Gesture Recognition Case Study

In this project, we are supposed to build a deep learning model which takes videos of 30 fps as inputs and helps us in recognizing which gesture in the input has been performed to link to a well-defined action in the television.

Each gesture is linked to some action to be performed on the television.

- **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

We first begin with building a generator function. The generator function is used for better utilization of memory as memory is used according to the batch size.

Now, to further reduce the training time, we do not use all the 30 frames. We may reduce the number of frames to be used per video using the following option:

- **Using Alternate Images:** We chose this option because consecutive frames/images might contain similar information. Hence, we may use only one of them. Through this method, we take the number of frames as just 15.

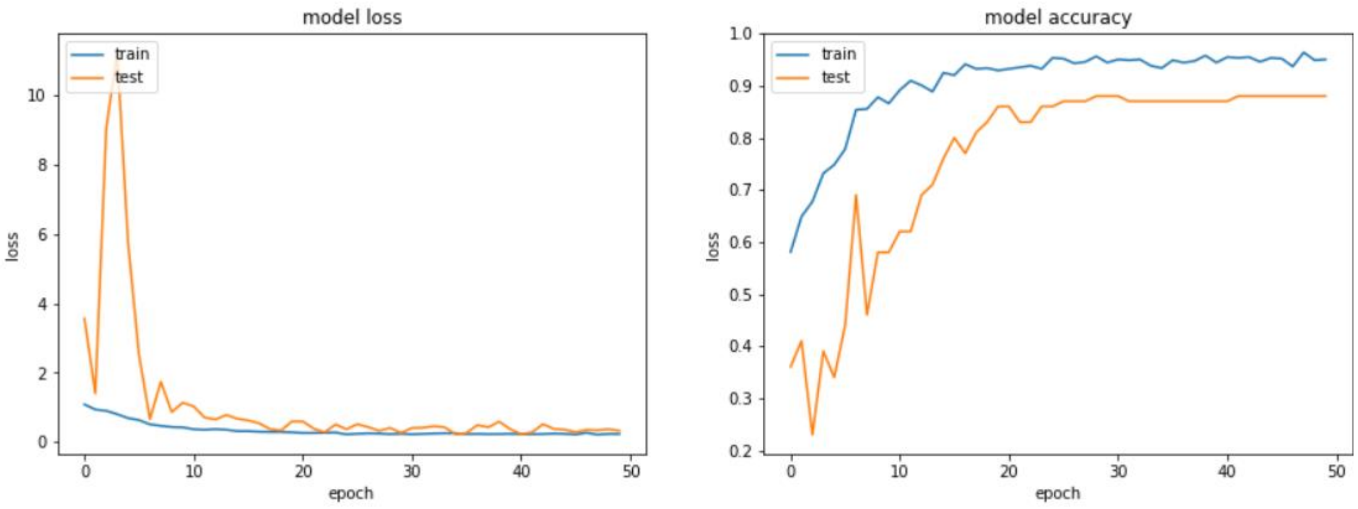
We then resize the images using cv2.resize() command from the Open CV Library in Python. We use the **divide by 255 method** to normalize the images.

Model History: Each of us tried at least 30 experiments before. But in all these experiments, our models were overfitting. (You may check those experiments here). We already used different kind of models which reduced the number of training parameters. Upon careful thought, we realized that our generator code was not properly made. We assumed that the images shall have central cropping which was wrong. So we used the cv2.resize() function to resize our images. After the change in generator code, we started getting good results. We chose the models from which we got the best results during our earlier experiments. The results of our experiments after changing the generator code are as follows:

Experiment No.	Model Type	Brief Description	Batch Size	Image Size	Number of Epochs	Result	Explanation for next step
1	Conv3D	Three convolution groups with 8,16 and 32 filters of size (3,3,3). Each conv groups consists of two conv layers and each group is followed by maxpooling layer with (2,2,2) filter size. This is followed by a dense layer with 128 neurons and then again by a dense layer with 16 neurons. Lastly, there is a softmax layer with 5 neurons.	64	(90,90)	12	Training Accuracy: 100%, Validation Accuracy: 24%	Clearly, the model is learning but there is overfitting.
2	Conv3D with Dropouts	The same model as mentioned above but we add dropout of value 0.5 in the first dense layer and another dropout of value 0.25 in the second dense layer.	64	(90,90)	50	Training Accuracy: 72.4%, Validation Accuracy: 71%	Overfitting has been reduced. Moreover, seeing the Loss Accuracy plot, we think there is scope for further improvement rate. We increase the learning rate to 0.1 for faster training.
3	Conv 3D with Dropouts	The same model as mentioned above but we change the learning rate to 0.1.	64	(90,90)	50	Training Accuracy: 95%, Validation Accuracy: 88%	Overfitting is reduced and we have improvement in accuracies. Moreover, seeing the loss accuracy plot, we realize the model training has reached its plateau (flatter curve).
4	CNN+RNN: Self-Made	Five time distributed conv layers with number of filters from 2^3 to 2^7. Each filter is of (3,3,3) size.	32	(90,90)	13	Training Accuracy: 80%, Validation Accuracy: 30%	We observe overfitting. We now try Transfer Learning for faster training.
5	CNN+RNN: Using Transfer Learning with Mobilenet as base model.	We use Mobilenet as base model and then to the time based analysis using GRU	32	(90,90)	14	Training Accuracy: 76%, Validation Accuracy: 75%	There is no overfitting. However, seeing the Loss Accuracy plot, we observe that there is scope for improvement in accuracy.

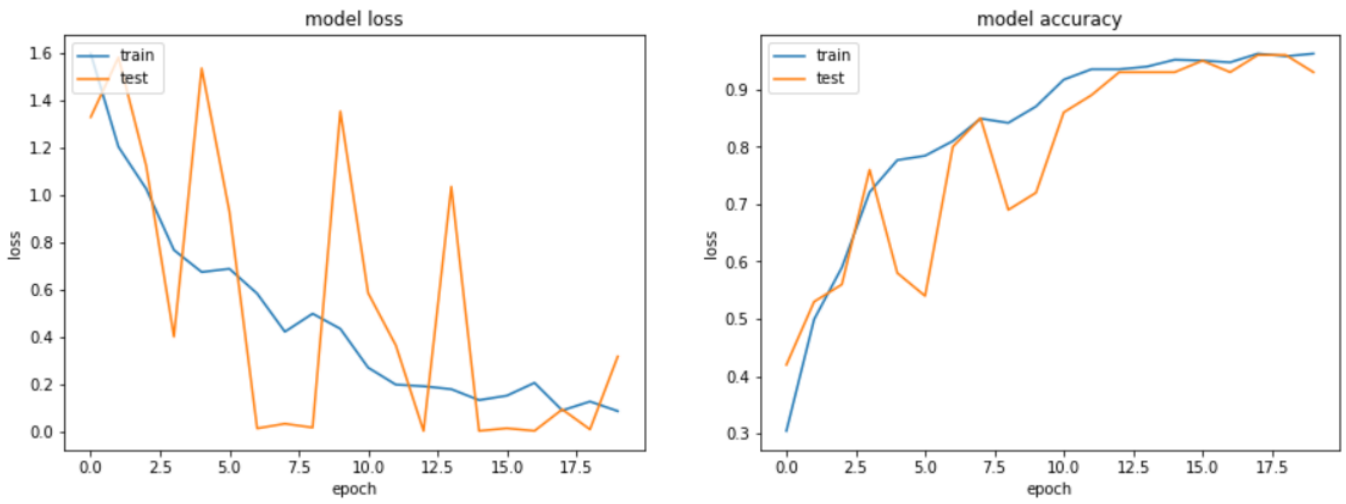
6	CNN+RNN: Using Transfer Learning with Mobilenet as base model.	Same as above	20	(90,90)	25	Training Accuracy:86%, Validation Accuracy:81%	We observe that there is an improvement in accuracy with reduction in batch size. Therefore, we build another model with even lesser batch size and a little higher learning rate.
7	CNN+RNN: Using Transfer Learning with Mobilenet as base model.	Same as above	16	(90,90)	20	Training Accuracy:96%, Validation Accuracy:93%	There is hardly any overfitting and we reach very high accuracy.
8	Conv3D with Dropouts	We use the same model as in Experiment 3 but apply edge detection as a pre-processing step in the data.	64	(90,90)	14	Training Accuracy: 95.3%, Validation Accuracy:35%	There is overfitting in the model.
9	Conv 3D with Dropouts	Same as above but we add dropouts in the 3rd convolution group layers with 0.2 q value.	32	(90,90)	13	Training Accuracy: 60%, Validation Accuracy:16%	Again, there is overfitting. Therefore, we do not try this approach anymore and stick with the best models achieved before.

<u>BEST CONV 3D MODEL:</u>



Experiment 3: This model has 95% Training Accuracy and 88% Validation Accuracy. Moreover, the loss is almost similar for both. You may find the .h5 link here.

<u>BEST CNN+RNN MODEL:</u>



Experiment 7: This model has 96% Training Accuracy and 93% validation accuracy. This is our best model because it gives us high accuracy with minimal overfitting. You may find the .h5 link here.

<u>FINAL MODEL:</u>

We choose the Mobilenet CNN+RNN model because it gives us high accuracy with minimal overfitting. Even though this model has more than 3 million parameters, we are certain that it will run very fast as per the following reference from [tensorflow:](tensorflow)

Model Name	Device	CPU, 4 threads	GPU	NNAPI
Mobilenet_1.0_224(float)	Pixel 3	23.9 ms	6.45 ms	13.8 ms
	Pixel 4	14.0 ms	9.0 ms	14.8 ms
Mobilenet_1.0_224 (quant)	Pixel 3	13.4 ms	---	6.0 ms
	Pixel 4	5.0 ms	---	3.2 ms