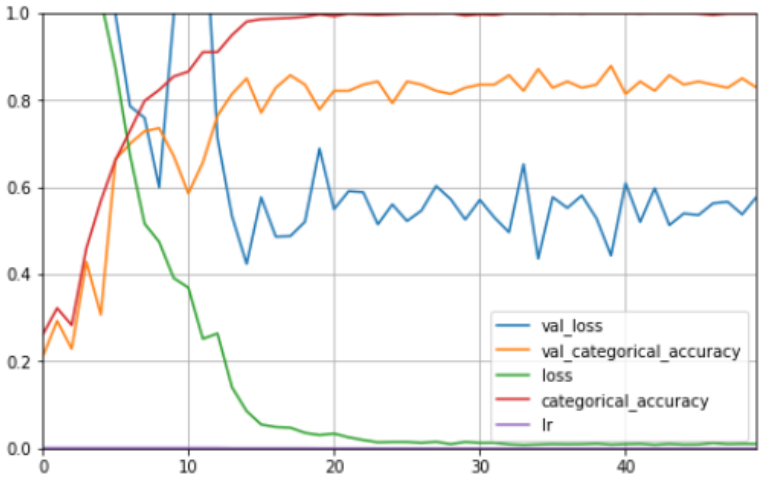


Model 10: (CONV3D)

In previous model, the validation accuracy is more than the training accuracy. So to improve the learning we will add one batch normalization in the 2nd Convolution layer and dropouts and we will add one FC layer with 128 neurons:

```
#Define model
model = Sequential()


model.add(Conv3D(16, kernel_size=(4, 4,4),padding='same', input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
model.add(Dropout(0.2))

model.add(Conv3D(32, kernel_size=(3, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
model.add(Dropout(0.2))


model.add(Conv3D(64, kernel_size=(2, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
model.add(Dropout(0.25))


# flatten and put a fully connected layer
model.add(Flatten())
model.add(Dense(256, activation='relu')) # fully connected


model.add(Dense(128, activation='relu')) # fully connected


# softmax layer
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

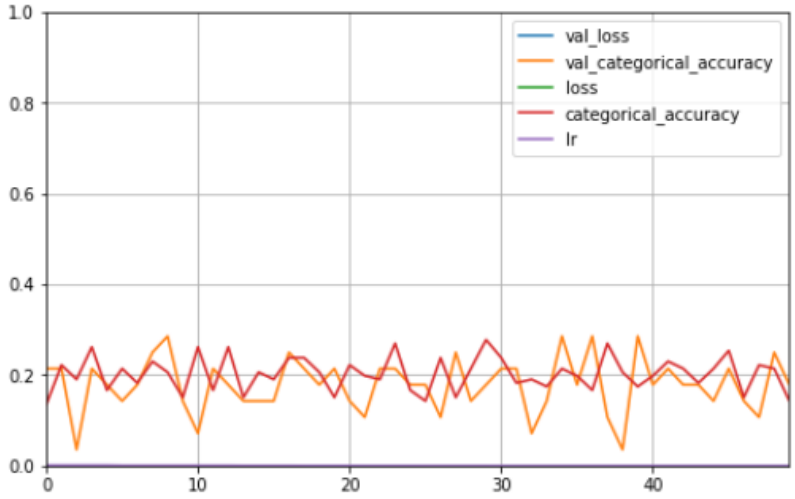
Training Accuracy : 99.8 %
Validation Accuracy: 82%
The model has overfit.



Model 11: (CONV3D)

We changed the kernel size back to (3, 3, 3).

```
#Define model
model = Sequential()


model.add(Conv3D(16, kernel_size=(3, 3,3),padding='same', input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
model.add(Dropout(0.2))

model.add(Conv3D(32, kernel_size=(3, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
model.add(Dropout(0.2))

model.add(Conv3D(64, kernel_size=(3, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
model.add(Dropout(0.2))


# flatten and put a fully connected layer
model.add(Flatten())
model.add(Dense(256, activation='relu')) # fully connected

model.add(Dense(128, activation='relu')) # fully connected


# softmax layer
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```



Training Accuracy: 18%
Validation Accuracy: 17%

Model 12: (CONV3D)

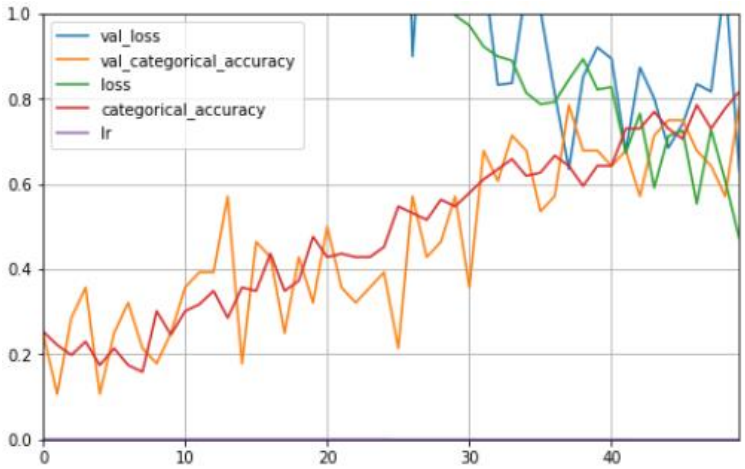
So far we have tried different model and experimented with changing the number of epochs, Batch size, kernel size, adding dropouts and Batch Normalization and we have not reached to a decent model with good accuracy and lower loss

In next we will increase the depth of the model by adding two layers of 16 filter, two layers of 32 and two layers of 64.

In the last set of convolution layer we will keep the Kernel size as (2, 3, 3) as the last layers are more for extracting the and dropouts only in FC layers.

```
#Define model
model = Sequential()


model.add(Conv3D(16, kernel_size=(3, 3,3),padding='same', input_shape=input_shape))
model.add(Activation('relu'))
model.add(Conv3D(16, kernel_size=(3, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))


model.add(Conv3D(32, kernel_size=(3, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv3D(32, kernel_size=(3, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))


model.add(Conv3D(64, kernel_size=(2, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv3D(64, kernel_size=(2, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))


# flatten and put a fully connected layer
model.add(Flatten())
model.add(Dense(256, activation='relu')) # fully connected
model.add(Dropout(0.50))

model.add(Dense(128, activation='relu')) # fully connected
model.add(Dropout(0.50))

# softmax layer
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

Training accuracy: 81%
Validation Accuracy: 78%



Model 13: (CONV3D) This Model is our Final Model.

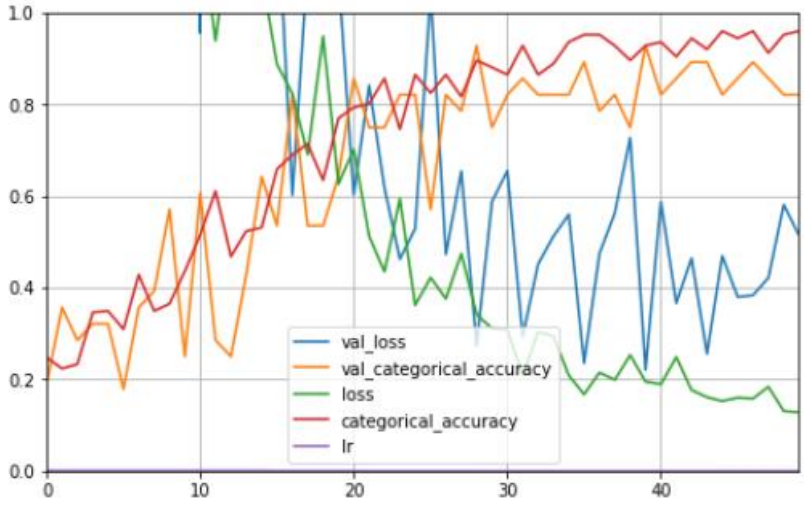
We removed the dropouts from the FC layer and added in the Convolution layer:

```
#Define model
model = Sequential()

model.add(Conv3D(16, kernel_size=(3, 3,3),padding='same', input_shape=input_shape))
model.add(Activation('relu'))
model.add(Conv3D(16, kernel_size=(3, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
model.add(Dropout(0.2))

model.add(Conv3D(32, kernel_size=(3, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv3D(32, kernel_size=(3, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
model.add(Dropout(0.2))


model.add(Conv3D(64, kernel_size=(2, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv3D(64, kernel_size=(2, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
model.add(Dropout(0.25))


# flatten and put a fully connected layer
model.add(Flatten())
model.add(Dense(256, activation='relu')) # fully connected
#model.add(Dropout(0.50))

model.add(Dense(128, activation='relu')) # fully connected
#model.add(Dropout(0.50))

# softmax layer
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

Training Accuracy: 96.82%
Validation Accuracy: 92.86%

The training accuracy is continuously increasing and the validation accuracy is also following the same trend. The Training loss is decreasing and so is the validation loss (though it has fluctuation for alternate epoch, but on an average it is decreasing)

This model we have given for final submission.



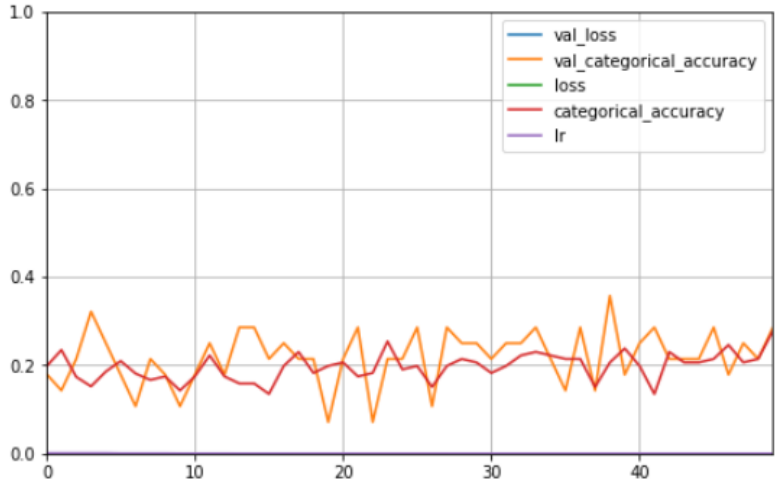
Model 14: CNN + LSTM

For below models, We have used Frame size of (80,80,3) and 15 frames per video.
Batch size of 16 and Number of epochs 50.

```
model = Sequential()

model.add(TimeDistributed(Conv2D(16, (3, 3),
    activation='relu', padding='same'), input_shape=input_shape))
model.add(TimeDistributed(Conv2D(16, (3,3),
    kernel_initializer="he_normal", activation='relu')))
model.add(TimeDistributed(MaxPooling2D((2, 2))))

model.add(TimeDistributed(Conv2D(32, (3,3),
    padding='same', activation='relu')))
model.add(TimeDistributed(Conv2D(32, (3,3),
    padding='same', activation='relu')))
model.add(TimeDistributed(MaxPooling2D((2, 2))))

model.add(TimeDistributed(Conv2D(64, (3,3),
    padding='same', activation='relu')))
model.add(TimeDistributed(Conv2D(64, (3,3),
    padding='same', activation='relu')))
model.add(TimeDistributed(MaxPooling2D((2, 2))))

model.add(TimeDistributed(Conv2D(128, (3,3),
    padding='same', activation='relu')))
model.add(TimeDistributed(Conv2D(128, (3,3),
    padding='same', activation='relu')))
model.add(TimeDistributed(MaxPooling2D((2, 2))))

model.add(TimeDistributed(Flatten()))

model.add(Dropout(0.5))
model.add(LSTM(256, return_sequences=False, dropout=0.5))
model.add(Dense(nb_classes, activation='softmax'))
```

Training accuracy: 27.8%

Validation Accuracy: 28.8%



May be the model is too complex, there for the model is not learning due to lots of trainable parameters. In the next model we will decrease the complexity.

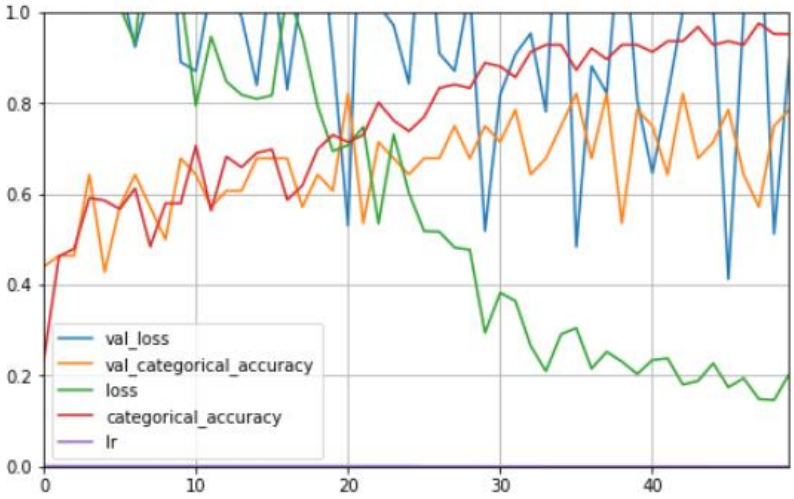
Model 15: (CNN + LSTM)

We removed the Convolution layer with 128 filters and also reduced the LSTM filters from 256 to 128.

```
model = Sequential()

model.add(TimeDistributed(Conv2D(16, (3, 3),
    activation='relu', padding='same'), input_shape=input_shape))
model.add(TimeDistributed(Conv2D(16, (3,3),
    kernel_initializer="he_normal", activation='relu')))
model.add(TimeDistributed(MaxPooling2D((2, 2))))

model.add(TimeDistributed(Conv2D(32, (3,3),
    padding='same', activation='relu')))
model.add(TimeDistributed(Conv2D(32, (3,3),
    padding='same', activation='relu')))
model.add(TimeDistributed(MaxPooling2D((2, 2))))

model.add(TimeDistributed(Conv2D(64, (3,3),
    padding='same', activation='relu')))
model.add(TimeDistributed(Conv2D(64, (3,3),
    padding='same', activation='relu')))
model.add(TimeDistributed(MaxPooling2D((2, 2))))

model.add(TimeDistributed(Flatten()))

model.add(Dropout(0.5))
model.add(LSTM(128, return_sequences=False, dropout=0.5))

model.add(Dense(nb_classes, activation='softmax'))
```

Training accuracy: 89%
Validation accuracy: 82%

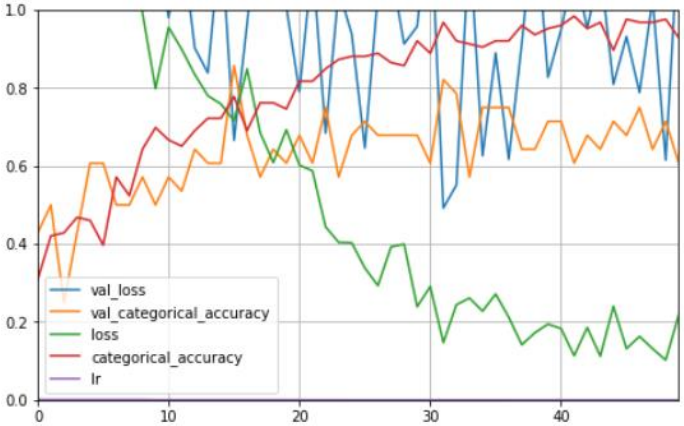
From the graph we can see that the model has started overfitting after 35 epochs



Model 16: CNN (VGG16) + LSTM

```
base_model = VGG16(include_top=False, weights='imagenet', input_shape=(img_rows,img_cols,img_channels))
x = base_model.output
x = Flatten()(x)

features = Dense(64, activation='relu')(x)
conv_model = Model(inputs=base_model.input, outputs=features)

for layer in base_model.layers:
    layer.trainable = False

model = Sequential()
model.add(TimeDistributed(conv_model, input_shape=(img_frms,img_rows,img_cols,img_channels)))
model.add(GRU(32, return_sequences=True))
model.add(GRU(16))
model.add(Dropout(0.5))
model.add(Dense(8, activation='relu'))
model.add(Dense(5, activation='softmax'))
```

Training Accuracy: 96%
Validation Accuracy: 75%



From the graph, we can see that the model has overfit.

The Final Model

Model 12: (CONV3D)

We have selected the Model 12 (epoch 47th) as our final model as it gave the best accuracy for Validation and Training together and also the model is not overfitting as the difference between training and validation accuracy is just 4%.

```
#Define model
model = Sequential()

model.add(Conv3D(16, kernel_size=(3, 3,3),padding='same', input_shape=input_shape))
model.add(Activation('relu'))
model.add(Conv3D(16, kernel_size=(3, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
model.add(Dropout(0.2))

model.add(Conv3D(32, kernel_size=(3, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv3D(32, kernel_size=(3, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
model.add(Dropout(0.2))


model.add(Conv3D(64, kernel_size=(2, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv3D(64, kernel_size=(2, 3, 3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling3D(pool_size=(2, 2, 2)))
model.add(Dropout(0.25))


# flatten and put a fully connected layer
model.add(Flatten())
model.add(Dense(256, activation='relu')) # fully connected
#model.add(Dropout(0.50))

model.add(Dense(128, activation='relu')) # fully connected
#model.add(Dropout(0.50))

# softmax layer
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

Training Accuracy: 96.82%
Validation Accuracy: 92.86%

The training accuracy is continuously increasing and the validation accuracy is also following the same trend. The Training loss is decreasing and so is the validation loss (though it has fluctuation for alternate epoch, but on an average it is decreasing)

This model we have given for final submission.