Deep Learning Project- Gesture Recognition

GESTURE RECOGNITION PROJECT

Our aim is to build a 3D Convolutional Network and a CNN+RNN architecture that will be able to predict the 5 gestures correctly.

PROBLEM STATEMENT:

A home electronics company manufactures state of the art smart televisions. We have 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 video is a sequence of 30 frames (or images).

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

For analysing 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.
- The other popular architecture used to process videos is a natural extension of CNNs - **a 3D convolutional network**.

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 - like what the smart TV will use.

Initially, we built a base model for **3D convolutional network**. We used the following model architecture for the base model.

- 3 convolutional layers (Conv3D)
- 2 dense layers
- 1 softmax dense output layer
- Batch size: 20
- Optimizer: ‘Adam’

- Activation function: 'Relu'
- # of Epochs: 10
- Image dimension (height X width) = 100 X 100
- Without augmentation, Batch Normalization and dropout layers

With this model as base, we experimented various models by tuning the hyper parameters like **batch size, number of epochs, image dimensions, frames sampled for processing, learning rate, pool size, number of layers, kernel size** and so on to get better accuracy and minimum validation loss.

The changes from the base model are mentioned in the '**HYPER PARAMETERS &MODEL ARCHITECTURE'** column of the following table. The impact of the change is written in the '**DECISION + EXPLANATION'** field of the table.

The following table consists of the experiments done to build the model to predict the gestures from the given data set.

EXP #	MODEL	HYPER PARAMETERS &MODEL ARCHITECTURE	RESULTS	DECISION + EXPLANATION
1	Conv3D	- Augmentation, batch normalization, Drop out layers=False	Train Accuracy: **0.89** Val Accuracy: **0.77**	Model is overfitting. Let us add augmentation to see if it improves. # Parameters= **669,381**
2	Conv3D	- With Augmentation & all other configuration remains the same as previous model	Train Accuracy: **0.93** Val Accuracy: **0.86**	Though the accuracy increases, it still overfits. Let's change the dimensions. # Parameters= **669,381**
3	Conv3D	- With Augmentation - Image Dimension = 160x160	Train Accuracy: **0.96** Val Accuracy: **0.66**	Accuracy has decreased drastically for larger dimensions. Also, a greater number of parameters are required for larger dimensions. Let's now try to increase the batch size with reduced dimensions. # Parameters= **1,717,957**
4	Conv3D	- With Augmentation - Image Dimension = 100x100 - Batch size = 50	Train Accuracy: **0.79** Val Accuracy: **0.70**	Overfitting has been handled to some extent, but accuracy is decreased. Let's increase the no of epochs. # Parameters= **669,381**
5	Conv3D	- Epoch = 20	Train Accuracy: **0.90** Val Accuracy: **0.68**	Accuracy has dropped. Let's change the selected sample frames. # Parameters= **669,381**

6	Conv3D	- Epoch = 10, Batch size = 20 - img_idx = [4,7,11,14,17,20,24,27]	Train Accuracy: **0.62** Val Accuracy: **0.51**	For the newly selected samples of frames, the model does not give better accuracy. # Parameters= **669,381**
7	Conv3D	- Dropout layers added with the previously selected samples of frames.	Train Accuracy: **0.87** Val Accuracy: **0.82**	Model accuracy has increased with less overfitting. Also, we can see improvement in the training & validation loss data as well. # Parameters= **669,381**
8	Conv3D	- 4 convolutional layers (lets add one more convolutional layer to the model)	Train Accuracy: **0.75** Val Accuracy: **0.74**	Adding a layer to the model does not improve the accuracy and performance as well. # Parameters= **697,061**
9	Conv3D	- 3 convolutional layers - Kernel size: (2,2,2)	Train Accuracy: **0.75** Val Accuracy: **0.61**	Changing the kernel size does not improve the model. # Parameters= **619,829**
10	Conv3D	- Kernel size: (3,3,3) - Optimizer: 'SGD' (Stochastic Gradient Descent)	Train Accuracy: **0.34** Val Accuracy: **0.20**	Model performs poor with SGD optimizer. # Parameters= **669,381**
11	Conv3D	- Optimizer: 'Adam' - Learning rate: 0.002	Train Accuracy: **0.84** Val Accuracy: **0.72**	The model with increased learning rate does not show better accuracy than the model with standard learning rate (0.001). # Parameters= **669,381**
12	Conv3D	- Optimizer: 'Adam' - Learning Rate: 0.00095	Train Accuracy: **0.89** Val Accuracy: **0.79**	Accuracy has increased slightly when the learning rate is slightly reduced from the standard one. # Parameters= **669,381**
13	Conv3D	- With batch normalization	Train Accuracy: **0.53** Val Accuracy: **0.29**	Model does not work well with BatchNormalization. # Parameters= **669,605**

14	Conv3D	- Pool Size: (2,2,2) - Image Dimension = 100x100	Train Accuracy: **0.84** Val Accuracy: **0.77**	Overfitting has been reduced and the model performs decently good. # Parameters= **411,333**
15	Conv3D	- Pool Size: (3,3,3)	Train Accuracy: **0.85** Val Accuracy: **0.71**	Model performs good with this pool size but not as much as the previous one. Let's switch to Time Distributed Conv2D + GRU model since there is not much improvement in the Conv3D model. # Parameters= **575,173**
16	Time Distributed + 2D Conv GRU	- Batch size = 20 - Augmentation, Drop out layers=False - # Epochs = 10 - Image Dimension = 100x100 - Activation function =Relu - 3 2D- convolutional layers,2 dense Time distributed layers, 1 SoftMax dense output layer - Optimizer: Adam	Train Accuracy: **0.94** Val Accuracy: **0.75**	The model is overfitting. Let's add augmentation, drop out layers & also let's reduce the image dimensions. # Parameters= **541,285**
17	Time Distributed + 2D Conv GRU	- Augmentation, Drop out layers = True - Image Dimension = 90x90	Train Accuracy: **0.75** Val Accuracy: **0.58**	Model performs poor and so let's increase the number of epochs & batch size with some changes like adding dropout layers and BatchNormalization in the model architecture. However, the number of parameters has been reduced. # Parameters= **463,461**
18	Time Distributed + 2D Conv GRU	- # Epochs = 15 - Batch size = 50	Train Accuracy: **0.81** Val Accuracy: **0.74**	Now the model performs decently good and number of parameters are also reduced. # Parameters= **463,461**
19	Time Distributed + 2D Conv GRU	- With augmentation, dropout layers - 4 2D-convolutional layers, 2 Dense TimeDistributed layers, 1 GRU layer, 1 softmax Dense output layer	Train Accuracy: **0.92** Val Accuracy: **0.86**	This is the best accuracy obtained with lesser number of parameters. Lets also incorporate transfer learning to get better accuracy and results. # Parameters= **65,221**

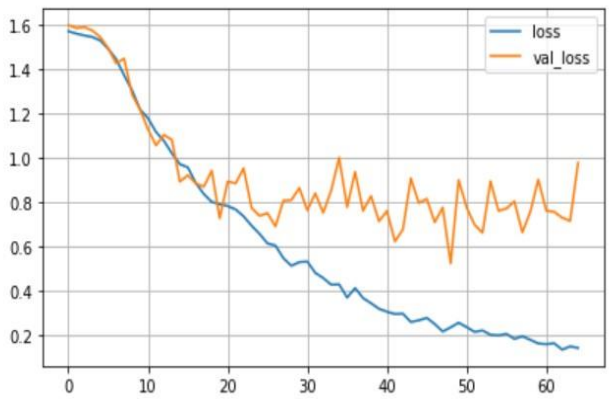
		- Activation function : 'relu', Optimizer : 'Adam' - Image dimension = 100X100, Batch size = 20, # epochs = 65		
20	Transfer learning + GRU	- With trained weights - 1 GRU layer - 1 additional dense layer -Activation function:'relu'	Train Accuracy: **0.90** Val Accuracy: **0.95**	The model with trained weights gives best accuracy but the number of parameters involved are huge. # Parameters in the model= **3,284,389**

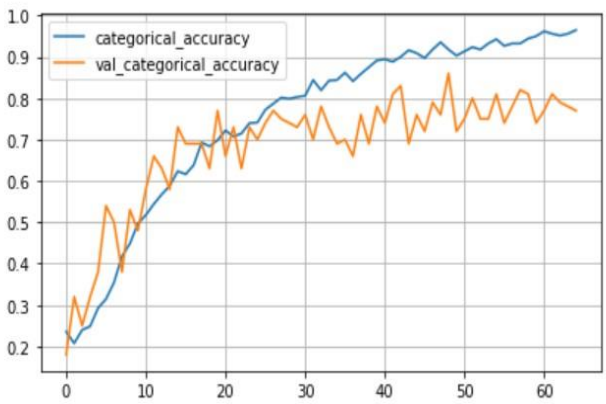
FINAL MODEL:

- **2D convolutional network + GRU model**
- **With augmentation**
- **With dropout layers**
- **4 2D-convolutional layers**
- **2 Dense TimeDistributed layers**
- **1 GRU layer**
- **1 softmax Dense output layer**
- **Activation function**: 'relu'
- **Optimizer**: 'Adam'
- **Image height** = 100
- **Image width** = 100
- **Batch size** = 20
- **Number of epochs** = 65
- **Metrics**: Accuracy

RESULTS:

- Training accuracy: **0.9176**
- Validation accuracy: **0.8600**
- Training loss: **0.2340**
- Validation loss: **0.5227**
- # Parameters: **65,221**





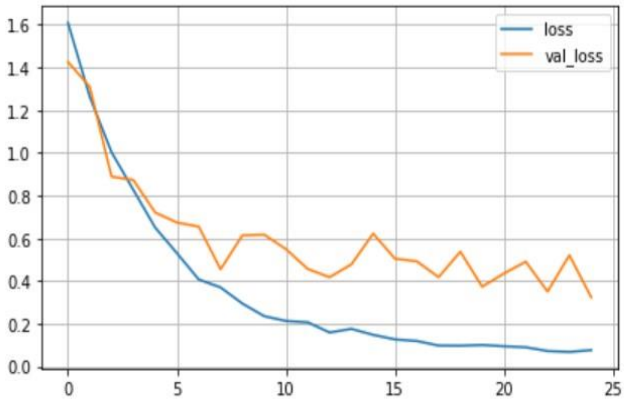
- Among the models that are tried for the given problem statement, MODEL 19 gives decent accuracy and performance with lesser number of parameters compared to other models. Overfitting is controlled to an extent but still there is a scope for improvement in the model.

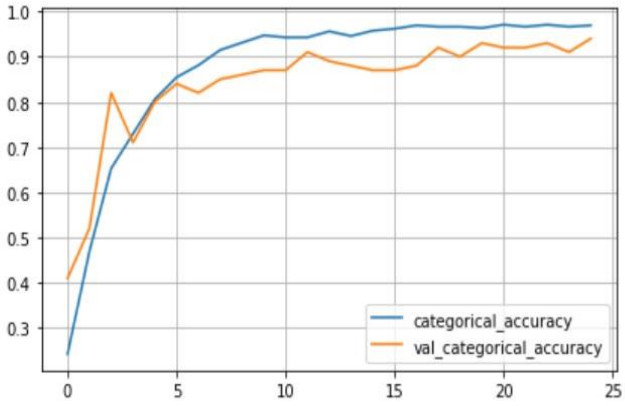
- **Transfer learning:** Using a pre-trained *mobilenet* model with imagenet weights, we tried to identify the initial feature vectors and passed them further to a *GRU* for sequence information before finally passing it to a softmax layer for classification of gestures. We obtained remarkable validation accuracy of 0.94. But however, number of parameters involved in this model are huge. If the required computational efficacies are available, then we can go with this model.

- For transfer learning, we used following architecture:

    - Transfer learning + GRU model
    - With augmentation
    - With dropout layers
    - Activation function : 'relu'
    - Optimizer : 'Adam'
    - Image height = 100
    - Image width = 100
    - Batch size = 20
    - Number of epochs = 25
    - Metrics : Accuracy

- Number of parameters : **3,284,389**
- Training accuracy : **0.9691**
- Validation accuracy : **0.9400**
- Training loss : **0.0753**
- Validation loss : **0.3231**





As we can see, the best accuracy is obtained with the transfer learning model.