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

**Pratik Kumar**

**DL Track Dsc23**

**Problem Dataset**

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 recognise 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 - like what the smart TV will use.

**Overvieew**

The 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. Final model's performance will be tested on the 'test' set.

Architecture

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 *84 x 84 x 3*, for example, the video becomes a 4D tensor of shape *84 x 84 x 3 x 30* which can be written as *(84 x 84 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 *(84 x 84 x 30)* tensor.

**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.
* Normalizationof 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.



**CAUTION: It was taken into consideration that we don’t rotate images to a greater extent as this would change the meaning of the gestures completely☺ !!**

* Development and Testing:
* 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. I also tried with different learning rates and *ReduceLROnPlateau* was used to decrease the learning rate.
* I also 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 also made use of *Batch Normalization*, *dropout* *layers*.
* Finally with decreasing the batch size to 20 helped me to get out of the error ans started training

**Findings**

* 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.

|  |  |  |  |
| --- | --- | --- | --- |
| **Experiment Number** | **Model** | **Result** | **Decision + Explanation** |
| **1** | **Conv3D** | **Throws Generator error** | **Crop the images correctly, try to overfit on less amount of data** |
| **2** | **Conv3D** | **Model not trainable due to no memory available error** | **Reduce the size of the image/Reduce the number of batch to 30 from 40** |
| **3** | **Conv3D** | **Model not trainable due to no memory available error** | **Reduce the number of batch to 20 from 30** |
|  |  |  |  |
|  |  |  |  |
| **Final Model** | **……………….** | **………….** | **…………………** |