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

* **Abhishek Kumar Tiwary**

# 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.

# 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.

**A picture containing photo, many, various, sitting

Description automatically generated**

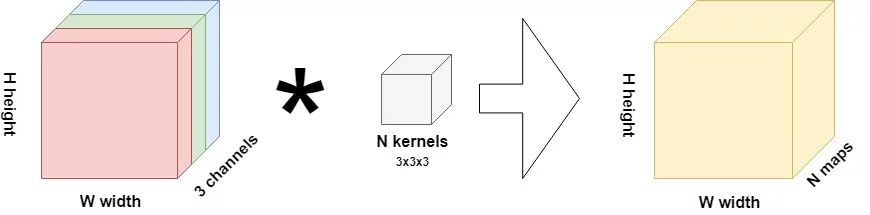
# Objective

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)**

Three-dimensional convolutional layers, things are different - but not too different. Instead of three dimensions in the input image (the two image dimensions and the *channels* dimension, you'll have four: the two image dimensions, the time/height dimension, and the channels dimension). As such, the feature map is also three-dimensional. This means that the filters move in three dimensions instead of two: not only from left to right and from the top to the bottom, but also forward and backward. Three-dimensional convolutional layers will therefore be more expensive in terms of the required computational resources, but allow you to retrieve much richer insights..

****

**Figure 1: A simple representation of working of a 3D-CNN**

1. **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).

A close up of a sign

Description automatically generated

**Figure 2: A simple representation of an ensembled CNN+LSTM 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.
* I have experimented with *Adam* optimizer as it lead to improvement in model’s accuracy by rectifying high variance in the model’s parameters.
* I 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 ☺.
* I was not getting good results with Batch Normalization and most the models were overfitting. When I have removed Batch Normalization, Then I got good results for the validation data.
* *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.

**Note: - I have used both google colab and local server in lab for model train and execution**

# 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 I had to play around with the batch size till I was able to arrive at an optimal value of the batch size which our GPU could support
* With Google Colab the batch size of 64 was giving OOM error, while running on lab server I was able to run with 128 batch size as well. I have trained most of the models with 64 batch size.
* *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 for the train data.* But , I have got good validation results with 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.
* For detailed information on the Observations and Inference, please refer Table 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model Type** | **Model** | **Results** | **Explanation/Comments** | **Parameters** |
| **Conv3D** | 0 | OOM Error | When ran with batch size = 64 and Frames = 30  on google colab, Got the OOM error | 1,117,061 |
| **Conv3D** | 1 | Train Acc - 72% Val Acc - 32% | - frames\_to\_sample=20, batch\_size=64, num\_epochs=25 - dense\_neurons=256, dropout=0.5 - image\_height=160, image\_width=160  -- Overfitting  -- early stopping due to val-loss is not coming down -- It seems network Architecture has some issues | 3,638,981 |
| **Conv3D** | 2 | Train Acc - 49% Val Acc - 56% | - frames\_to\_sample=20, batch\_size=32, num\_epochs=10 - dense\_neurons=64, dropout=0.25 - image\_height=120, image\_width=120  --Removed the Batch Normalization Layer -- Clearly Underfit but not overfit -- We can train with more epochs and extra parameters to see the behavior of same network | 697,797 |
| **Conv3D** | 3 | Train Acc - 94.48% Val Acc - 93% | -frames\_to\_sample=16, batch\_size=32, num\_epochs=25 - filtersize=(2,2,2), dense\_neurons=256, dropout=0.5 - image\_height=120, image\_width=120  -- Drastic improvement after removing batch normalization and running for 25 iterations -- Model size is 20MB, which is bit heavy for prediction.  Increase of Dropout value has also helped in handling overfitting.  -- Let's decrease the number of parameters and build a lighter model with less parameters | 1,759,605 |
| **Conv3D** | 4 | Train Acc - 90% Val Acc - 96% | - frames\_to\_sample=16, batch\_size=64, num\_epochs=50 - filtersize=(2,2,2), dense\_neurons=64, dropout=0.5 - image\_height=128, image\_width=128  -- Best Model in terms of validation results. -- Trained it for 50 epochs to get best validation result. -- I had also ran the same model for 25 epochs and found out that val\_loss is going down -- Model size is 7MB which is three times lesser than the earlier best model. -- No of parameters are also three times lesser than the earlier Model - 3 | 615,477 |
| **Conv2D + LSTM** | 5 | Train Acc -99.32% Val Acc - 86% | -frames\_to\_sample=20, batch\_size=64, num\_epochs=25 - lstm\_cells=128, dense\_neurons=128, dropout=0.25 - image\_height=120, image\_width=120  -- There are signs of Overfitting  -- Although Model 5 gave the best results in terms of train data till now. -- Let's increase the dropout value to 0.5 to handle overfitting in next model | 1,655,461 |
| **Conv2D + LSTM** | 6 | Train Acc - 99.77% Val Acc - 87% | -frames\_to\_sample=20, batch\_size=64, num\_epochs=25 - lstm\_cells=128, dense\_neurons=128, dropout=0.5 - image\_height=120, image\_width=120  -- Overfitting again -- Got the best results in terms of train data. -- We need to tweak the architecture little bit and then train again to handle overfitting.  --But due lack of time and resource to train, I am leaving this up to this point only. | 1,655,461 |
| **Conv2D + GRU** | 7 | Train Acc - 99.17% Val Acc - 87% | frames\_to\_sample=20, batch\_size=64, num\_epochs=25 lstm\_cells=128, dense\_neurons=128, dropout=0.5 image\_height=120, image\_width=120  Overfitting again Got the best results in terms of train data. -- We need to tweak the architecture little bit and then train again to handle overfitting.  --But due lack of time and resource to train, I am leaving this up to this point only. I am leaving this up to this point only. -- GRU has given similar results to LSTM as well. | 2,572,965 |

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

# Best Results and Observations:

After doing all the experiments, we finalized **Model-4(conv\_3d4)** - CNN3D, which performed well.

**Results and Observations:**

* **Max Training Accuracy : 90%, Max Validation Accuracy : 96%**
* **Number of Trainable Parameters for the best Model 4 is 615477 which is far less than the other models of CNN3D and CNN + LSTM models**
* **The best weights of CNN-LSTM: model-00042-0.33726-0.86802-0.12430-0.96000.h5 (~7 MB) which has minimum val\_loss**
* **The CNN + LSTM models are giving very good result on train data, but they are little overfitting. We need to fine tune them. This will take some more time and resource. I think that can be considered as out of scope of this assignment**

# 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.)
* **Tuning CNN + RNN Model:**  CNN + RNN is showing goos result signs, But I need to tune the model further to get better validation result.
* **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, dropouts* etc. can further help improve performance.

**\*\* END OF THE DOCUMENT \***