## **Gesture Recognition-ML-C40**

**Maniraj Madishetty – Group Facilitator**

**Yashyasvi Agarwal**

# Problem Statement

We want to develop a cool feature in the smart-TV that can recognize 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 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.

# 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.

# Three types of architectures suggested for analysing videos using deep learning:

1. **3D Convolutional Neural Networks (Conv3D)**
2. **CNN + RNN architecture**
3. **Transfer Learning (CNN Pretrained + GRU)**

# Data Pre-processing

* ***Resizing* and *cropping* of the images.** This was mostly done to make sure that the NN only effectively recognises the motions and doesn't concentrate on the other background noise in the image.
* ***Normalization* of the images.** To remove distortions brought on by lighting and shadows in an image.
* ***Data Augmentation*.** In order to give the model additional data to train on and to make it broader in nature as occasionally the positioning of the hand won't always be within the camera frame always images were rotated the pre-processed photos of the motions.

Graphical user interface, application

Description automatically generated Graphical user interface, application

Description automatically generated

# NN Architecture development and training

* Iterations and combinations of batch sizes, image dimensions, filter sizes, padding, and stride length were tested along with various model configurations and hyper-parameters. As part of our experimentation with various learning rates, ReduceLROnPlateau was utilised to slow down learning if the measured metrics (val loss) remained constant between epochs.
* When our model began to overfit, we additionally used batch normalisation, pooling, and dropout layers. This was evident when our model began to provide poor validation accuracy despite having good training accuracy.

# Observations

* We had to experiment with the batch size until we found an ideal batch size that our GPU could tolerate because a huge batch size can cause GPU Out of memory errors.
* As Number of trainable parameters increase, the model takes much more time for training.
* The training time is significantly decreased by increasing the batch size, but the model accuracy suffers as a result. This made us realise that there is always a trade-off based on priority: If we want our model to be ready faster, we should choose a larger batch size; but, if we want our model to be more accurate, we should choose a smaller batch size.
* Data augmentation was quite helpful in resolving the overfitting issue that our initial model version was having.
* The model's overall accuracy was improved by transfer learning.

Model Experimentation Results:

We have experimented with multiple parameters, however the final notebook has only the final models selected for each architecture.

|  |  |  |  |
| --- | --- | --- | --- |
| **Experiment Number** | **Model** | **Result** | **Decision + Explanation** |
| **1** | **Conv3D** | **Out Of Memory Error** | **Reduce the batch size and crop the images correctly, try to overfit on less amount of data** |
| **2** | **Conv3D** | **Training Accuracy: 0.99**  **Validation Accuracy: 0.81** | **Overfitting**  **Let’s add some Dropout Layers** |
| **3** | **Conv3D** | **Training Accuracy: 0.72**  **Validation Accuracy: 0.79** | **Adding dropouts has caused validation accuracy to increase more than training accuracy. Let us move to CNN+LSTM** |
| **4** | **CNN+LSTM** | **Training Accuracy: 0.83**  **Validation Accuracy: 0.78** | **Although the accuracy has improved it is still low, we might need to induce transfer learning** |
| **Final Model** | **Transfer Learning + GRU** | **Training Accuracy: 94**  **Validation Accuracy: 97** | **Transfer learning with learnt weights worked well and we are getting very high accuracy.** |

After going through all the models, we finally decided to use the model with transfer learning as it gives us the most accuracy on both Train and Validation set. This model is performing very accurately and can be used as the final model in production.