Deep Learning Gesture Recognition Case Study

In this group project, we are going to build a 3D Conv model that will be able to predict the 5 gestures correctly.

Submitted By:

1. Gursewak Singh
2. Bharat Bhushan
3. Himanshu Yadav

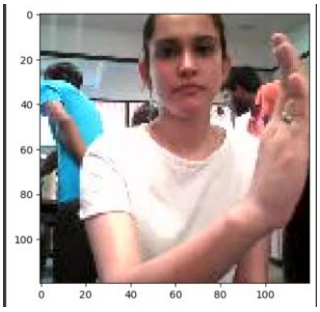
Problem Statement

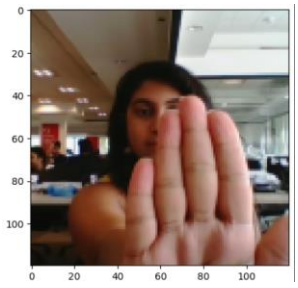
As a data scientist at a home electronics company which manufactures state of the art smart televisions. We want to develop a cool feature in the smart-TV that can recognise five different gestures performed by the user which will help users control the TV without using a remote.

1. Thumbs up : Increase the volume.
2. Thumbs down : Decrease the volume.
3. Left swipe : 'Jump' backwards 10 seconds.
4. Right swipe : 'Jump' forward 10 seconds.
5. 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.





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.

Commonly Used Architecture

1. **Convolutions + RNN**
   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).

2. **3D Convolutional Network, or 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 100x100x3, for example, the video becomes a 4-D tensor of shape 100x100x3x30 which can be written as (100x100x30)x3 where 3 is the number of channels. Hence, deriving the analogy from 2-D convolutions where a 2-D kernel/filter (a square filter) is represented as (fxf)xc where f is filter size and c is the number of channels, a 3-D kernel/filter (a 'cubic' filter) is represented as (fxfxf)xc (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 (100x100x30) tensor.

Data Generator

This part of the code is really important. We have two different sizes of photos (360 x 360 and 120 x 160). We also need to put these photos together to make a video. The code we're writing should be able to do all of this without any mistakes. It's like making sure the photos are the right size and look good. We want to make sure everything works smoothly when we put all these photos in batches to our model.

Data Pre-processing

Before we feed the data (batches of videos) into our model, we need to do the following:

1. ***Normalization* of the images.** Normalizing the RGB values of an image we can get rid of distortions due to shadows and illumination.

2. ***Resizing* and *cropping* of the images.** We need this to help our model to understand the hand gestures better and not get confused by the noise in the background. We want the model to pay attention only to the hand movements and not get distracted by anything else.

NN Architecture development and training

- we tested both the SGD() and Adam() optimizers, but we ultimately chose to proceed with Adam() as it led to an improvement in the model's accuracy by addressing the high variance in the model's parameters.
- We applied early stopping to our training process, which allowed us to halt the training when the validation loss reached a saturation point or when the model's performance ceased to show improvement.
- we engaged in a series of experiments that involved testing various model configurations and tweaking hyper-parameters. These experiments invlves numerous iterations, where we explored different combinations of batch sizes, image dimensions, filter sizes, padding, and stride length. Additionally, we delved into the optimization of learning rates and implemented the "ReduceLROnPlateau" technique to dynamically reduce the learning rate if the monitored metrics, specifically "val_loss," remained unchanged over successive epochs.
- Additionally, we incorporated Batch Normalization, pooling layers, and dropout layers into our model when we observed signs of overfitting. This was evident when our model achieved high training accuracy but began to exhibit poor validation accuracy.

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 as a lot of parameters	Reduce the size of the image in generator Drop the initial frames and alternate frame (for faster processing)
3	Conv3D	OOM Error	Reduce the number of neurons in Dense layer, reduce the batch size
4	Conv3D	Training Accuracy: 0.3170 Validation Accuracy: 0.4219	Increase the number of convolutional units/ layers
5	Conv3D	Training Accuracy: 0.2292 Validation Accuracy: 0.1953	Add Batch normalization, Increase the Conv unit
6	Conv3D	Training Accuracy: 0.9792 Validation Accuracy: 0.7109	Add dropout layers

7	**Conv3D**	**Training Accuracy: 0.9583** **Validation Accuracy: 0.7031**	**Remove one CNV layer and replace Flatten layer with GlobalAveragePooling3d layer**
8	**Conv3D**	**Training Accuracy: .9435** **Validation Accuracy: 0.6953**	**Use data augmentation**
9	**Conv3D + Data Augmentation**	**Training Accuracy: 0.6272** **Validation Accuracy: 0.7031**	**Need more epochs**
10	**`TimeDistributed + ConvLSTM2D`**	**Training Accuracy: 0.8318** **Validation Accuracy: 0.8047**	**More epoch may increase the accuracy.**
11	**`CNN+LSTM`**	**Training Accuracy: 0.9777** **Validation Accuracy: 0.8203**	

Final Model

After doing all the experiments, we finalized Model 11 - CNN+LSTM, which performed well.

Reasons

1. Training Accuracy : 0.9777 , Validation Accuracy : 0.8203
2. Number of trainable Parameters (1,004,549) with size 3.83 MB
3. Total trainable parameters (1,005,541) with size 3.84 MB
4. The best weights of CNN-LSTM: model-00048-0.04068-0.97768-0.62349-0.82031.h5 (11.6 MB).

Observations

- We encountered GPU out-of-memory errors when using a large batch size, which prompted us to experiment with different batch sizes until we found an optimal value that our GPU could handle.
- Model takes more time for training, if the number of trainable parameter increases.
- *Data Augmentation* and *Early stopping* greatly helped in overcoming the problem of overfitting which our initial version of model was facing.
- Our understanding is that the performance of model is influenced by various factors, including the nature of the data we used, the specific architecture we designed, and the hyper-parameters we selected. These elements collectively played a crucial role in determining which model performed better. As we noticed that CNN+LSTM-based model cells outperformed the Conv3D model in our experiments.
- Training time reduces as we increase batch size. But it also decrease the model accuracy.