Hand gesture recognition using neural networks

Problem Statement :

Imagine you are working as a data scientist at a home electronics company which manufactures state of the art smart televisions. You want to develop a cool feature in the smart-TV that can recognise five different gestures performed by the user which will help users control the TV without using a remote.

The gestures are continuously monitored by the webcam mounted on the TV. Each gesture corresponds to a specific command:

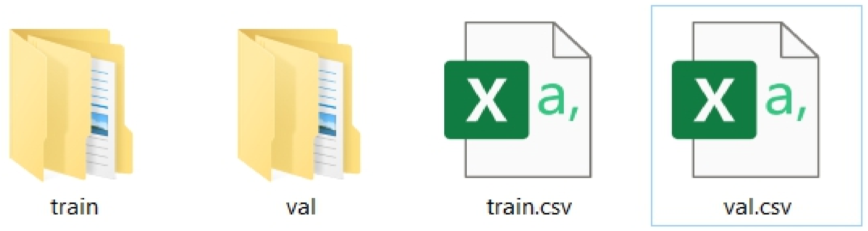
- Thumbs up: Increase the volume
- Thumbs down: Decrease the volume
- Left swipe: 'Jump' backwards 10 seconds
- Right swipe: 'Jump' forward 10 seconds
- Stop: Pause the movie

Each video is a sequence of 30 frames (or images)

Understanding the Dataset :

The training data consists of a few hundred videos categorised into one of the five classes. Each video (typically 2-3 seconds long) is divided into a sequence of 30 frames(images). These videos have been recorded by various people performing one of the five gestures in front of a webcam - similar to what the smart TV will use.

The data is in a zip file. The zip file contains a 'train' and a 'val' folder with two CSV files for the two folders.



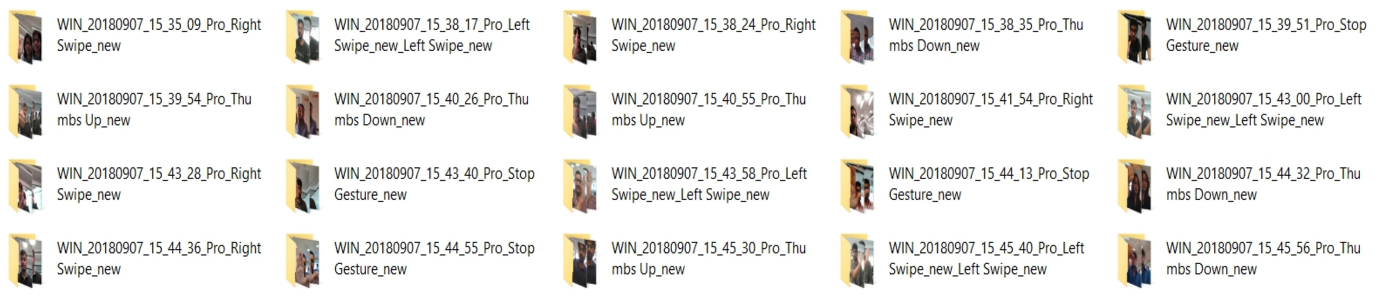
train

val

train.csv

val.csv

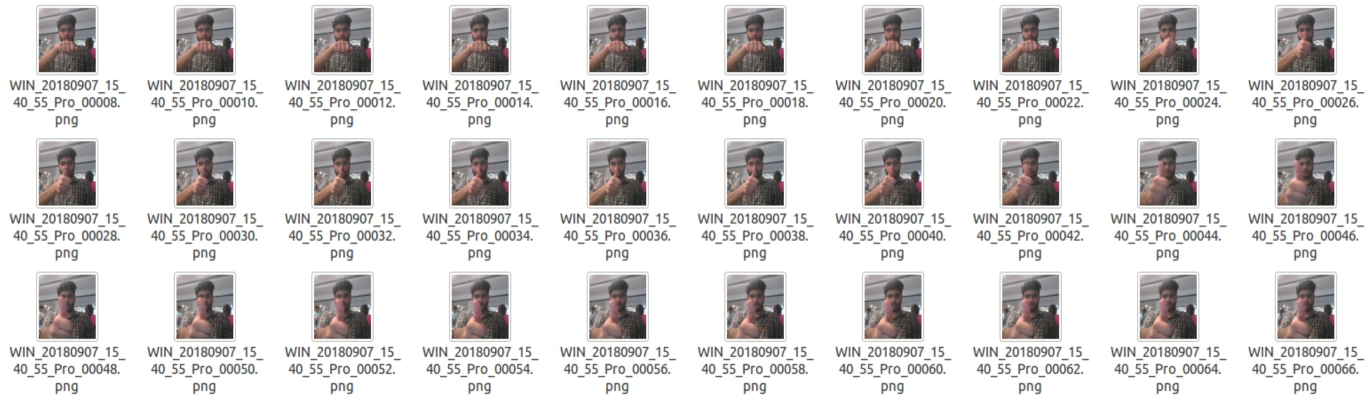
These folders are in turn divided into subfolders where each subfolder represents a video of a particular gesture.



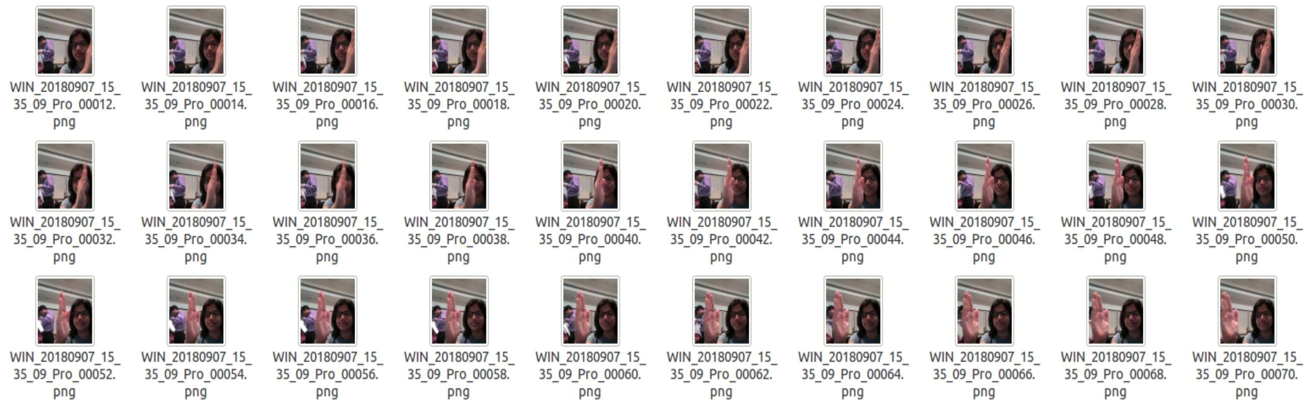
WIN_20180907_15_35_09_Pro_Right Swipe_new
WIN_20180907_15_38_17_Pro_Left Swipe_new_Left Swipe_new
WIN_20180907_15_38_24_Pro_Right Swipe_new
WIN_20180907_15_38_35_Pro_Thumbs Down_new
WIN_20180907_15_39_51_Pro_Stop Gesture_new
WIN_20180907_15_39_54_Pro_Thumbs Up_new
WIN_20180907_15_40_26_Pro_Thumbs Down_new
WIN_20180907_15_40_55_Pro_Thumbs Up_new
WIN_20180907_15_41_54_Pro_Right Swipe_new
WIN_20180907_15_43_00_Pro_Left Swipe_new_Left Swipe_new
WIN_20180907_15_43_28_Pro_Right Swipe_new
WIN_20180907_15_43_40_Pro_Stop Gesture_new
WIN_20180907_15_43_58_Pro_Left Swipe_new_Left Swipe_new
WIN_20180907_15_44_13_Pro_Stop Gesture_new
WIN_20180907_15_44_32_Pro_Thumbs Down_new
WIN_20180907_15_44_36_Pro_Right Swipe_new
WIN_20180907_15_44_55_Pro_Stop Gesture_new
WIN_20180907_15_45_30_Pro_Thumbs Up_new
WIN_20180907_15_45_40_Pro_Left Swipe_new_Left Swipe_new
WIN_20180907_15_45_56_Pro_Thumbs Down_new

Each subfolder, i.e. a video, contains 30 frames (or images).

- Thumbs Up


WIN_20180907_15_40_55_Pro_00008.png
WIN_20180907_15_40_55_Pro_00010.png
WIN_20180907_15_40_55_Pro_00012.png
WIN_20180907_15_40_55_Pro_00014.png
WIN_20180907_15_40_55_Pro_00016.png
WIN_20180907_15_40_55_Pro_00018.png
WIN_20180907_15_40_55_Pro_00020.png
WIN_20180907_15_40_55_Pro_00022.png
WIN_20180907_15_40_55_Pro_00024.png
WIN_20180907_15_40_55_Pro_00026.png
WIN_20180907_15_40_55_Pro_00028.png
WIN_20180907_15_40_55_Pro_00030.png
WIN_20180907_15_40_55_Pro_00032.png
WIN_20180907_15_40_55_Pro_00034.png
WIN_20180907_15_40_55_Pro_00036.png
WIN_20180907_15_40_55_Pro_00038.png
WIN_20180907_15_40_55_Pro_00040.png
WIN_20180907_15_40_55_Pro_00042.png
WIN_20180907_15_40_55_Pro_00044.png
WIN_20180907_15_40_55_Pro_00046.png
WIN_20180907_15_40_55_Pro_00048.png
WIN_20180907_15_40_55_Pro_00050.png
WIN_20180907_15_40_55_Pro_00052.png
WIN_20180907_15_40_55_Pro_00054.png
WIN_20180907_15_40_55_Pro_00056.png
WIN_20180907_15_40_55_Pro_00058.png
WIN_20180907_15_40_55_Pro_00060.png
WIN_20180907_15_40_55_Pro_00062.png
WIN_20180907_15_40_55_Pro_00064.png
WIN_20180907_15_40_55_Pro_00066.png

- Right Swipe


WIN_20180907_15_35_09_Pro_00012.png
WIN_20180907_15_35_09_Pro_00014.png
WIN_20180907_15_35_09_Pro_00016.png
WIN_20180907_15_35_09_Pro_00018.png
WIN_20180907_15_35_09_Pro_00020.png
WIN_20180907_15_35_09_Pro_00022.png
WIN_20180907_15_35_09_Pro_00024.png
WIN_20180907_15_35_09_Pro_00026.png
WIN_20180907_15_35_09_Pro_00028.png
WIN_20180907_15_35_09_Pro_00030.png
WIN_20180907_15_35_09_Pro_00032.png
WIN_20180907_15_35_09_Pro_00034.png
WIN_20180907_15_35_09_Pro_00036.png
WIN_20180907_15_35_09_Pro_00038.png
WIN_20180907_15_35_09_Pro_00040.png
WIN_20180907_15_35_09_Pro_00042.png
WIN_20180907_15_35_09_Pro_00044.png
WIN_20180907_15_35_09_Pro_00046.png
WIN_20180907_15_35_09_Pro_00048.png
WIN_20180907_15_35_09_Pro_00050.png
WIN_20180907_15_35_09_Pro_00052.png
WIN_20180907_15_35_09_Pro_00054.png
WIN_20180907_15_35_09_Pro_00056.png
WIN_20180907_15_35_09_Pro_00058.png
WIN_20180907_15_35_09_Pro_00060.png
WIN_20180907_15_35_09_Pro_00062.png
WIN_20180907_15_35_09_Pro_00064.png
WIN_20180907_15_35_09_Pro_00066.png
WIN_20180907_15_35_09_Pro_00068.png
WIN_20180907_15_35_09_Pro_00070.png

Note that all images in a particular video subfolder have the same dimensions but different videos may have different dimensions. Specifically, videos have two types of dimensions - either 360x360 or 120x160 (depending on the webcam used to record the videos).

Data Preprocessing :

Generator

This is one of the most important part of the code. In the generator, we have preprocessed the images as the images are of 2 different dimensions as well as created a batch of video frames. We have experimented with img_idx, y,z and resized & normalized such that we got high accuracy.

Understanding Generators: As we already know, in most deep learning projects we need to feed data to the model in batches. This is done using the concept of generators.

Creating data generators is probably the most important part of building a training pipeline. Although libraries such as Keras provide builtin generator functionalities, they are often restricted in scope and you have to write your own generators from scratch. In this project we will implement our own cutom generator, our generator will feed batches of videos, not images.

Let's take an example, assume we have 23 samples and we pick batch size as 10.

In this case there will be 2 complete batches of ten each

- Batch 1: 10
- Batch 2: 10
- Batch 3: 3

The final run will be for the remaining batch that was not part of the the full batch.

Full batches are covered as part of the for loop the remainder are covered post the for loop.

Note: this also covers the case, where in batch size is day 30 and we have only 23 samples. In this case there will be only one single batch with 23 samples.

Reading Video as Frames

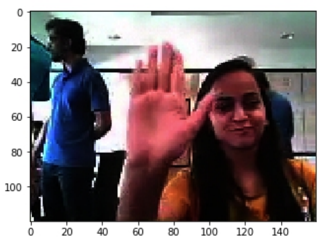
Note that in our project, each gesture is a broken into indivdual frame. Each gesture consists of 30 individual frames. While loading this data via the generator there is need to sort the frames if we want to maintain the temporal information.

The order of the images loaded might be random and so it is necessary to apply sort on the list of files before reading each frame.

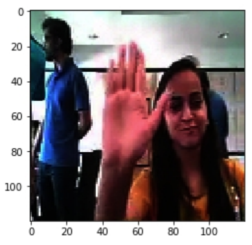
Resize

We converted each image of the train and test set into a matrix of size 84*84

```
In [4]: img = Image.open('images/sample_160_120.png')
        plt.imshow(img)

Out[4]: <matplotlib.image.AxesImage at 0x7fe12e6e4588>
```



```
In [4]: img = img.resize((new_width, new_height), Image.ANTIALIAS)
        plt.imshow(img)

Out[4]: <matplotlib.image.AxesImage at 0x7f7e9c5712e8>
```



Normalization

We used mean normaliztion for each of the channel in the image.

Implementation :

3D Convolutional Network, or Conv3D

Now, lets implement a 3D convolutional Neural network on this dataset. To use 2D convolutions, we first convert every image into a 3D shape : width, height, channels. Channels represents the slices of Red, Green, and Blue layers. So it is set as 3. In the similar manner, we will convert the input dataset into 4D shape in order to use 3D convolution for : length, breadth, height, channel (r/g/b).

Note: even though the input images are rgb (3 channel), we will perform image processing on each frame and the end individual frame will be grayscale (1 channel) for some models

Lets create the model architecture. The architecture is described below:

While we tried with multiple **filter size**, bigger filter size is resource intensive and we have done most experiment with 3*3 filter

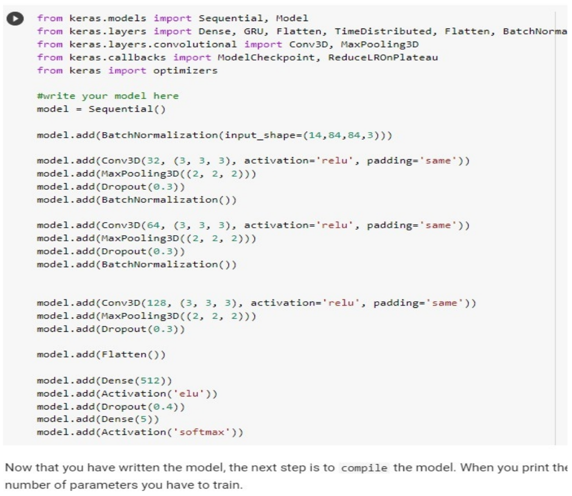
We have used **Adam** optimizer with its default settings. We have additionally used the ReduceLROnPlateau to reduce our learning alpha after 3 epoch on the result plateauing.

We have tried our model with 32, 64, 128 as the batch size. Others were throwing error except for 32 as the batch size.

We have used the **'relu'** activation function in the subsequent Conv-3D layers. After the Dense layer, we used **'elu'** activation function, followed by **'softmax'** activation function to predict in 5 different classes.

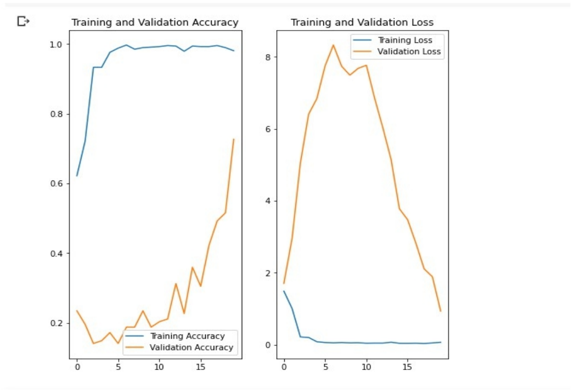
Model #1

Build a 3D convolutional network, based loosely on C3D including all the features like MaxPool layers, Dropout layers, Batch Normalization layers



```
from keras.models import Sequential, Model
from keras.layers import Dense, GRU, Flatten, TimeDistributed, Flatten, BatchNorma
from keras.layers.convolutional import Conv3D, MaxPooling3D
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from keras import optimizers

#write your model here
model = Sequential()

model.add(BatchNormalization(input_shape=(14,84,84,3)))

model.add(Conv3D(32, (3, 3, 3), activation='relu', padding='same'))
model.add(MaxPooling3D((2, 2, 2)))
model.add(Dropout(0.3))
model.add(BatchNormalization())

model.add(Conv3D(64, (3, 3, 3), activation='relu', padding='same'))
model.add(MaxPooling3D((2, 2, 2)))
model.add(Dropout(0.3))
model.add(BatchNormalization())


model.add(Conv3D(128, (3, 3, 3), activation='relu', padding='same'))
model.add(MaxPooling3D((2, 2, 2)))
model.add(Dropout(0.3))

model.add(Flatten())

model.add(Dense(512))
model.add(Activation('elu'))
model.add(Dropout(0.4))
model.add(Dense(5))
model.add(Activation('softmax'))
```

Now that you have written the model, the next step is to `compile` the model. When you print the
number of parameters you have to train.



```
Epoch 00019: saving model to model_init_2021-05-0906_03_46.326218/model-00019-0.05048-0.98958-1.88956-0.51562.h5
Epoch 20/20
21/21 [==============================] - 49s 2s/step - loss: 0.0811 - categorical_accuracy: 0.9775 - val_loss: 0.9364 - val_categorical_accuracy: 0.7266

Epoch 00020: saving model to model_init_2021-05-0906_03_46.326218/model-00020-0.06812-0.98065-0.93636-0.72656.h5
<tensorflow.python.keras.callbacks.History at 0x7f8c496e9990>
```

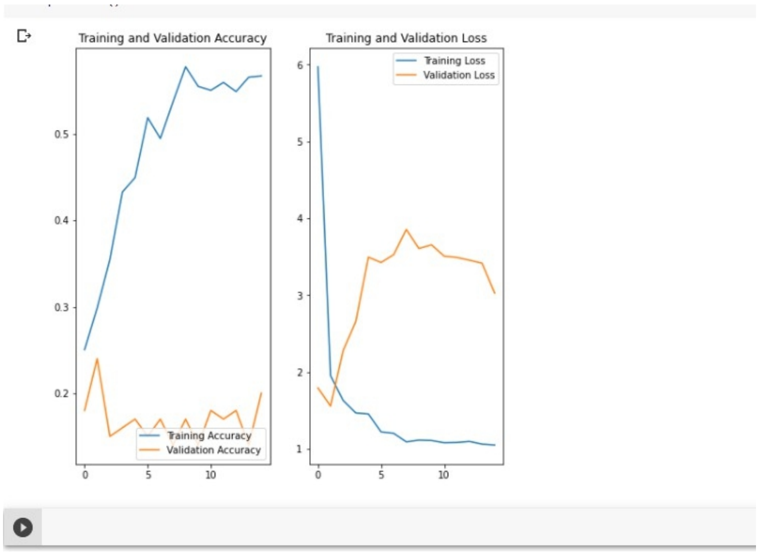
The train and validation accuracies are **98%** and **73%** respectively.

Model #2

Build a 3D convolutional network, By removing the 2nd, 3rd and 4th Batch Normalization layer.

```
[68] #write your model here
     model = Sequential()

     model.add(BatchNormalization(input_shape=(14,84,84,3)))

     model.add(Conv3D(32, (3, 3, 3), activation='relu', padding='same'))
     model.add(MaxPooling3D((2, 2, 2)))
     #model.add(BatchNormalization())


     model.add(Conv3D(64, (3, 3, 3), activation='relu', padding='same'))
     model.add(MaxPooling3D((2, 2, 2)))
     #model.add(BatchNormalization())


     model.add(Conv3D(128, (3, 3, 3), activation='relu', padding='same'))
     model.add(MaxPooling3D((2, 2, 2)))
     #model.add(BatchNormalization())
     model.add(Dropout(0.25))


     model.add(Flatten())

     model.add(Dense(512))
     model.add(Activation('elu'))
     model.add(Dropout(0.25))
     model.add(Dense(5))
     model.add(Activation('softmax'))
```

Now that you have written the model, the next step is to `compile` the model. When you print the `summary` of the model, you'll see the total number of parameters you have to train.



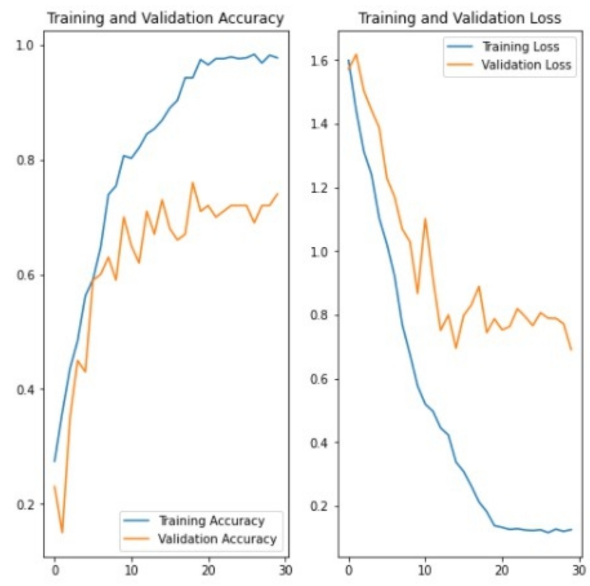
```
Epoch 00014: saving model to model_init_2021-05-0907_39_34.411331/model-00014-1.06200-0.56561-3.41576-0.14000.h5
Epoch 15/15
21/21 [==============================] - 48s 2s/step - loss: 1.0469 - categorical_accuracy: 0.5678 - val_loss: 3.0276 - val_categorical_accuracy: 0.2000

Epoch 00015: saving model to model_init_2021-05-0907_39_34.411331/model-00015-1.04988-0.56712-3.02756-0.20000.h5
<tensorflow.python.keras.callbacks.History at 0x7fd270193fd0>
```

The train and validation accuracies are **57%** and **20%** respectively.

Model #3

```
[68] #write your model here
     model = Sequential()

     model.add(BatchNormalization(input_shape=(14,84,84,3)))

     model.add(Conv3D(32, (3, 3, 3), activation='relu', padding='same'))
     model.add(MaxPooling3D((2, 2, 2)))
     #model.add(BatchNormalization())


     model.add(Conv3D(64, (3, 3, 3), activation='relu', padding='same'))
     model.add(MaxPooling3D((2, 2, 2)))
     #model.add(BatchNormalization())


     model.add(Conv3D(128, (3, 3, 3), activation='relu', padding='same'))
     model.add(MaxPooling3D((2, 2, 2)))
     #model.add(BatchNormalization())
     model.add(Dropout(0.25))


     model.add(Flatten())

     model.add(Dense(512))
     model.add(Activation('elu'))
     model.add(Dropout(0.25))
     model.add(Dense(5))
     model.add(Activation('softmax'))
```

Now that you have written the model, the next step is to `compile` the model. When you print the `summary` of the model, you'll see the total number of parameters you have to train.

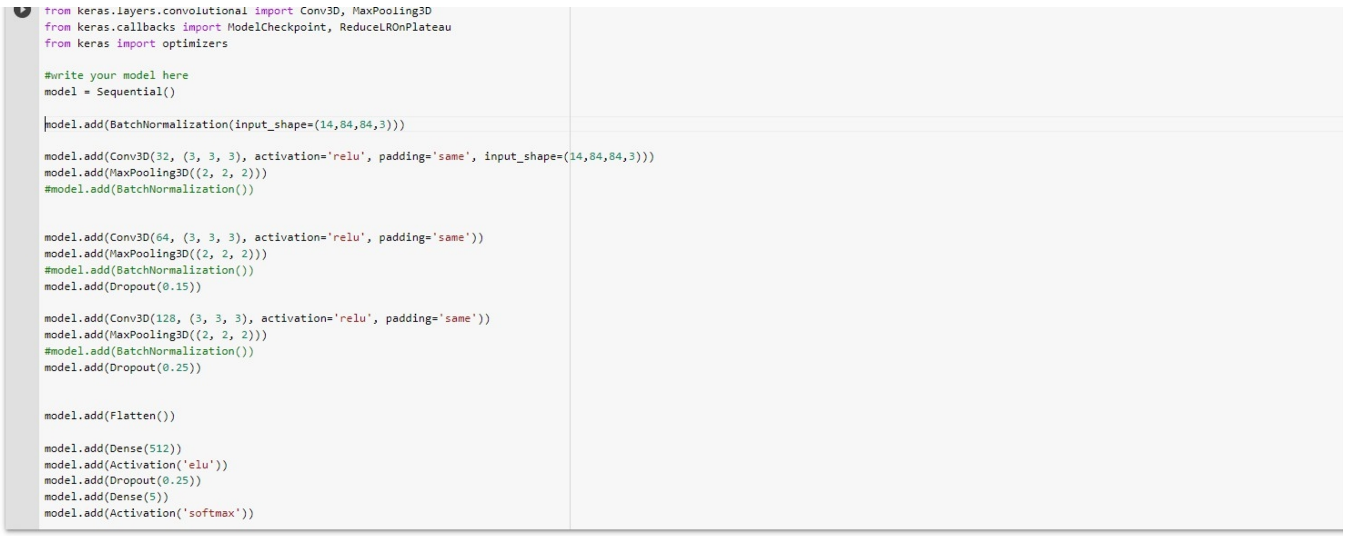


```
Epoch 00029: saving model to model_init_2021-05-0911_13_45.864137/model-00029-0.12119-0.98190-0.77233-0.72000.h5
Epoch 30/30
21/21 [==============================] - 46s 2s/step - loss: 0.1179 - categorical_accuracy: 0.9808 - val_loss: 0.6915 - val_categorical_accuracy: 0.7400

Epoch 00030: saving model to model_init_2021-05-0911_13_45.864137/model-00030-0.12626-0.97738-0.69149-0.74000.h5
<tensorflow.python.keras.callbacks.History at 0x7f1ecf9980d0>
```

The train and validation accuracies are **98%** and **74%** respectively.

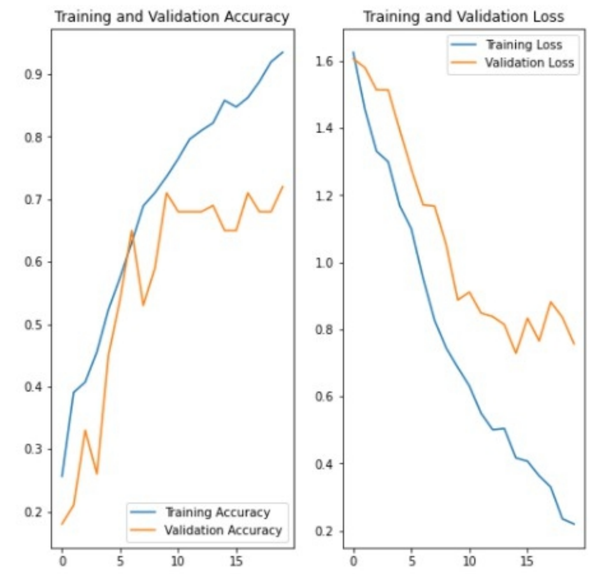
Model #4

Build a 3D convolutional network, after adding a Dropout layer(added after 2nd MaxPool layer) with rate 0.15



```
from keras.layers.convolutional import Conv3D, MaxPooling3D
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from keras import optimizers

#write your model here
model = Sequential()

model.add(BatchNormalization(input_shape=(14,84,84,3)))

model.add(Conv3D(32, (3, 3, 3), activation='relu', padding='same', input_shape=(14,84,84,3)))
model.add(MaxPooling3D((2, 2, 2)))
#model.add(BatchNormalization())


model.add(Conv3D(64, (3, 3, 3), activation='relu', padding='same'))
model.add(MaxPooling3D((2, 2, 2)))
#model.add(BatchNormalization())
model.add(Dropout(0.15))

model.add(Conv3D(128, (3, 3, 3), activation='relu', padding='same'))
model.add(MaxPooling3D((2, 2, 2)))
#model.add(BatchNormalization())
model.add(Dropout(0.25))


model.add(Flatten())

model.add(Dense(512))
model.add(Activation('elu'))
model.add(Dropout(0.25))
model.add(Dense(5))
model.add(Activation('softmax'))
```

```
    plt.show()
```

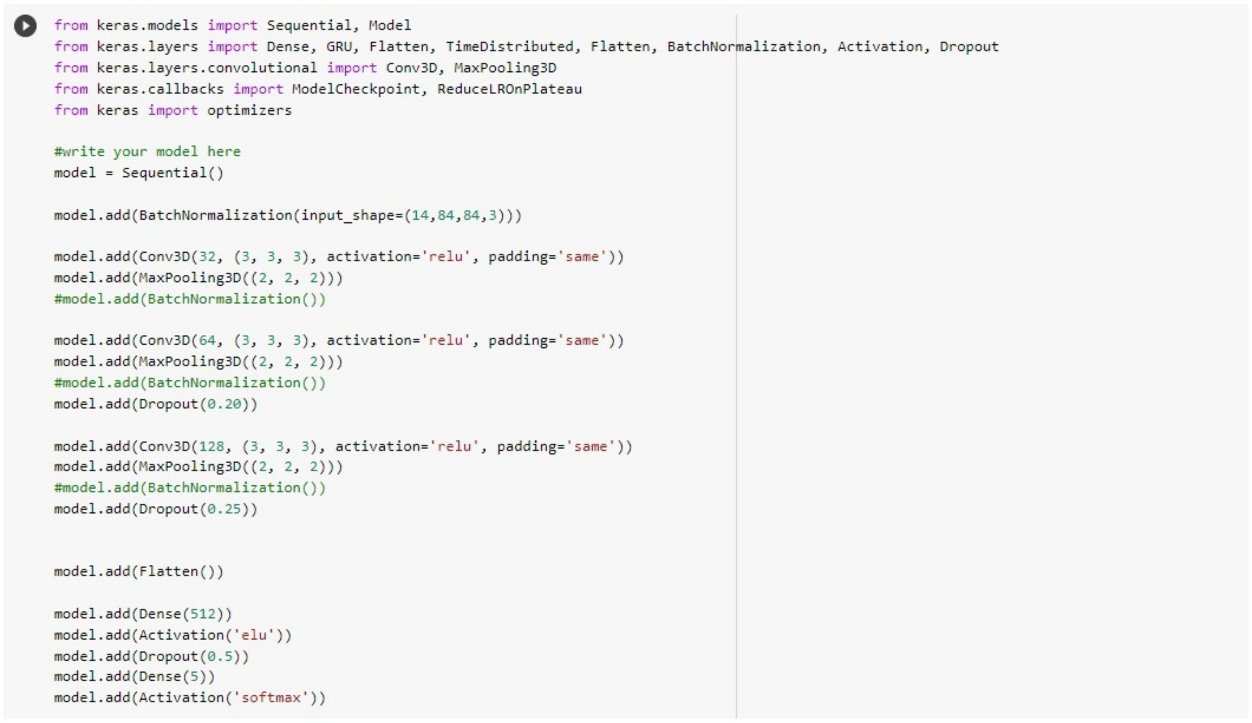


```
Epoch 00019: saving model to model_init_2021-05-0911_47_27.407062/model-00019-0.23528-0.92006-0.83557-0.68000.h5
Epoch 20/20
21/21 [==============================] - 45s 2s/step - loss: 0.2400 - categorical_accuracy: 0.9236 - val_loss: 0.7571 - val_categorical_accuracy: 0.7200

Epoch 00020: saving model to model_init_2021-05-0911_47_27.407062/model-00020-0.22001-0.93514-0.75711-0.72000.h5
<tensorflow.python.keras.callbacks.History at 0x7f1ed15f76d0>
```

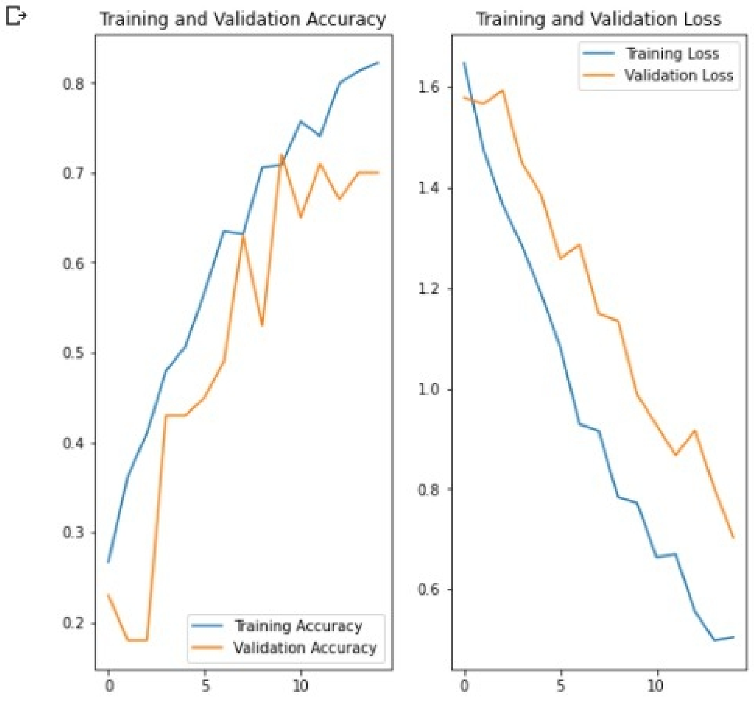
The train and validation accuracies are **92%** and **72%** respectively.

Model #5

Build a 3D convolutional network, after changing the Dropout layer(added after 2nd MaxPool layer) with rate 0.15



```
from keras.models import Sequential, Model
from keras.layers import Dense, GRU, Flatten, TimeDistributed, Flatten, BatchNormalization, Activation, Dropout
from keras.layers.convolutional import Conv3D, MaxPooling3D
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from keras import optimizers

#write your model here
model = Sequential()

model.add(BatchNormalization(input_shape=(14,84,84,3)))

model.add(Conv3D(32, (3, 3, 3), activation='relu', padding='same'))
model.add(MaxPooling3D((2, 2, 2)))
#model.add(BatchNormalization())

model.add(Conv3D(64, (3, 3, 3), activation='relu', padding='same'))
model.add(MaxPooling3D((2, 2, 2)))
#model.add(BatchNormalization())
model.add(Dropout(0.20))

model.add(Conv3D(128, (3, 3, 3), activation='relu', padding='same'))
model.add(MaxPooling3D((2, 2, 2)))
#model.add(BatchNormalization())
model.add(Dropout(0.25))


model.add(Flatten())

model.add(Dense(512))
model.add(Activation('elu'))
model.add(Dropout(0.5))
model.add(Dense(5))
model.add(Activation('softmax'))
```



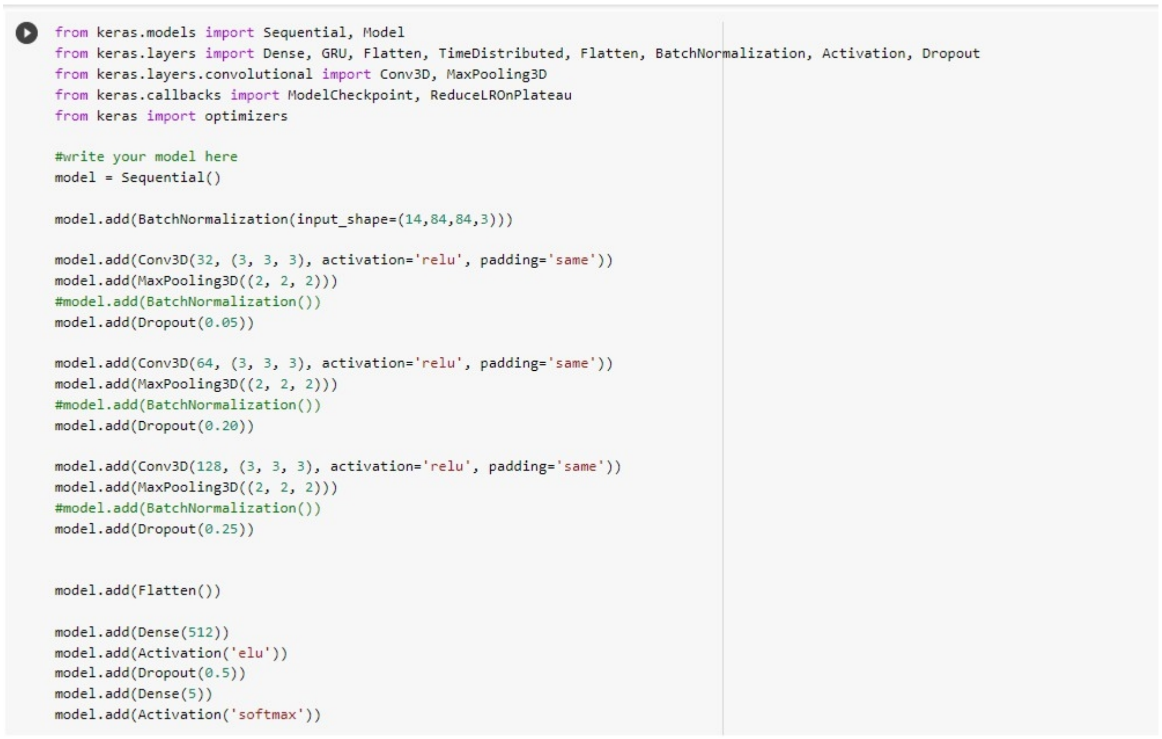
```
Epoch 00014: saving model to model_init_2021-05-0913_19_28.197335/model-00014-0.49824-0.81297-0.80113-0.70000.h5
Epoch 15/15
21/21 [==============================] - 47s 2s/step - loss: 0.5202 - categorical_accuracy: 0.8039 - val_loss: 0.7030 - val_categorical_accuracy: 0.7000

Epoch 00015: saving model to model_init_2021-05-0913_19_28.197335/model-00015-0.50447-0.82202-0.70297-0.70000.h5
<tensorflow.python.keras.callbacks.History at 0x7f1eba242ed0>
```

The train and validation accuracies are **80%** and **70%** respectively.

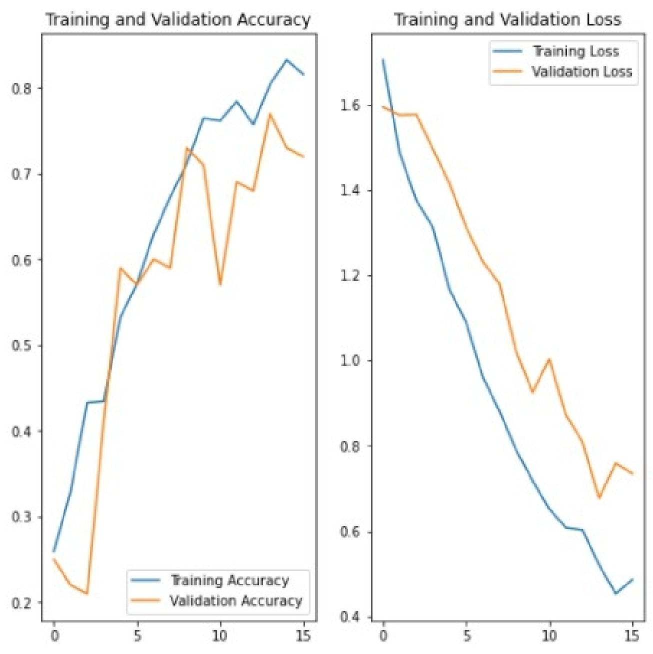
Model #6

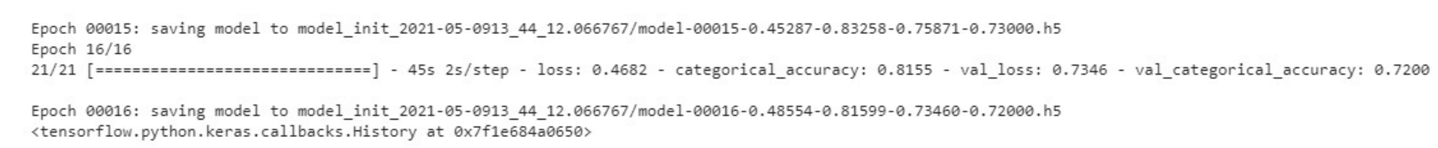
Build a 3D convolutional network, after changing the Dropout layer(added after 2nd MaxPool layer) with rate 0.20.

Also adding Dropout layers after 1st MaxPool layer and 3rd MaxPool layer with rate 0.05 and 0.25 respectively.



```
from keras.models import Sequential, Model
from keras.layers import Dense, GRU, Flatten, TimeDistributed, Flatten, BatchNormalization, Activation, Dropout
from keras.layers.convolutional import Conv3D, MaxPooling3D
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from keras import optimizers

#write your model here
model = Sequential()

model.add(BatchNormalization(input_shape=(14,84,84,3)))

model.add(Conv3D(32, (3, 3, 3), activation='relu', padding='same'))
model.add(MaxPooling3D((2, 2, 2)))
#model.add(BatchNormalization())
model.add(Dropout(0.05))

model.add(Conv3D(64, (3, 3, 3), activation='relu', padding='same'))
model.add(MaxPooling3D((2, 2, 2)))
#model.add(BatchNormalization())
model.add(Dropout(0.20))

model.add(Conv3D(128, (3, 3, 3), activation='relu', padding='same'))
model.add(MaxPooling3D((2, 2, 2)))
#model.add(BatchNormalization())
model.add(Dropout(0.25))


model.add(Flatten())

model.add(Dense(512))
model.add(Activation('elu'))
model.add(Dropout(0.5))
model.add(Dense(5))
model.add(Activation('softmax'))
```



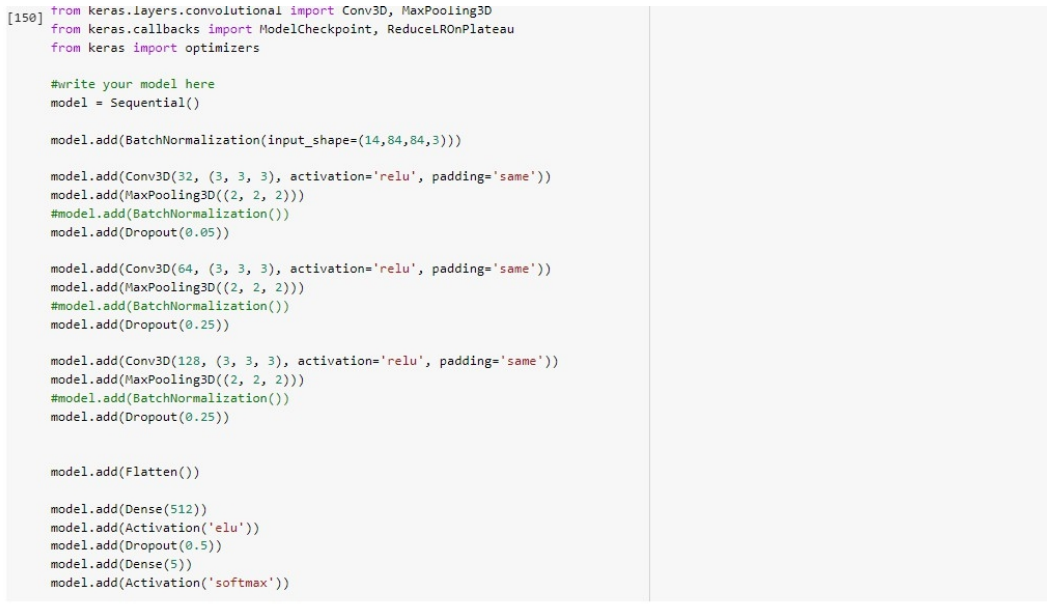


```
Epoch 00015: saving model to model_init_2021-05-0913_44_12.066767/model-00015-0.45287-0.83258-0.75871-0.73000.h5
Epoch 16/16
21/21 [==============================] - 45s 2s/step - loss: 0.4682 - categorical_accuracy: 0.8155 - val_loss: 0.7346 - val_categorical_accuracy: 0.7200

Epoch 00016: saving model to model_init_2021-05-0913_44_12.066767/model-00016-0.48554-0.81599-0.73460-0.72000.h5
<tensorflow.python.keras.callbacks.History at 0x7f1e684a0650>
```

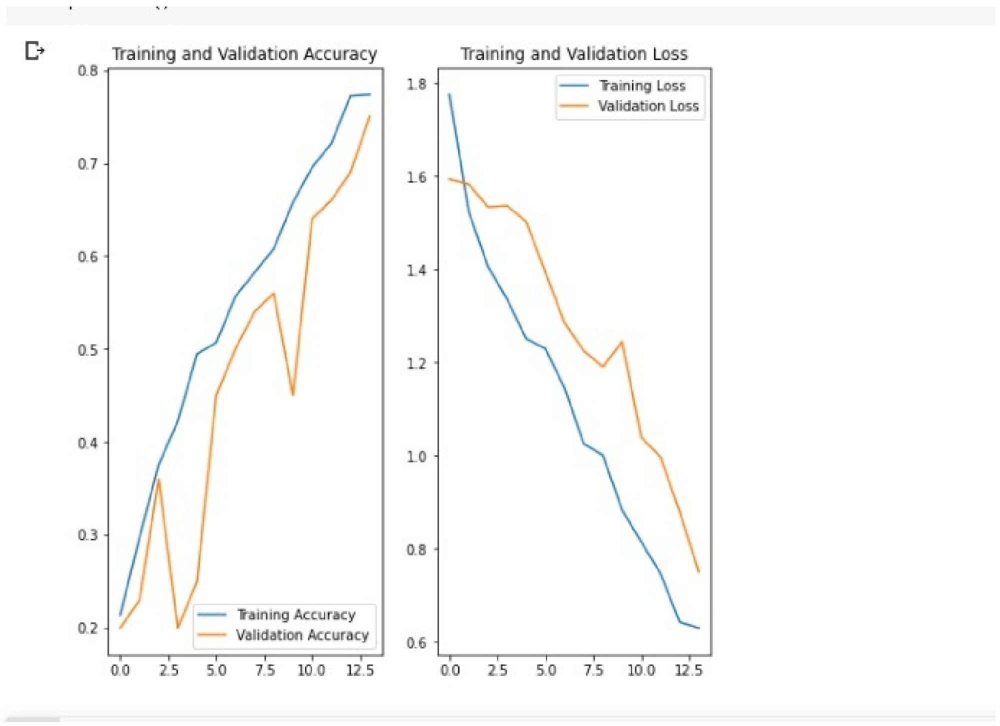
The train and validation accuracies are **82%** and **72%** respectively.

Model #7 Final Model

Build a 3D convolutional network, after changing the Dropout layer(added after 2nd MaxPool layer) with rate 0.25 after the previous model(model-6).



```
[150] from keras.layers.convolutional import Conv3D, MaxPooling3D
      from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
      from keras import optimizers

      #write your model here
      model = Sequential()

      model.add(BatchNormalization(input_shape=(14,84,84,3)))

      model.add(Conv3D(32, (3, 3, 3), activation='relu', padding='same'))
      model.add(MaxPooling3D((2, 2, 2)))
      #model.add(BatchNormalization())
      model.add(Dropout(0.05))

      model.add(Conv3D(64, (3, 3, 3), activation='relu', padding='same'))
      model.add(MaxPooling3D((2, 2, 2)))
      #model.add(BatchNormalization())
      model.add(Dropout(0.25))

      model.add(Conv3D(128, (3, 3, 3), activation='relu', padding='same'))
      model.add(MaxPooling3D((2, 2, 2)))
      #model.add(BatchNormalization())
      model.add(Dropout(0.25))


      model.add(Flatten())

      model.add(Dense(512))
      model.add(Activation('elu'))
      model.add(Dropout(0.5))
      model.add(Dense(5))
      model.add(Activation('softmax'))
```



```
Epoch 00013: saving model to model_init_2021-05-0914_19_13.867308/model-00013-0.64262-0.77225-0.87942-0.69000.h5
Epoch 14/14
21/21 [==============================] - 47s 2s/step - loss: 0.6414 - categorical_accuracy: 0.7777 - val_loss: 0.7506 - val_categorical_accuracy: 0.7500

Epoch 00014: saving model to model_init_2021-05-0914_19_13.867308/model-00014-0.62993-0.77376-0.75060-0.75000.h5
<tensorflow.python.keras.callbacks.History at 0x7f1e5e4708d0>
```

The train and validation accuracies are **78%** and **75%** respectively.

Final Conclusion :

Model 7 gave us **train accuracy of 78% and validation accuracy of 75%** using all the 14 frames. The same model is submitted for the review. While we did try model lesser frames and by using full frames but we felt more comfortable using 14 frames. Resizing and other preprocessing also did not affect much on the final accuracy.