GESTURE RECOGNITION GROUP PROJECT

Contributors: Sabyasachi, Sreedhar, Yubaraj

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 recognize 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:

gestures	Corresponding Action
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.

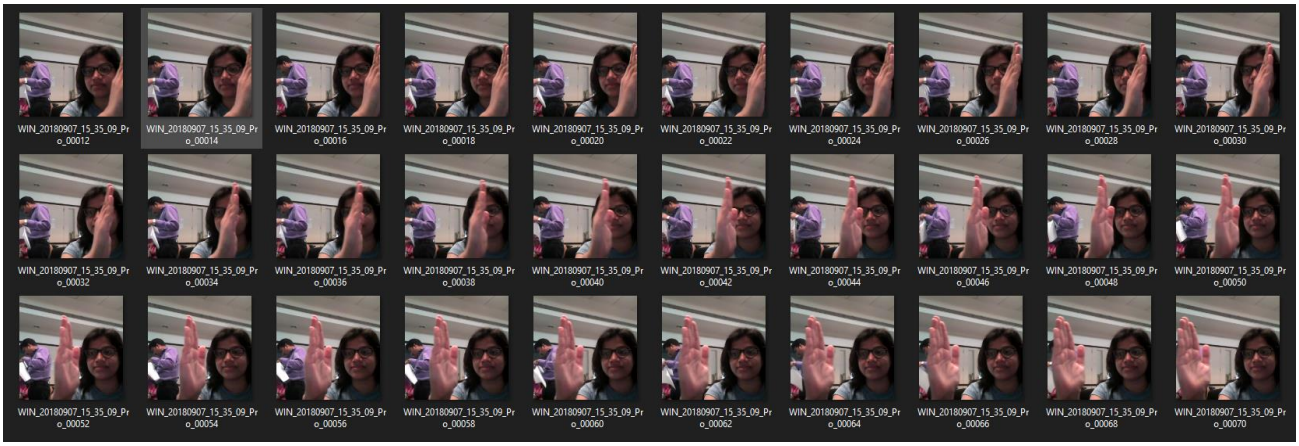
Each video is a sequence of 30 frames (or images).

Objectives

Our task is to train different models 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

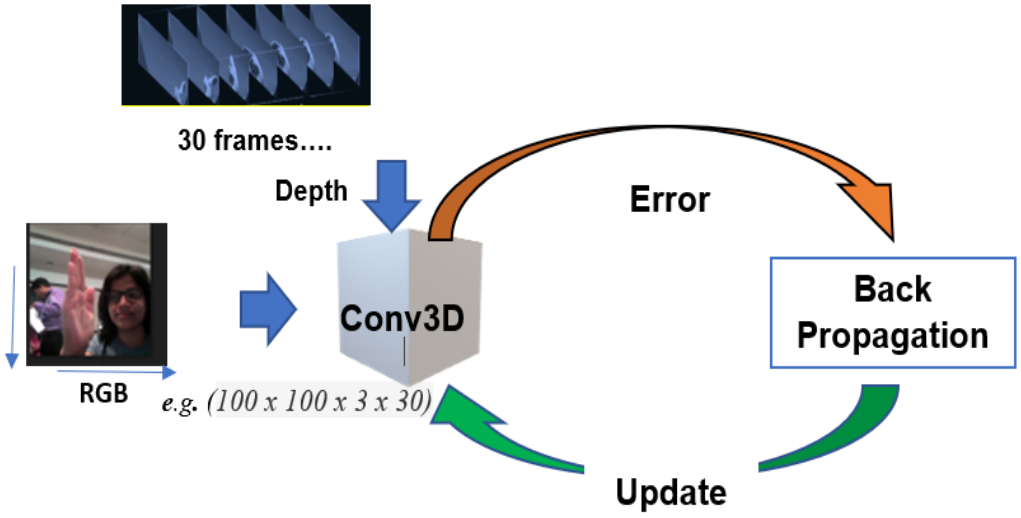
Understanding the Dataset

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.

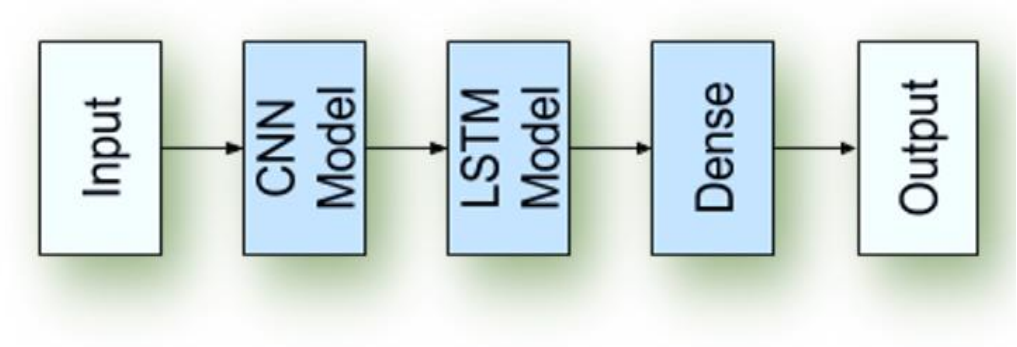


Types Of architecture Used

1. **3D Convolutional Neural Networks (Conv3D)**
   3D convolutions are a natural extension to the 2D convolutions you are already familiar with. Just like in 2D conv, you move the filter in two directions (x and y), in 3D conv, you 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 100 x 100 x 3, for example, the video becomes a 4D tensor of shape 100 x 100 x 3 x 30 which can be written as (100 x 100 x 30) x 3 where 3 is the number of channels. Hence, deriving the analogy from 2D convolutions where a 2D kernel/filter (a square filter) is represented as (f x f) x c where f is filter size and c is the number of channels, a 3D kernel/filter (a 'cubic' filter) is represented as (f x f x f) x c (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 (100 x 100 x 30) tensor



2. **CNN + RNN architecture**
   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).



Data Generator

This is one of the most important part of the code. In the generator, we are going to pre-process the images as we have images of 2 different dimensions (360 x 360 and 120 x 160) as well as create a batch of video frames. The generator should be able to take a batch of videos as input without any error. Steps like cropping, resizing and normalization should be performed successfully.

Data Pre-processing

- Resizing and cropping of the images. This was mainly done to ensure that the NN only recognizes the gestures effectively rather than focusing on the other background noise present in the image.
- Normalization of 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.
- At the later stages for improving the model's accuracy, we have also made use of data augmentation, where we have slightly rotated the pre-processed images of the gestures in order to bring in more data for the model to train on and to make it more generalizable in nature as sometimes the positioning of the hand won't necessarily be within the camera frame always.
- If we would rotate the picture the meaning of the gesture will change completely so rotation wasn't taken into consideration.
  Eg:-

Process/Experiments that we followed for Neural Net development(briefly)

- We have experimented with different models and hyperparameters with various combination of batch size , image size , filter size , padding etc.
- We also made use of Batch Normalization, pooling and dropout layers when our model started to overfit, this could be easily witnessed when our model started giving poor validation accuracy in spite of having good training accuracy.
- Early stopping helped us to put a halt at the training process when the val_loss(validation loss) got saturated or became stagnant .

Observations

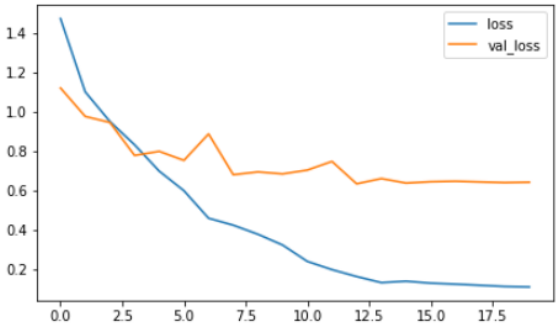
Model	Experiment	Result	Decision + Explanation	Parameters
CONV3D	0	Initial Model	image resolution and number of frames in sequence have more impact on training time than batch_size. We can consider the Batch Size around 15-40 We will use the resoulution in between 160 X 160, 100 X 100 as per the model performance	687,813
	1	Traning Accuracy: 97% Validation Accuracy: 58	Model seem to overfit	1,117,061
	2	Traning Accuracy: 86% Validation Accuracy: 82	Best weights are saved automatically.	3,638,981
	3	Traning Accuracy: 79% Validation Accuracy: 75	The validation loss did not improve from 0.58494	1,762,613
	4	Traning Accuracy: 86% Validation Accuracy: 73	We can see the validation loss did not improve from 0.59194, so early stopping stoped the epoch automatically and just ran 20/25 epochs.	2,556,533
	5	Traning Accuracy: 79% Validation Accuracy: 56	The model seem to overfit Let us try to reduce the parameters and see the performance in the coming models.	2,556,533
	6	Traning Accuracy: 82% Validation Accuracy: 71	The model seem to overfit	2,556,533
	7	Traning Accuracy: 86% Validation Accuracy: 73	Epoch 00017: val_loss did not improve from 0.80514 hence early stopping	504,709
CNN+LSTM	**8**	**Traning Accuracy: 92% Validation Accuracy: 75%**	**Epoch 00020: val_loss improved from 0.60878 to 0.60324, saving model to model_init_2021-02-0717_05_50.861517/model-00020-0.21173-0.92157-0.60324-0.75000.h5**	**1,657,445**
Augmentation	9	Traning Accuracy: 79% Validation Accuracy: 77%	The Model 9 training accuracy has reduced from 92% to 79% as compare with Model-8	3,638,981
	10	Traning Accuracy: 80% Validation Accuracy: 68%	Categorical and validation accuracy are 80% and 68% respectively in the last epoch.	504,709
CNN+LSTM+ Augmentation	11	Traning Accuracy: 97% Validation Accuracy: 75%	The augmentation seems to be not perfoming as we can see the model is overfitting. Lets try MobileNet for Model building(Transfer Leaning) as it is light weight.	2,573,541
Transfer Learning	12	Traning Accuracy: 98% Validation Accuracy: 96%	The Model seems to be working with high accuracy.	3,692,869

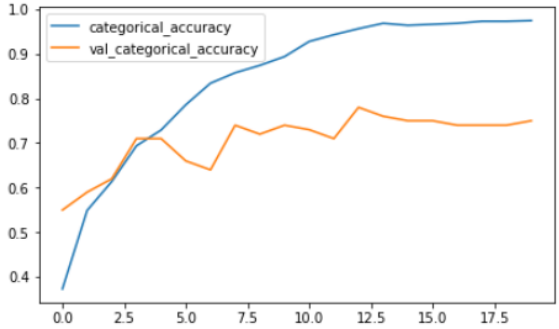
CPU Stats before starting to train Models.

```
Sun Feb  7 20:50:07 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 390.30                 Driver Version: 390.30                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   55C    P0    58W / 149W |  10961MiB / 11441MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0      2161      C   /mnt/disks/user/anaconda3/bin/python       10947MiB |
+-----------------------------------------------------------------------------+
```

- With increase in number of parameters the model took more time to Train.
- If we increased the batch size by huge margin the training time reduces but the model accuracy reduced.
- So we concluded that if we want our model to train in short time we can take huge batch size and if we want the model to have better accuracy we need smaller Batch size.
- Data augmentation didn't help us much. E.g.-





- CNN+LSTM based model with had better performance than Conv3D. As per our understanding, this is something which depends on the kind of data we used, the architecture we developed and the hyper-parameters we chose.

CNN+LSTM	**8**	**Traning Accuracy: 92% Validation Accuracy: 75%**	**Epoch 00020: val_loss improved from 0.60878 to 0.60324, saving model to model_init_2021-02-0717_05_50.861517/model-00020-0.21173-0.92157-0.60324-0.75000.h5**	**1,657,445**

- The best result was obtained by the transfer learning model with MobileNet .

Transfer Learning	12	Traning Accuracy: 98% Validation Accuracy: 96%	The Model seems to be working with high accuracy.	3,692,869

- After doing all the experiments, we finalized Model 8– CNN+LSTM, which performed well.
  As per below inferences.
  - (Training Accuracy: 92%, Validation Accuracy: 75%)
  - Number of Parameters (1,657,445) less according to other models' performance
  - Learning rate gradually decreasing after some Epochs and Val loss improved.

How this can be improved.

- The model can be improved if we use Transfer learning.
- Using a GRU Model in place of LSTM model as it is faster. (the tradeoff is the accuracy)
- Deeper Understanding of the data.
- Use of the optimum hyperparameters.
