## **Deep Learning Course Project- Gesture Recognition**

# 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.

* 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.

# Objective

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.

# Two types of architectures suggested for analysing videos using deep learning:

1. **3D Convolutional Neural Networks (Conv3D)**
2. **CNN + RNN architecture**

# 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.

# NN Architecture development and training

* Experimented with different model configurations and hyper-parameters and various iterations and combinations of batch sizes, image dimensions, filter sizes, padding and stride length were experimented with. We also played around with different learning rates and *ReduceLROnPlateau* was used to decrease the learning rate if the monitored metrics (*val\_loss*) remains unchanged in between epochs.
* We experimented with *SGD()* and *Adam()* optimizers but went forward with *Adam()* as it lead to improvement in model’s accuracy by rectifying high variance in the model’s parameters. We were unsupportive of experimenting with *Adagrad()* and *Adadelta()* due to the limited computational capacity as these take a lot of time to converge because of their dynamic learning rate functionalities.
* 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 inspite of having good training accuracy.
* *Early stopping* was used to put a halt at the training process when the *val\_loss* would start to saturate / model’s performance would stop improving.

# Observations

* It was observed that as the Number of trainable parameters increase, the model takes much more time for training.
* **Batch size ∝ GPU memory / available compute.** A large batch size can throw *GPU Out of memory error,* and thus here we had to play around with the batch size till we were able to arrive at an optimal value of the batch size which our GPU could support.
* Increasing the batch size greatly reduces the training time but this also has a negative impact on the model accuracy. This made us realise that there is always a trade-off here on basis of priority -> If we want our model to be ready in a shorter time span, choose larger batch size else you should choose lower batch size if you want your model to be more accurate.
* *Data Augmentation* and *Early stopping* greatly helped in overcoming the problem of overfitting which our initial version of model was facing.
* *CNN+LSTM* based model with *GRU* cells 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.
* *Transfer learning* **boosted** the overall accuracy of the model. We made use of the [*MobileNet*](https://arxiv.org/abs/1704.04861) Architecture due to it’s light weight design and high speed performance coupled with low maintenance as compared to other well-known architectures like VGG16, AlexNet, GoogleNet etc.
* For detailed information on the Observations and Inference, please refer Table 1.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| MODEL TYPE | MODEL DETAILS | EXPERIMENT NO. | RESULT | DECISION + EXPLANATION | PARAMETERS |
| Conv3D | **Conv3D**  **-Batch size = 30/20/15**  **-Epochs =2**  **-Number of frames =30/16**  **- Image size = (160,160)/(100,100)**  **-Kernel size = (2,2,2)**  **-Activation = relu**  **-Dropout = 0.25** | **1**  **(Sample Model A,B,C)** | **OOM Error:**  **We had hit the limit on memory resources with image resolution of 160x160 with 30 frames and batch\_size of 40** | **Reduce the batch size and Reduce the number of neurons in Dense layer** | **1,736,389** |
| **Conv3D**  **-Batch size = 40**  **-Epochs =15**  **-Number of frames =20**  **- Image size = (160,160)**  **-Kernel size = (3,3,3)**  **-Activation = relu** | **2**  **(Model-1)** | **Training Accuracy : 0.96**  **Validation Accuracy : 0.19** | **Overfitting!!**  **Let’s add some Dropout Layers** | **1,117,061** |
| **Conv3D**  **-Batch size = 20**  **-Epochs =25**  **-Number of frames =20**  **- Image size = (160,160)**  **-Kernel size = (3,3,3)**  **-Activation = relu**  **-Dropout = 0.5** | **3**  **(Model-2)** | **Training Accuracy : 0.79**  **Validation Accuracy : 0.20** | **Val\_loss didn’t improve from 3.38 so early stopping stop the training process. Let’s lower the learning rate to 0.0002.**  **Reduce the filter size and image resolution** | **3,638,981** |
| **Conv3D**  **-Batch size = 30**  **-Epochs =25**  **-Number of frames =16**  **- Image size = (120,120)**  **-Kernel size = (2,2,2)**  **-Activation = relu**  **-Dropout = 0.5** | **4**  **(Model-3)** | **Training Accuracy : 0.68**  **Validation Accuracy : 0.26** | **Overfitting has reduced but accuracy hasn't improved.We  able to reduce the parameter size by half the earlier model.**  ***Let's trying adding more layers!*** | **1,762,613** |
| **Conv3D**  **-Batch size = 20**  **-Epochs =25**  **-Number of frames =16**  **- Image size = (120,120)**  **-Kernel size = (3,3,3)**  **-Activation = relu**  **-Dropout = 0.5** | **5**  **(Model-4)** | **Training Accuracy : 0.73**  **Validation Accuracy : 0.33** | **Don’t see much performance improvement with more no. of layers*.***  ***Let's try adding dropouts at the convolution layers.*** | **2,556,533** |
| **Conv3D**  **-Batch size = 20**  **-Epochs =15**  **-Number of frames =16**  **- Image size = (120,120)**  **-Kernel size = (3,3,3)**  **-Activation = relu**  **-Dropout = 0.25** | **6**  **(Model-5)** | **Training Accuracy : 0.83**  **Validation Accuracy : 0.25** | **Overfitting Increase, adding dropouts has further reduced validation accuracy. Let's try to reduce the parameters** | **2,556,533** |
| **Conv3D**  **-Batch size = 20**  **-Epochs =20**  **-Number of frames =16**  **- Image size = (100,100)**  **-Activation = relu**  **-Dropout = 0.25**  **dense\_neurons=128** | **7**  **(Model-6)** | **Training Accuracy : 0.76 Validation Accuracy : 0.27** | **Overfitting reduced, but validation accuracy low. Let's try to reduce the parameters.** | **696,645** |
| **Conv3D**  **-Batch size = 20**  **-Epochs =25**  **-Number of frames =16**  **- Image size = (120,120)**  **-Activation = relu**  **-Dropout = 0.25**  **dense\_neurons=64** | **8**  **(Model-7)** | **Training Accuracy : 0.75 Validation Accuracy : 0.27** | **Accuracy remains below same. Let’s switch to CNN+LSTM.** | **504,709** |
| CNN+LSTM | **Conv2D**  **-Batch size = 20**  **-Epochs =20**  **-Number of frames =18**  **- Image size = (120,120)**  **-Kernel size = (3,3,3)**  **-Activation = relu**  **-Dropout = 0.25**  **lstm\_cells=128**  **dense\_neurons=128** | **9**  **(Model-8)** | **Training Accuracy : 0.87**  **Validation Accuracy : 0.31** | **CNN - LSTM model Performance Improved.**  **Applying Data Augmentation will Reduce Overfitting Problems.** | **1,657,445** |
| Conv3D |  | **Let's apply some Data Augmentation techniques & check the Model Performance** | | | |
| **Conv3D+Augm.**  **-Batch size = 20**  **-Epochs =20**  **-Number of frames =20**  **- Image size = (160,160)**  **-Kernel size = (3,3,3)**  **-Activation = relu**  **-Dropout = 0.5**  **dense\_neurons=256** | **10**  **(Model-9)** | **Training Accuracy : 0.72 Validation Accuracy : 0.74** | **Performance Increased.**  **Using Data Augmentation, Overfitting Issue Resolved!** | **3,638,981** |
| **Conv3D+Augm.**  **-Batch size = 30**  **-Epochs =25**  **-Number of frames =16**  **- Image size = (120,120)**  **-Kernel size = (2,2,2)**  **-Activation = relu**  **-Dropout = 0.5** | **11**  **(Model-10)** | **Training Accuracy : 0.68 Validation Accuracy : 0.40** | **(2, 2, 2) Filter & 120 x 120 image resolution. Increase epoch count. Network is generalizing well.**  **Parameters Reduced but Performance too Reduced.** | **1,762,613** |
| **Conv3D+Augm.**  **-Batch size = 20**  **-Epochs =25**  **-Number of frames =16**  **- Image size = (120,120)**  **-Kernel size = (3,3,3)**  **-Activation = relu**  **-Dropout = 0.5** | **12**  **(Model-11)** | **Training Accuracy : 0.71 Validation Accuracy : 0.75**  ***(Epoc 23/25 best Accuracy)*** | **Adding more layers.**  **Increased Performance Again with “Least Parameters” & Data Augmentation Solves Overfitting Issue.**  **Best Model Overall.** | **2,556,533** |
| **Conv3D+Augm**  **-Batch size = 20**  **-Epochs =25**  **-Number of frames =16**  **- Image size = (120,120)**  **-Kernel size = (3,3,3)**  **-Activation = relu**  **-Dropout = 0.25**  **dense\_neurons=256** | **13**  **(Model-12)** | **Training Accuracy : 0.63 Validation Accuracy : 0.33**  ***(Epoc 23/25 best Accuracy)*** | **On Adding More layers of dropout layers, decreases Performance Again.**  **Overfitting Badly.**  **Let’s reduce the network parameters.** | **2,556,533** |
| **Conv3D+Augm.**  **-Batch size = 20**  **-Epochs =25**  **-Number of frames =16**  **- Image size = (100,100)**  **-Kernel size = (3,3,3)**  **-Activation = relu**  **-Dropout = 0.25**  **dense\_neurons=128** | **14**  **(Model-13)** | **Training Accuracy : 0.78 Validation Accuracy : 0.67** | **After reducing network parameters, model’s performance is quite good.**  **Overfitting Resolved.** | **696,645** |
| **Conv3D+Augm.**  **-Batch size = 20**  **-Epochs =25**  **-Number of frames =16**  **- Image size = (120,120)**  **-Kernel size = (3,3,3)**  **-Activation = relu**  **-Dropout = 0.25**  **dense\_neurons=64** | **15**  **(Model-14)** | **Training Accuracy : 0.73 Validation Accuracy : 0.74**  ***(Epoc 23/25 best Accuracy)*** | **- Least Parameters.**  **- Performance Increased.**  **- No Overfitting.**  **- Validation Accuracy Improved.**  **- Best Model.** | **504,709** |
| CNN LSTM with GRU | **Conv2D**  **-Batch size = 20**  **-Epochs =20**  **-Number of frames =18**  **- Image size = (120,120)**  **-Activation = relu**  **-Dropout = 0.25**  **lstm\_cells=128**  **dense\_neurons=128** | **16**  **(Model-15)** | **Training Accuracy : 0.91 Validation Accuracy : 0.76**  ***(Epoc 19/20 best Accuracy)*** | **Overfitting is considerably high, not much improvement.** | **2,573,925** |
| Transfer Learning (Optional) | **MobileNet**  **-Batch size = 5**  **-Epochs =20**  **-Number of frames =16**  **- Image size = (120,120)**  **-Activation = relu**  **-Dropout = 0.25**  **lstm\_cells=128**  **dense\_neurons=128** | **17**  **(Model-16)** | **Training Accuracy : 0.98 Validation Accuracy : 0.84**  ***(Epoc 14/20 best Accuracy)*** | ***Performance Improved by using Transfer learning with reduced Batch Size.***  ***Overfitting Resolved.*** | **3,840,453** |
| Transfer Learning with GRU & Training Weights  (Optional) | **MobileNet**  **-Batch size = 5**  **-Epochs =20**  **-Number of frames =16**  **- Image size = (120,120)**  **-Activation = relu**  **-Dropout = 0.25**  **gru\_cells=128**  **dense\_neurons=128** | **18**  **(Model-17)** | **Training Accuracy : 0.99 Validation Accuracy : 0.96**  ***(Epoc 16/20 best Accuracy)*** | **Awesome result !!!**  **(MODEL FINALIZED)** | **3,693,253** |

**After doing all the experiments, we FINALIZED Model 17– Conv3D+GRU+TRANSFER LEARNING, which performed Extremely well.**

**Reason: AWESOME PERFORMANCE (Training Accuracy : 99%, Validation Accuracy : 96%)**

**- Performance Increased. No Overfitting. Validation Accuracy Improved Drastically.**

**- Number of Parameters(3,693,253)**

**Table 1: Observations and Results for numerous tested NN architectures**

**BEST MODEL**

**The specifications of the best model (Conv3D+GRU+TRANSFER LEARNING) is:**

**1. Training accuracy: 99%**

**2. Validation accuracy: 96%**

**3. Number of Frames: 16**

**4. Image Size: 120, 120**

**5. Dropout = 0.25**

**6. Batch Size: 5**

**7. Number of Epochs: 20**

**8. Total Parameters: 3,693K (approximately)**

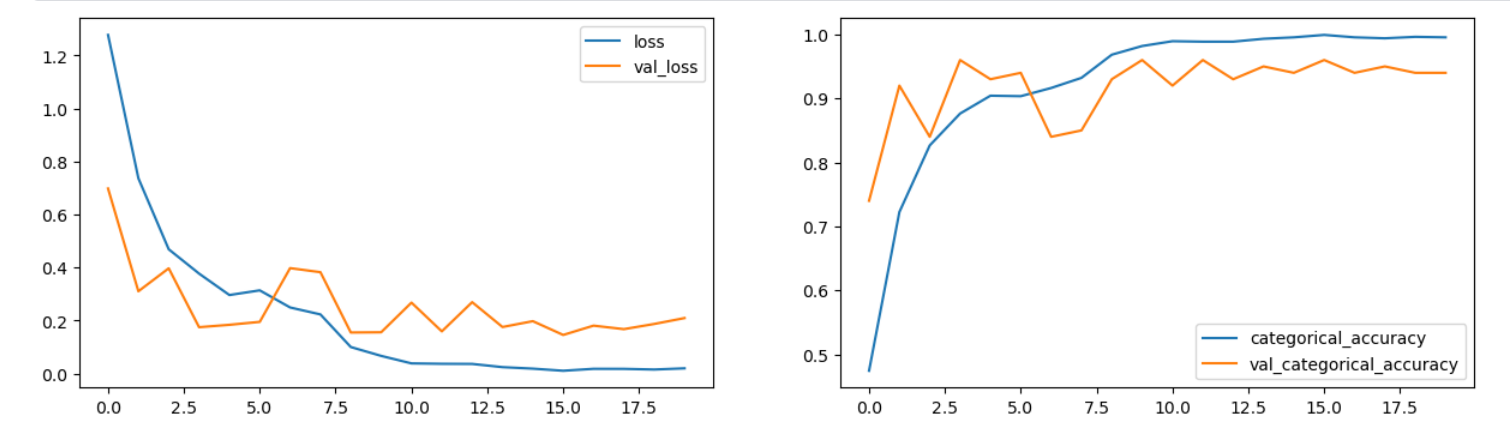
**9. Activation = relu**

**10. gru\_cells=128**

**11. dense\_neurons=128**

**12. Transfer Learning : MobileNet**

A plot showing the variation of loss and accuracy across epochs is shown below.



# Further suggestions for improvement:

* **Using Transfer Learning**: Using a pre-trained *ResNet50/ResNet152/Inception V3* 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 other pre-trained models couldn’t be tested due to lack of time and disk space in the nimblebox.ai platform.)
* **Using GRU:** A *GRU* model in place 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. However, its effect on 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 recorded in different backgrounds, lightings, persons and different cameras where used. Further exploration on the available images could give some more information about them and bring more diversity in the dataset. This added information can be exploited in favour inside the generator function adding more stability and accuracy to model.
* **Tuning hyperparameters:** Experimenting with other combinations of hyperparameters like, activation functions (*ReLU, Leaky ReLU, mish, tanh, sigmoid*), other optimizers like *Adagrad()* and *Adadelta()* can further help develop better and more accurate models. Experimenting with other combinations of hyperparameters like the *filter size, paddings, stride\_length, batch\_normalization, dropouts* etc. can further help improve performance.