Gesture Recognition – Deep learning

Problem Statement:

Imagine you are working 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 five different gestures performed by the user which will help users control the TV without using a remote.

The gestures are continuously monitored by the webcam mounted on the TV. Each gesture corresponds to a specific command:

- 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 categorised 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 which is similar to what the smart TV will use.

The data is in a zip file. The zip file contains a 'train' and a 'val' folder with two CSV files for the two folders.

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:

Model Description:

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).

2. **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 100 x 100 x 3, for example, the video becomes a 4D tensor of shape 100 x 100 x 3 x 30 which can be written as (100 x 100 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 '3Dconvolve' on each of the three channels of the (100 x 100 x 30) tensor.

Data Generator:

This is one of the most important parts of the code. In the generator, we are going to pre-process the images as we have images of different dimensions (*50 x 50, 70 x 70* and *120 x 120*) 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**: This was mainly done to ensure that the NN only recognizes the gestures effectively.
- **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.

CNN 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. 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 SGD as it lead to improvement in model's accuracy by rectifying high variance in the model's parameters. Played with multiple parameters of the SGD like decay_rate, starting learning rate.
- 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 in spite 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 Vs GPU memory:** A large batch size can throw **GPU Out of memory error** (eg: Model-1 has batch size of 64), 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 (RTX 5000 in Jarvis Labs).
- We also found out that the middle frames gives us most of the information and because the train images were chosen so carefully, data augmentation was not required though left-right flipping and zoom, slight rotation could have been done.
- Increasing the batch size leads to decrease in 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 or for more accuracy we can choose smaller batch size.
- **Conv3D** had better performance than **CNN2D+LSTM** based model with GRU cells. 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** Architecture due to its light-weight design and high-speed performance coupled with low maintenance as compared to other well-known architectures like VGG16, AlexNet, GoogleNet etc.

Model Overview:

Conv3D:

Model Name	Model Type	Number of parameters	Frames/ Batch_size	Epochs	Best Validation accuracy	Corresponding Training accuracy	Observations
conv_3d1_model	Conv3D	18,615,813	10/64	15	-	-	Due to higher batch size, the GPU is not able to load the entire model and fit failed with our of memory error. So, we tried next model with less batch size.
conv_3d2_model	Conv3D	6,557,189	6/20	20	70.83%	88.68%	Training and validation Accuracy are good but not up to the mark. Frame shape used 50,50. Next we can try to increase the frames and batch size.
conv_3d3_model	Conv3D	6,557,189	10/30	20	70.83%	82.75%	Keeping the same shape and increasing the number of frames we have observed that training and validation accuracy decreased and as compared to Model-2. Let's try resizing to 100*100 in the next iteration.
conv_3d4_model	Conv3D	14,814,725	10/50	25	78.00%	91.29%	With the increase of resize shape and the batch, we see there is a increase in both training and validation accuracy, and it shows somewhere the signs of overfitting. Let's try some shape in the middle: 70 * 70 and check with more frames in the hope of better results.

conv_3d5_model	Conv3D	14,683,653	18/50	25	74.00%	93.29%	With shape 70 * 70 and 18 frames out of 30, the model is clearly an overfit model and can see that increasing in number of frames and epochs causing the noise to be learned from all the frames
	Conclusion: By analysing above models, we observe that, with the lower frame shape and one fifth of the frames, we were able to get much better accuracy in both training and validation. It also computes faster.						

Time Distributed (CNN + LSTM/GRU):

Model Name	Model Type	Number of parameters	Frames/ Batch_size	Epochs	Highest Validation accuracy	Corresponding Training accuracy	Observations
CNN_RNN_1	TimeDistributed	3,807,589	18/50	25	50.00%	75.00%	We tried a basic CNN 2d with RNN LSTM and we didn't get good accuracy and sees overfit. Model not learning much info in training, not performing well in validation also. We took image shape 70 *70, using 18 frames. Let's, try with shape of 50*50.
CNN_RNN_2	TimeDistributed	3,807,589	10/20	20	54.69%	86.08%	With the new shape and the frames reduced and keeping the same layers we can see improve in training accuracy, but the validation accuracy didn't improve much. Still, this seems to be overfit. Let's replace LSTM with GRU and check. Also, we will tweak the layers to reduce the overfitting.

CNN_RNN_3 (CONV2D + GRU)	TimeDistributed	336,741	18/20	20	61.00%	90.15%	With the same shape 50*50, we have got better validation but still there is a overfit visible between the training and the validation accuracy.

Transfer Learning Models (CNN + RNN):

MobileNet model is considered as its parameter size is less compared to Inception and Resnet models

Model Name	Model Type	Number of parameters	Frames/ Batch_size	Epochs	Highest Validation accuracy	Corresponding Training accuracy	Observations
MobileNet Transfer learning + GRU	Transfer Learning	3,693,253	18/5	15	94.00%	99.25%	Usage of transfer learning with all the layers trainable=True, we just added few layers and GRU which gave us the almost perfect score. This model is almost learning everything in training and validation.

Conclusion:

The Model built with MobileNet Transfer learning + GRU (Model#9) gave better results compared to all the other models and also the model has a smaller number of parameters compared to other models.