**Deep Learning Course Project – Gesture Recognition**

*By:*

* *Anuraag Godika*
* *Nitesh Maan*
* *Amar Nandan*
* *Akshay Rohankar*

**Problem Statement**

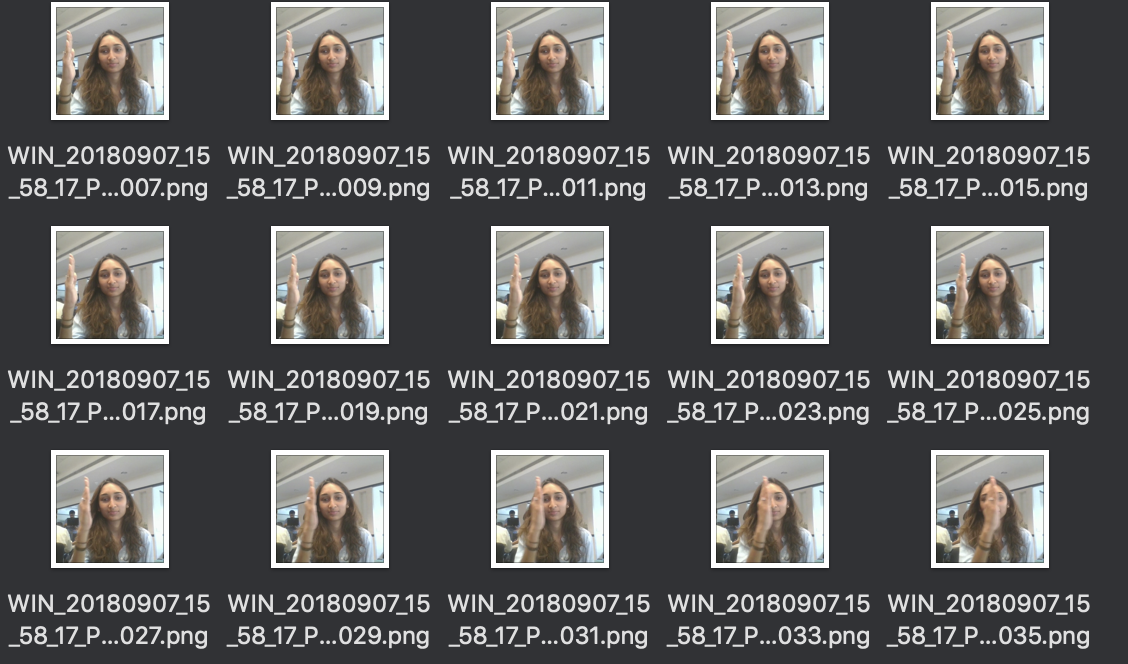
As a data scientist at a home electronics company which manufactures state of the art smart televisions. We want to develop a cool feature in the smart-TV that can recognise five different gestures performed by the user which will help users control the TV without using a remote.

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

**Understanding the Dataset**

The training data consists of a few hundred videos categorized 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.



**Objective**

Our task is to train different model on the 'train' folder to predict the action performed in each sequence or video and which performs well on the 'val' folder as well. The final test folder for evaluation is withheld - final model's performance will be tested on the 'test' set.

**Model Architectures**

For analyzing videos using neural networks, **two types of architectures**are used commonly. One is the standard **CNN + RNN architecture** in which you pass the images of a video through a CNN which extracts a feature vector for each image, and then pass the sequence of these feature vectors through an RNN. This is something you are already familiar with (in theory).

The other popular architecture used to process videos is a natural extension of CNNs - a **3D convolutional network**. In this project, you will try both these architectures.

1. **3D Convolutional Network, or 3D Conv**

3D convolutions are a natural extension to the 2D convolutions. Just like in 2D conv, we move the filter in two directions (x and y), in 3D conv, we move the filter in three directions (x, y and z). In this case, the input to a 3D conv is a video (which is a sequence of 30 RGB images). If we assume that the shape of each image is 120x120x3, for example, the video becomes a 4-D tensor of shape 120x120x3x30 which can be written as (120x120x30)x3 where 3 is the number of channels. Hence, deriving the analogy from 2-D convolutions where a 2-D kernel/filter (a square filter) is represented as (fxf)xc where f is filter size and c is the number of channels, a 3-D kernel/filter (a 'cubic' filter) is represented as (fxfxf)xc (here c = 3 since the input images have three channels). This cubic filter will now '3D-convolve' on each of the three channels of the (120x120x30) tensor.

3DConv

Input

Dense

Output

1. **Convolutions + RNN**

The conv2D 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 (for a classification problem such as this one).

Dense

Output

RNN

2DConv

Input

**Data Generator**

This is one of the most important part of the code. In the generator, We are going to preprocess the images as we have images of 2 different dimensions as well as create a batch of video frames. We did experiment with img\_idx, y,z and normalization such that we get a high accuracy.

Experimented with variety of permutations and combination of the different augmentations techniques and parameters.

The generator, ***yields*** the batch output and labels along with other augmentations as listed below.

Augmentations:

* Flipping

Flip array in the left/right direction. Flip the entries in each row in the left/right direction. Columns are preserved, but appear in a different order than before.

* Translation

AffineTransform with

**Shifting Left :** transaltion\_transforms = tf.AffineTransform(translation=(rn.choice(range(15,40)), 0))

**Shifting Right :** transaltion\_transforms = tf.AffineTransform(translation=(rn.choice(range(-40,-15)), 0))

**Shifting Up :** transaltion\_transforms = tf.AffineTransform(translation=(0, rn.choice(range(-40,-15))))

**Shifting Down :** transaltion\_transforms = tf.AffineTransform(translation=(0, rn.choice(range(15, 40))))

**Zoom in and Zoom Out :** scale\_transforms = tf.AffineTransform(scale=(zoom, zoom))

* Noise

Adding Random Noise may help with lighting distortions and make the model more robust in general.

* Contrast and Brightness

Rescaling the intensities of the contrast and Brightness of the Images for adding lighting variety to the images.

* Jitters

Adding Jitters further to make the model more robust in general.

* Rotations

Rotate : rot(image, rn.choice(range(5,20)))

anti-rotate images : rot(image, rn.choice(range(-20,-5))),

**Attempted model configurations:**

| **Experiment #** | **Model** | **Result** | **Decision + Explanation** |
| --- | --- | --- | --- |
| 1 | Conv3D | OOM Error | Reduce the batch size and Reduce the number of neurons in Dense layer |
| 2 | Conv3D | Generator error | Fix the return statement of generator so that it doesn't raise StopIteration prematurely |
| 3 | Conv3D | Validation Accuracy 0.22 | Increase neurons in the Dense layer |
| 4 | Conv3D | Val Accuracy 0.20, Staying flat across Epochs | Reduce learning rate to 0.01 and add clipNorm |
| 5 | Conv3D | Val Accuracy 0.48, Train Accuracy 0.77 | Overfitting. Add Dropout. |
| 6 | Conv3D | Val Accuracy 0.46, Train Accuracy 0.54 | Overfitting has reduced, but accuracy hasn't. Use consecutive frames of images to catch movement better. |
| 7 | Conv3D | Val Accuracy 0.54, Train Accuracy 0.54 | Switch to LSTM. Conv3D isn't giving the desired accuracy. |
| 8 | ConvLSTM | OOM when allocating Tensor on GPU | Reduce batch\_size |
| 9 | ConvLSTM | Val Accuracy 0.46, Train Accuracy 0.74 | Overfitted. Add Dropout. |
| 10 | ConvLSTM | Network taking long time to train | Reduce layers |
| 10 | ConvLSTM | Val Accuracy 0.44, Train Accuracy 0.59 | Overfitting reduced, but validation accuracy low. Add Image augmentation to generator function. |
| 11 | ConvLSTM | Val Accuracy 0.49, Train Accuracy 0.54 | Accuracy remains below 50%. Switch to CNN+LSTM |
| 12 | CNN+LSTM | Training is too slow | Reduce Conv and LSTM layers to 1 each |
| 13 | CNN+LSTM | Training is fast, Accuracy is still below 0.5 | Fix bugs in generator function (double normalization getting applied) |
| 14 | CNN+LSTM | Beginning to generalize better, Accuracy hovers near 0.50 | Reduce frame count more - use 1 in every 3 frames |
| 15 | CNN+LSTM | Validation Accuracy 0.52, Train Accuracy 0.54 | Increase Epoch count to 20. Network is generalizing well, maybe it needs more Epochs to train. |
| 16 | CNN+LSTM | Learning at the same rate as above | Increase learning rate to 0.01 |
| 17 | CNN+LSTM | Learning faster, but accuracy stays near 0.52 | Reduce batch size to 7 |
| 18 | CNN+LSTM | Not much difference to accuracy | Add Randomization to Image Augmentation |
| 19 | CNN+LSTM | Training and Validation Accuracy tracking close to each other Network is generalizing well. Val accuracy 0.55, Train accuracy 0.56 | Add a CNN Layer |
|  |  | Same as above | Increase Epoch count more |
|  |  | Same as above | Try the other 2 architectures - Conv3D, ConvLSTM |

* Normalized the images. Normalizing the RGB values of an image can at times be a simple and effective way to get rid of distortions caused by lights and shadows in an image.
* Augmented the data with various trasformations.
* Various iterations and combinations of filter sizes, padding and stride length were experimented.
* It was observed, as the Number of trainable parameters increases the model takes much more time for training.
* SGD() and Adam() Optimizers were experimented.
* Played around with different learning rates and ReduceLROnPlateau was used to change the learning rate if the monitored metrics (val\_loss) remains unchanged in between epochs.
* Experimented with different model configurations and hyper-parameters

**Submition Jupyter Notebook Models**

**#Models**

**Model – 1: Conv3D**

**Training Parameters:**

num\_classes = 5

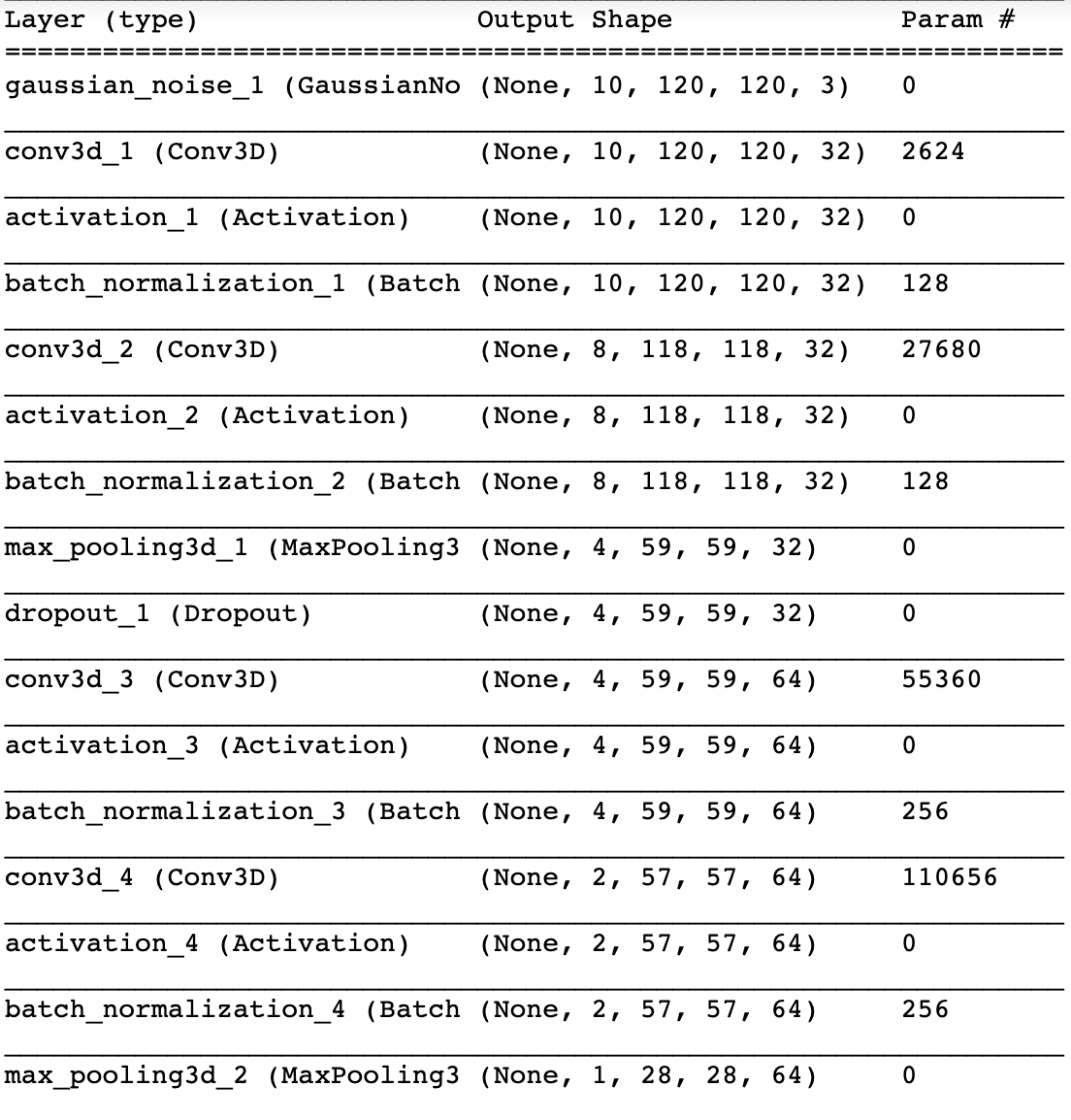
num\_epochs = 30

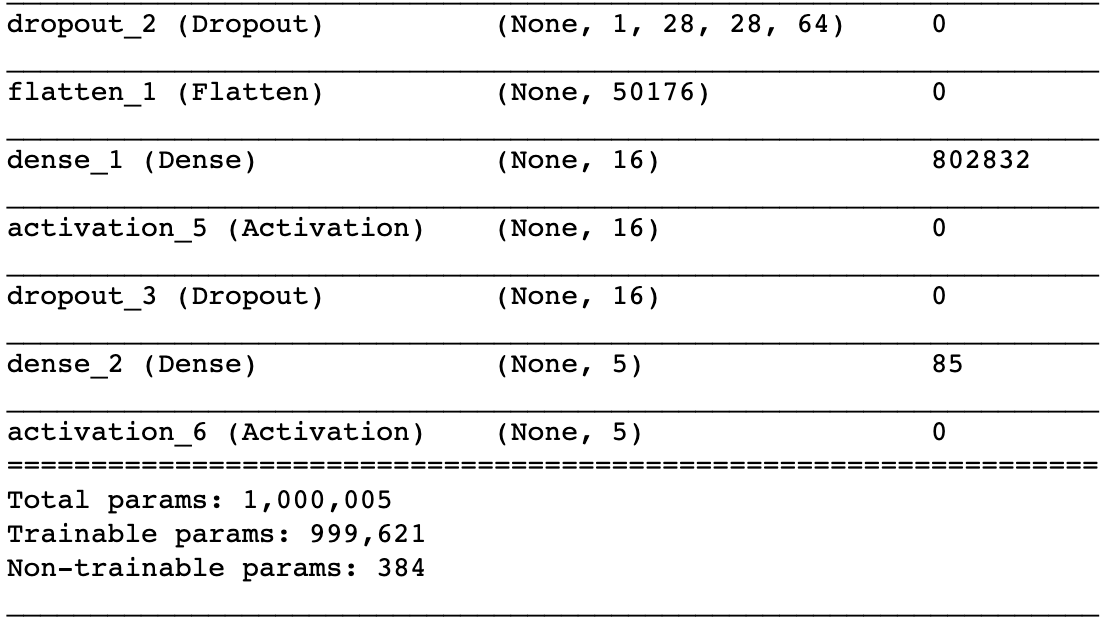
loss='categorical\_crossentropy'

optimizer = SGD (lr=0.01, clipnorm=1.)

steps\_per\_epoch = steps\_per\_epoch \* 6 🡪 #Because we are augmenting each image

LR = ReduceLROnPlateau(monitor='val\_loss', factor=0.2,patience=3, min\_lr=0.001)

****



Total params: 1,000,005

Trainable params: 999,621

Best Model 1 Accuracy:

model\_init\_2019-06-2320\_14\_42.038750/model-00030-1.57303-0.37557-1.29646-0.52000.h5

- categorical\_accuracy: 0.3744

- val\_loss: 1.2965

- val\_categorical\_accuracy: 0.5200

**Model – 2: model\_LSTM**

**Training Parameters:**

num\_classes = 5

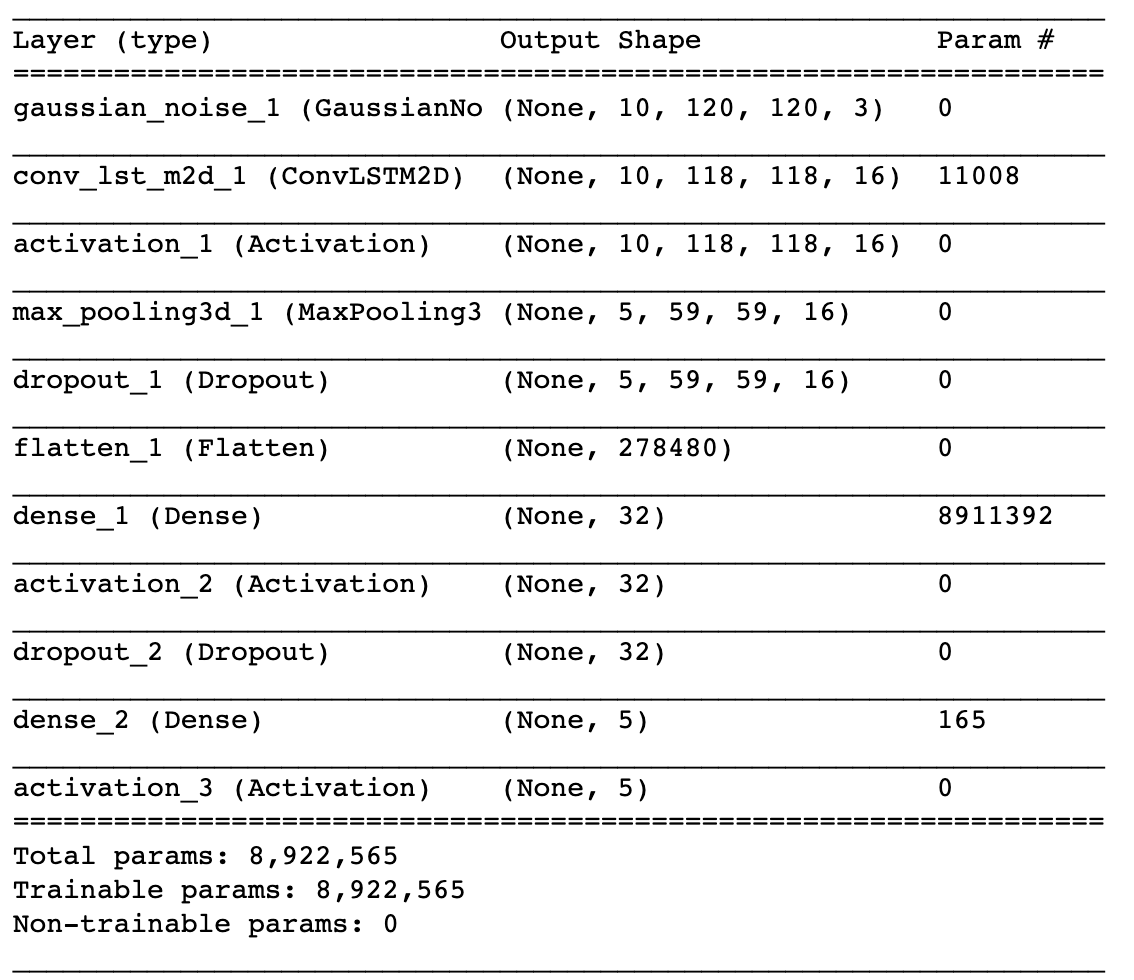
num\_epochs = 30

loss='categorical\_crossentropy'

optimizer = SGD (lr=0.01, clipnorm=1.)

steps\_per\_epoch = steps\_per\_epoch \* 6 🡪 #Because we are augmenting each image

LR = ReduceLROnPlateau(monitor='val\_loss',factor=0.2,patience=3, min\_lr=0.001)



Total params: 8,922,565

Trainable params: 8,922,565

Best Model 2 Accuracy:

model\_init\_2019-06-2310\_06\_50.311870/model-00030-1.61525-0.55153-1.46911-0.60000.h5

- categorical\_accuracy: 0.5502

- val\_loss: 1.4691

- val\_categorical\_accuracy: 0.6000

**Model – 3: model\_LSTM**

**Training Parameters:**

num\_classes = 5

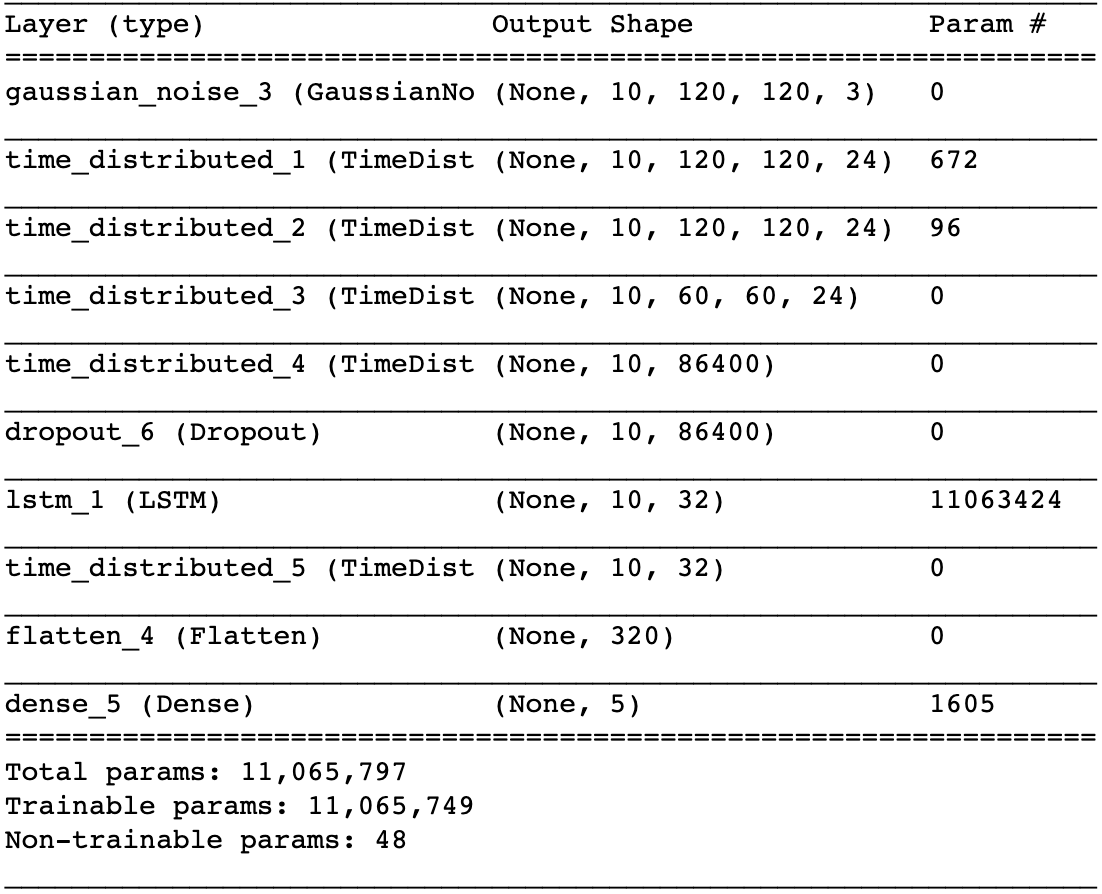
num\_epochs = 30

loss='categorical\_crossentropy'

optimizer = SGD (lr=0.01, clipnorm=1.)

steps\_per\_epoch = steps\_per\_epoch \* 6 🡪 #Because we are augmenting each image

LR = ReduceLROnPlateau(monitor='val\_loss',factor=0.2,patience=3, min\_lr=0.001)

****

Total params: 11,065,797

Trainable params: 11,065,749

Non-trainable params: 48

Best Model 3 Accuracy:

model\_init\_2019-06-2410\_39\_51.646150/model-00015-1.06081-0.55933-1.10982-0.49000.h5

- loss: 1.0495

- categorical\_accuracy: 0.5589

- val\_loss: 1.1352

- val\_categorical\_accuracy: 0.5300

**Suggestions:**

Further suggestions for improvement:

* **Using transfer learning** : Using a pre-trained Resnet50/ImageNet to identify the initial feature vectors and passing them further to a RNN for sequence information before finally passing it to a softmax layer for classification of gestures. ( This was attempted but incomplete due to lack of time.)
* **Using GRU :** A GRU model inplace of LSTM appears to be a good choice. Trainable Parameters of a GRU are far less than that of a LSTM. Therefore would have resulted in faster computations. How ever its effect of the validation accuracies could be checked to determine if it is actually a good alternative over LSTM.
* **Deeper Understanding of Data:** The video clips where recorder in different backgrounds, lightings, persons and different cameras where used. Further exploration on the available images could give some more information about the them. This added information can be exploited in favor inside the generator function adding more stability and accuracy to model.
* **Tuning training parameters:** Experimenting with other combinations of training parameters like, batch\_size, num\_epochs, learning rate, optimizers functions can further help gain accurate model.
* **Tuning Model Parameters :** Experimenting with other combinations of Model Parameters like the filter size, paddings, stride\_length, batch\_normalization, dropouts etc can further help gain accurate model.