**Gesture Recognition - Write Up**

**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 . 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 .

**Data Provided:**

Each video is a sequence of 30 frames (or images). There are 666 videos provided as training data and 100 videos provided as validation data all images in a particular video subfolder have the same dimensions different videos may have different dimensions. Specifically, videos have two types of dimensions - either 360x360 or 120x160 There are two csv(one for train, one for validation) files having path of videos Our task is to train a model on the 'train' folder which performs well on the 'val' folder as well Thus, there are two types of architecture commonly used for analysing videos, both explained below. Convolutions + RNN 3D Convolutional Network, or Conv3D .

**Approach**

Clearly it’s a problem that would require a deep learning solution . Ideally this task can be performed in 2 ways:

* Using the Convolutional 3d approach .
* Using the CNN + LSTM approach

We picked up the Conv3D approach as it is more lightweight since it uses a comparatively lesser amount of parameters which ensures that the model does not take too long to train .

**Summary**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#Exp** | **Model** | **Params** | **Result** | **Decision + Explanation** |
| **1** | **Conv3D+ ADAM** | **Batch\_size**:32  **Image\_size**:84 x 84 ,  **Frames**: 15,  **Epochs** : 5 | loss: 1.1783  categorical\_accuracy: 0.4587  val\_loss: 46.6730  val\_categorical\_accuracy: 0.2400 | **A small ablation experiment to see our architecture is able to learn from the training data (overfits)** |
| **2.1** | **Conv3D+ADAM** | **Batch\_size**:32  **Image\_size**:84 x 84 ,  **Frames**: 15,  **Epochs** : 5 | **1 epoch takes 40-43 sec** | **We found out that length of sequence and image size has more impact on the training times as compared to the batch size**  **Also we need to keep an eye out for out of memory error if we try to load in too much data (large batch size and large input dimensions) into the RAM . (even though we have written generator for the same)** |
| **2.2** | **Conv3D+ADAM** | **Batch\_size**:32  **Image\_size**:84 x 84 ,  **Frames**: 21,  **Epochs** : 5 | **1 epoch takes 54-57 sec** |
| **2.3** | **Conv3D+ADAM** | **Batch\_size**:64  **Image\_size**:120 x 120 ,  **Frames**: 21,  **Epochs** : 5 | **1 epoch takes 59-61 sec** |
| **2.4** | **Conv3D+ADAM** | **Batch\_size**:128  **Image\_size**:120x120 ,  **Frames**: 21,  **Epochs** : 5 | **OOM when allocating tensor with shape [128,16,21,120,120]** |
| **3** | **Conv3D+ADAM** | **Batch\_size**:32  **Image\_size**:120x120 ,  **Frames**: 21,  **Epochs** : 35 | loss: 0.4328  categorical\_accuracy: 0.8313  val\_loss:0.5943  val\_categorical\_accuracy: 0.7500 | **As we can see , the 1st convolution layer has 16 kernels , the second layer has 32 and the subsequent layers has kernels that increase by a factor of x2 . It is kept in such a way as we move deeper into the network more complex features need to be derived from the output of previous layers .**  **Also we chose the filter size of 3 x 3 x 3 as it enables the neuron to look at the neighboring pixels in 8 directions .** |
| **4** | **Conv3D+ADAM** | **Batch\_size**:32  **Image\_size**:120x120 ,  **Frames**: 30,  **Epochs** : 35 | loss: 0.0832  categorical\_accuracy: 0.9798  val\_loss: 0.4992  val\_categorical\_accuracy: 0.8300 | **If we consider all the frames in the video , then we observe the learning process/ training becomes slower .**  **Also we get amazon accuracy figures for training and validation datasets .** |
| **5 (final model)** | **Conv3D + SGD** | **Batch\_size**:32  **Image\_size**:120x120 ,  **Frames**: 30,  **Epochs** : 35 | loss:0.1674 - categorical\_accuracy:0.9552 val\_loss:0.4339 val\_categorical\_accuracy: 0.8400 | **SGD, converges quickly as compared to ADAM optimizer alos we got higher accuracy with sgd and we choose this as our final model .** |

**Experimentation Detailed :**

**Experiment #1 - Ablation Experiment**

Objective : This experiment was sort of a **trial experiment** , to validate if we are headed in the right direction .

* Initially we picked up 15 sequences/images from the video , with a batch size of 32 .
* Each image was resized to 84 x 84
* Since this was sort of an ablation experiment we picked a very simple model whose layers are as follows :

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv3d\_33 (Conv3D) (None, 15, 84, 84, 16) 1312

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

activation\_30 (Activation) (None, 15, 84, 84, 16) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_52 (Batc (None, 15, 84, 84, 16) 64

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling3d\_30 (MaxPooling (None, 7, 42, 42, 16) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv3d\_34 (Conv3D) (None, 7, 42, 42, 16) 2064

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

activation\_31 (Activation) (None, 7, 42, 42, 16) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_53 (Batc (None, 7, 42, 42, 16) 64

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling3d\_31 (MaxPooling (None, 3, 21, 21, 16) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten\_12 (Flatten) (None, 21168) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_35 (Dense) (None, 64) 1354816

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_54 (Batc (None, 64) 256

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_22 (Dropout) (None, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_36 (Dense) (None, 5) 325

=================================================================

Total params: 1,358,901

Trainable params: 1,358,709

Non-trainable params: 192

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Observation/Result :

1. We ran this model for 5 epochs and everything seems to work fine . We noticed an increase in training accuracy and decrease in loss which provided enough evidence that we are headed into the right direction .

2. Upon running this model for longer time / mode epochs , we noticed that model begins to overfit . (categorical\_accuracy accuracy increases , val\_categorical\_accuracy remains constant)

3.After 5 epochs , we got the following observation

loss: 1.1783

categorical\_accuracy: 0.4587

val\_loss: 46.6730

val\_categorical\_accuracy: 0.2400

**Experiment #2**

Objective: Keeping the same architecture we want to find out how the training times are affected by changing the value of batch\_size , length of image sequence and image size .

* For batch\_size=32 , len\_sequence=15 , img\_size = 84 x 84 , 1 epoch takes about 40-43 seconds .
* For batch\_size=32 , len\_sequence=21 , img\_size = 84 x 84 , 1 epoch takes about 54-57 seconds .
* For batch\_size=32 , len\_sequence=21 , img\_size = 120 x 120 , 1 epoch takes about 57-60 seconds .
* For batch\_size=64 , len\_sequence=21 , img\_size = 120 x 120 , 1 epoch takes about 59-61 seconds .
* For batch\_size=128 , len\_sequence=21 , img\_size = 120 x 120 , we run into error **OOM when allocating tensor with shape[128,16,21,120,120]**

Observation :

1. We find out that length of sequence and image size has more impact on the training times as compared to the batch size .
2. As the length of training sequences increases or the image size increases the training time shoots up as well . The main reason is , as we modify the length/shape of the input sequence the trainable parameters changes accordingly.
3. Also we need to keep an eye out for out of memory error if we try to load in too much data (large batch size and large input dimensions) into the RAM . (even though we have written generator for the same)

**Experiment , #3**

Objective : Let’s make the architecture more complex by adding more layers .

Here is the model summary that we used :

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv3d (Conv3D) (None, 21, 120, 120, 16) 1312

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

activation (Activation) (None, 21, 120, 120, 16) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization (BatchNo (None, 21, 120, 120, 16) 64

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling3d (MaxPooling3D) (None, 10, 60, 60, 16) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv3d\_1 (Conv3D) (None, 10, 60, 60, 32) 13856

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

activation\_1 (Activation) (None, 10, 60, 60, 32) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_1 (Batch (None, 10, 60, 60, 32) 128

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling3d\_1 (MaxPooling3 (None, 5, 30, 30, 32) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv3d\_2 (Conv3D) (None, 5, 30, 30, 64) 55360

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

activation\_2 (Activation) (None, 5, 30, 30, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_2 (Batch (None, 5, 30, 30, 64) 256

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling3d\_2 (MaxPooling3 (None, 2, 15, 15, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv3d\_3 (Conv3D) (None, 2, 15, 15, 128) 221312

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

activation\_3 (Activation) (None, 2, 15, 15, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_3 (Batch (None, 2, 15, 15, 128) 512

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling3d\_3 (MaxPooling3 (None, 1, 7, 7, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten (Flatten) (None, 6272) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense (Dense) (None, 128) 802944

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_4 (Batch (None, 128) 512

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout (Dropout) (None, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 64) 8256

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_5 (Batch (None, 64) 256

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_1 (Dropout) (None, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 5) 325

=================================================================

Total params: 1,105,093

Trainable params: 1,104,229

Non-trainable params: 864

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Interesting thing to note here is that , although this model has more layers than the one used in ablation experiments . But the trainable params is around 1.1 million as compared to 1.3 million of the previous.

***Conv3d filter sizes :***



We modified the **patience** parameter in the ReduceLROnPlateau to 2 and set the **min\_lr = 0.001** , **factor=0.2**  and a **starting learning rate of 0.1 .** By setting patience to 2 , it basically means the training process would wait 2 epochs and if the val\_loss does not decrease it would then reduce the learning rate by a factor of 2 , capping the minimum learning rate to 0.001

Observations/Result :

1. As we can see , the 1st convolution layer has 16 kernels , the second layer has 32 and the subsequent layers has kernels that increase by a factor of x2 . It is kept in such a way as we move deeper into the network more complex features need to be derived from the output of previous layers .

2. Also we chose the filter size of 3 x 3 x 3 as it enables the neuron to look at the neighboring pixels in 8 directions .

3. Keeping the batch size =32 , image size=120 x 120 , number of frame =21 we train the model for 35 epochs using the Adam Optimizer .

4. The Initial learning rate of 0.1 was reduced to 0.02 after epoch 9 , which further reduced to 0.003 after epoch 16.

4.The best iteration was around epoch 18 with following metrics :

loss: 0.4328

categorical\_accuracy: 0.8313

val\_loss:0.5943

val\_categorical\_accuracy: 0.7500

After this the learning process becomes stagnant .

**Experiment #4**

Objective**:**  Let’s increase the number of frames to 30 (full video) , keeping the above parameters and model exactly the same .

Observation/Result :

1. If we consider all the frames in the video , then we observe the learning process/ training becomes slower .

2. We also observed a considerable improvement in performance

loss: 0.0832

categorical\_accuracy: 0.9798

val\_loss: 0.4992

val\_categorical\_accuracy: 0.8300 .

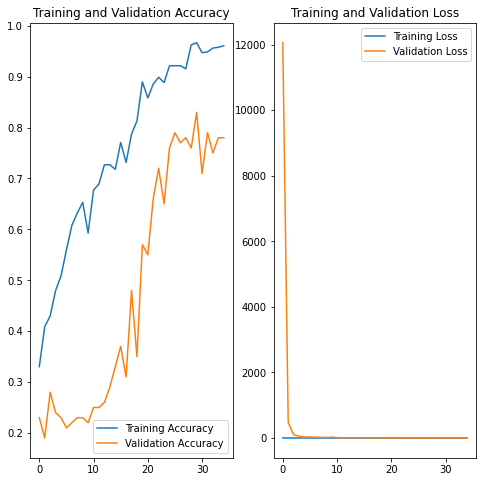
**Experiment #5**

**Objective :** Modifying the ReduceLROnPlateau params and trying it out with different optimizers (ADAM & SGD) , keeping the model the same as above.

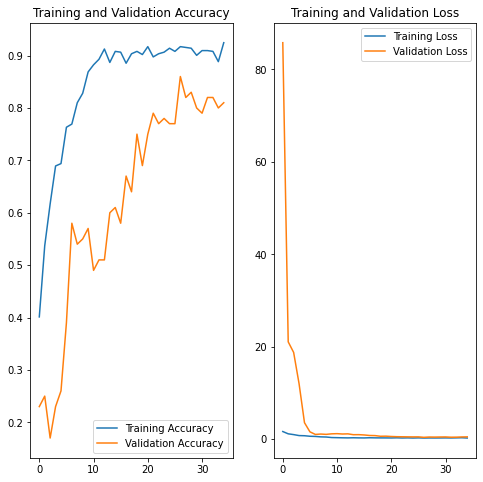
.

Observation/Result :

*Adams* :

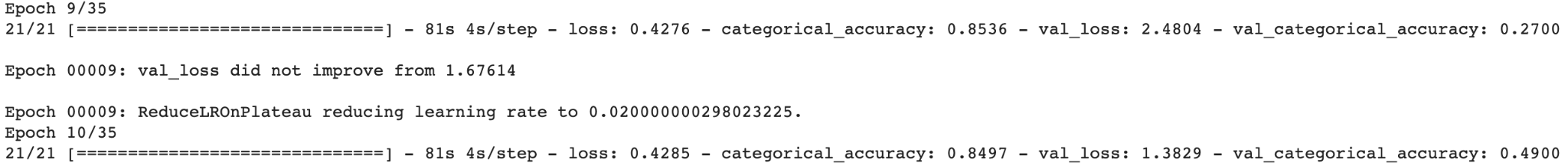


*SGD* :

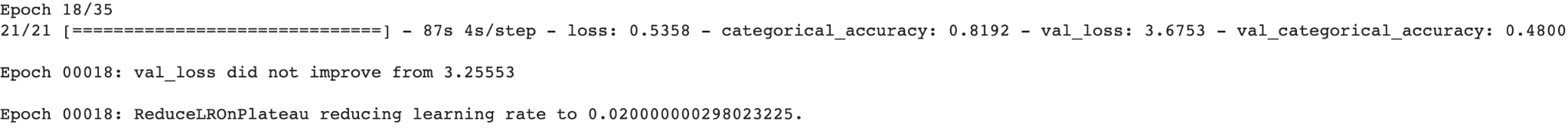


We found out that SGD converges quickly , just under 10 epochs sgd is able to achieve a training accuracy of 80%+ and validation accuracy of about 55% where as Adam’s progresses slowly with training accuracy around 65% and validation accuracy less than 40% .

Also one more interesting thing to note is that when we are training with the SGD optimiser the learning rate is reduced to 0.02 , just after 9 epochs .



Whereas in the case of Adam optimizer , the learning rate reduces to 0.02 after about 18 epochs .



This means that for this use case , SGD converges quickly (lesser epochs) as compared to Adams. Also with SGD we got higher accuracy about 84% on validation set .

Hence we make this as our **final model**

**Model-Configurations:**

**Optimiser : SGD**

**Model : Conv3D**

**Frames : 30**

**Batch Size : 32**

**Image Size : 120 x 120**

**Epochs : 35**

**Metrics:**

**loss:0.1674**

**categorical\_accuracy:0.9552**

**val\_loss:0.4339**

**val\_categorical\_accuracy: 0.8400**