## **Neural Network Project - Gesture Recognition**

**Problem Statement:**

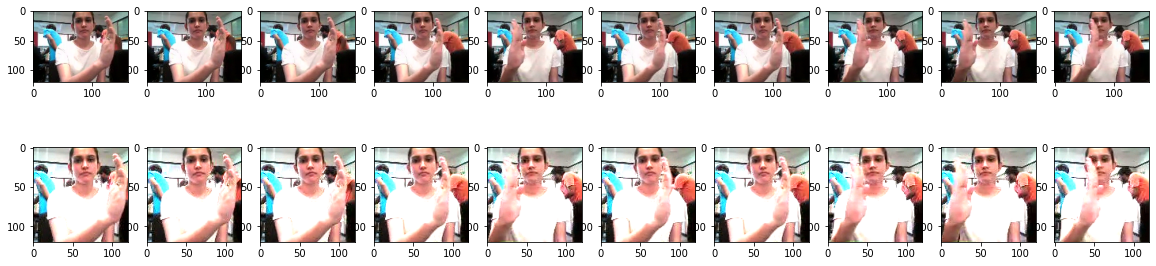
We as a Data Scientist needs to develop a cool feature in the smart-TV that can recognise five different gestures performed by the user which will help users control the TV without using a remote.

* 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.

**Dataset:** <https://drive.google.com/drive/folders/1o6sZUe51ODZPpMb5ydzbTfR6zju26oc?usp=sharing>

**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.

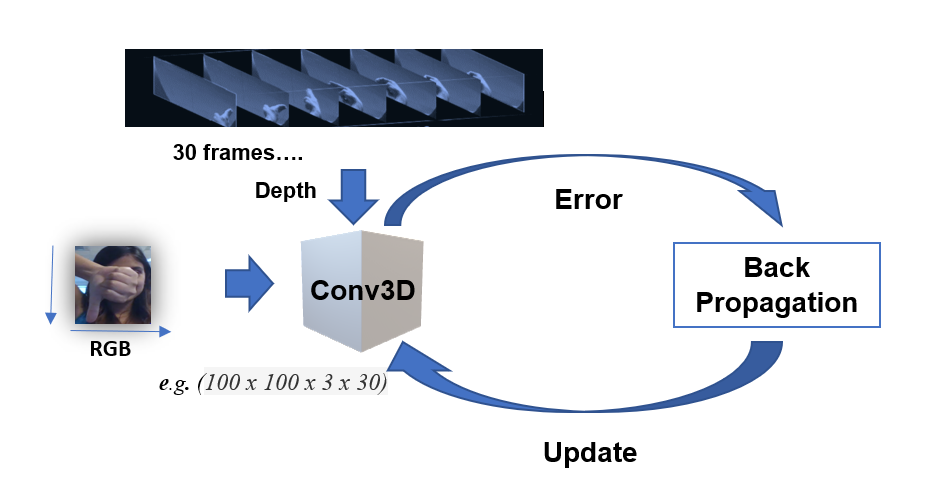


**Objective:**

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)**



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

1. **CNN + RNN architecture**

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.
* 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 ( NVIDIA Tesla K80 GPU with 12GB memory provided by nimblebox.ai platform.)
* 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 below Table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Exp. #** | **Model** | **Hyper- parameters** | **Result** | **Decision + Explanation** |
| **1** | **Conv3D** | **Batch size = 128,**  **Ablation = 20, Augmentation = False,**  **LR = 0.01,**  **Seq Length = 10,**  **Epoch = 20, Dim = 120x120** | **Train Accuracy: 0.15,**  **Validation Accuracy: 0.15** | **The Model is not learning anything throughout the epochs, the loss is not decreasing. Reducing the batch size further.** |
| **2** | **Conv3D** | **Batch size = 32** | **Train Accuracy: 0.15,**  **Validation Accuracy: 0.20** | **No improvement in the model, lets add more layers to the model so that it can learn from data.** |
| **3** | **Conv3D** |  | **Negative Dimension Error.** | **The new CNN kernel sizes are not compatible with the output of previous layers. Let’s reduce the kernel size of new layers.** |
| **4** | **Conv3D** |  | **Train Accuracy: 0.20,**  **Validation Accuracy: 0.20** | **Still there is no improvement in the model. Let’s add Batch normalization layers after every CNN and dense layers.** |
| **5** | **Conv3D** |  | **Train Accuracy: 0.9062,**  **Validation Accuracy: 0.2708** | **Model is able to over-fit on less data (Ablation data set), Let’s Training on full data and increasing epochs to 50.** |
| **6** | **Conv3D** | **Ablation = None, Epoch = 15** | **Train Accuracy: 0.9190,**  **Validation Accuracy: 0.70** | **Mode is having over-fitting as there is huge gap between training and validation accuracies. Let’s add some dropouts that the model can be generalized.** |
| **7** | **Conv3D** | **Dropout = 0.2** | **Train Accuracy: 0.9896,**  **Validation Accuracy: 0.7734** | **There is a bit of increase in the model validation accuracy and training accuracy also. Lets increase the drop out values from**  **0.2 to 0.5** |
| **8** | **Conv3D** | **Dropout = 0.5** | **Train Accuracy: 0.9777,**  **Validation Accuracy: 0.5391** | **After increase the dropout the model validation score further reduced and the model is over-fitted. Let’s use 0.2 only remove a CNN layer to reduce the complexity**  **of the model.** |
| **9** | **Conv3D** | **Dropout = 0.2** | **Train Accuracy: 1.00,**  **Validation Accuracy: 0.77** | **Still the model is over-fitting. Let’s use a Global Average Pooling instead of Flatten Layer.** |
| **10** | **Conv3D** |  | **Train Accuracy: 0.9509,**  **Validation Accuracy: 0. 9062** | **The model is wonderful and the training and validation scores are good. The model has 710,533 trainable parameters. Let’s try architectures too.** |
| **11** | **Time Distributed**  **+ GRU** |  | **Train Accuracy: 0.9554,**  **Validation Accuracy: 0. 8203** | **The model is working quite well on validation dataset with less trainable parameters(98,885), Lets add some drop outs after each layer, so that both train and**  **validation accuracies will be closure.** |
| **12** | **Time Distributed**  **+ GRU** | **Drop out = 0.2** | **Train Accuracy: 0.8720,**  **Validation Accuracy: 0.6016** | **The model accuracy further deteriorated; Let’s replace GRU with a plain Dense Layer Network and some Global Avg Pooling.** |
| **13** | **Time Distributed**  **+ Dense** |  | **Train Accuracy: 0.8780,**  **Validation Accuracy: 0.8750** | **This is good model with training and validation accuracies with number of params 128,517. Let’s use different architecture of model with time distributed and**  **ConvLSTM2D.** |
| **14** | **Time Distributed**  **+**  **ConvLSTM 2D** |  | **Train Accuracy: 0.9673,**  **Validation Accuracy: 0.9375** | **This is the best model so far we can get. The validation accuracy is good and the numbers of parameters are 13,589. The model size is also so small 226KB.** |

# Conclusion:

The Model built with Time distributed Conv2D and ConvLSTM2D (Experiment #14) gave better results compared to all the other models and also the model has very least number of parameters compared to other models.

**Compiled By:**

**Alok Kumar (Facilitator)**

**Pradip Tiwari (Team)**