CASE STUDY -: GESTURE RECOGNITION

Data Generator

This is one of the most crucial sections of the code. Generators are functions that return objects that may be traversed. They manufacture products one at a time and only when needed. The for loop is used to execute generators. We'll use the generator to pre-process the photos, which come in two sizes (360 × 360 and 120 x 160), as well as to generate a batch of video frames. The generator should be able to accept a large number of movies as input without mistake. Cropping, resizing, and normalising should all be completed successfully.

- ***Resizing* and *cropping* of the images.** This was done primarily to guarantee that the NN only detects the motions and not the other background noise in the image.
- ***Normalization* of the images.** Normalizing a picture's RGB values can sometimes be a quick and easy approach to remove image distortions caused by lights and shadows.
- Later on, to improve the model's accuracy, we used data augmentation, where we slightly rotated the pre-processed photos of the gestures to bring in additional data for the model to train on and to make it more generalizable in nature, since the hand's orientation may not always be within the camera frame.

NN Architecture development and training

- Different model configurations and hyper-parameters were tested, as well as numerous iterations and combinations of batch sizes, picture dimensions, filter sizes, padding, and stride length.

- ReduceLROnPlateau was used to lower the learning rate if the observed metrics (validation loss) did not change across epochs.

- We tested with Adam() optimizers but chose Adam() because it improved model correctness by correcting excessive variation in model parameters.

- Owing to the restricted computing power, we were against experimenting with Adagrad() and Adadelta() because they require a long time to converge due to their dynamic learning rate characteristics.

- When our model began to overfit, we used Batch Normalization, pooling, and dropout layers. This could be seen when our model began to give low validation accuracy while having high training accuracy.

- **Using GRU instead of LSTM**: Using GRU instead of LSTM looks to be a solid option. A GRU has much fewer trainable parameters than an LSTM.As a result, calculations would have been faster. However, its impact on validation accuracies might be investigated to see if it is a viable alternative to LSTM. Experimented with hyperparameters such as activation functions ReLU. filter size, paddings, batch normalization, and dropouts, among others.

Observations

- It was discovered that as the number of trainable parameters grows, the model takes much longer to train. Batch size / GPU memory / computation power.

- Downgraded the numpy version to 1.18.5, because while importing GRU nimble box was showing error.[Not implemented error]

- A huge batch size can cause a GPU Out of Memory error, therefore we had to experiment with the batch size until we found an appropriate amount that our GPU could tolerate (NVIDIA Tesla K80 GPU with 12GB memory provided by nimblebox.ai platform.)

- Increasing the batch size decreases training time significantly, but does so at the expense of model accuracy. This helped us realise that there is always a trade-off here based on priority: if we want our model to be available in a shorter time span, we should select a bigger batch size; if we want our model to be more accurate, we should choose a smaller batch size.

- Data augmentation and early halting were really helpful in resolving the issue of overfitting that our first model had. Conv3D was outperformed by a CNN+LSTM based model using GRU cells. As far as we can tell, this is dependent on the type of data we utilised, the architecture we built, and the hyper-parameters we selected. Transfer learning improved the model's overall

accuracy. In comparison to other well-known designs such as VGG16, AlexNet, GoogleNet, and others, we chose the MobileNet Architecture because of its light weight design and high speed performance combined with low maintenance.

SUMMARY -:

Model Name	**Model Type**	**Training Accuracy**	**Validation Accuracy**	**No. of parameters**	**Decision + Explanation**	**Result**
MODEL 1 Model_Conv3D_1	Conv3D	66.42%	62.00%	230,949	Reducing the number of parameters.	Low memory footprint.
MODEL 2 Model_Conv3D_2	Conv3D .	77.14 %	79.00 %	3,638,981	Using (3,3,3) Filter & 160x160 Image resolution. More Augmentation is done than MODEL 1.	Accuracy improved than model 1.
MODEL 3 RNNCNN_TL_3	**CNN + RNN**	**96.53 %**	**97.00 %**	**3,693,253**	**Transfer Learning with GRU and training all weights**	**Best model achieved in terms of model accuracy.**

MODEL 3 -: RNNCNN_TL_3 has training accuracy **96.53 %** and validation accuracy **97.00 %.** Therefore we consider model 3 as the best and final model.

SUGGESTION/TIPS-:

Transfer learning methods are best because Learning transfer activities increases both skill utilisation and performance results, according to the findings. When compared to those who just got training, skill utilisation was over 90% percent greater and performance effect was 48 percent higher in those who received training + learning transfer activities.