## **Gesture Recognition Project**

**Prasad YVVSN Pechetti**

# Project Goal

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.

* 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.

# Dataset Analysis

The training data includes several hundred videos, each assigned to one of five categories. Each video, usually lasting 2-3 seconds, is split into a sequence of 30 frames (images). These videos were captured by different individuals performing one of the five gestures in front of a webcam, resembling the setup used by a smart TV.**A picture containing photo, many, various, sitting

Description automatically generated**

# Aim

Our aim is to train various models using the 'train' folder to predict the action performed in each video sequence, ensuring they also perform well on the 'val' folder. The 'test' folder, reserved for final evaluation, will be used to assess the performance of the final model.

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

1. **3D Convolutional Neural Networks (Conv3D)**

The 3D convolutions are an extension of the 2D convolutions you're already familiar with. While in 2D convolutions the filter moves in two directions (x and y), in 3D convolutions, the filter moves in three directions (x, y, and z). In this case, the input to a 3D convolution is a video, which consists of a sequence of 30 RGB images. If each image has a shape of 100 x 100 x 3, the video becomes a 4D tensor with the shape 100 x 100 x 3 x 30, or equivalently (100 x 100 x 30) x 3, where 3 is the number of channels. Drawing an analogy to 2D convolutions, where a 2D filter is represented as (f x f) x c (with f being the filter size and c the number of channels), a 3D filter (a cubic filter) is represented as (f x f x f) x c, where c is 3 due to the three channels in each image. This cubic filter will now perform a 3D convolution on each of the three channels of the (100 x 100 x 30) tensor.

1. **CNN + RNN architecture**

The Conv2D network will generate a feature vector for each image, and a sequence of these feature vectors is subsequently passed to an RNN-based network. The RNN's output is a standard softmax, which is used for classification in this case

A close up of a sign

Description automatically generated

**Figure 2: A simple representation of an ensembled CNN+LSTM Architecture**

# Data Generator

# This is a critical part of the code. In the generator, we will pre-process images with dimensions of (120 x 120) and create a batch of video frames. The generator must handle a batch of videos as input without errors. It should successfully perform steps such as cropping, resizing, and normalization.

# Data Pre-processing

**Resizing and cropping of the images**: This step was primarily carried out to ensure that the neural network focuses on recognizing the gestures effectively, rather than being influenced by irrelevant background noise in the image.

**Normalization of the images**: Normalizing the RGB values helps eliminate distortions caused by lighting and shadows, providing a simple and effective way to improve thequality

of the images for processing.

***.***

# NN Architecture development and training

# Model configurations and hyperparameter tuning: We experimented with various model configurations and hyperparameters, testing different combinations of batch sizes, image dimensions, filter sizes, padding, and stride lengths. Additionally, we explored different learning rates and used ReduceLROnPlateau to decrease the learning rate if the monitored metric (val\_loss) remained unchanged across epochs.

# Optimizer choice: We tested the Adam() optimizer, as it helped improve the model's accuracy by addressing the high variance in the model's parameters.

# Overfitting prevention: To prevent overfitting, we implemented Batch Normalization, pooling, and dropout layers. Overfitting became evident when the model showed poor validation accuracy despite achieving good training accuracy.

# Early stopping: We employed early stopping to halt the training process when the validation loss began to plateau or when the model’s performance stopped improving.

# Observations

* It was observed that as the Number of trainable parameters increase, the model takes much more time for training.
* 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.
* *Data Augmentation* and *Early stopping* greatly helped in overcoming the problem of overfitting which our initial version of model was facing.
* *CNN+LSTM* based model with *GRU* cells had better performance than *Conv3D.* 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*](https://arxiv.org/abs/1704.04861) Architecture due to it’s light weight design and high speed performance coupled with low maintenance as compared to other well-known architectures like VGG16, AlexNet, GoogleNet etc.
* For detailed information on the Observations and Inference, please refer Table 1.

After doing all the experiments, we finalized Model 8– CNN+GRU, which performed well.

Reason:

 (Training Accuracy: 99%, Validation Accuracy: 97%)

 Number of trainable parameters (3,669,317)

 Learning rate gradually decreased after some Epochs.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model No** | **Model** | **No. of frames** | **Image Size** | **Batch Size** | **Result** | **Decision + Explanation** |
| 1 | Conv3D | 30 | 120 x 120 | 60 | Train Accuracy: 0.85  Validation Accuracy: 0.27 | This model has very low Validation accuracy when compared to the training accuracy. Also, the validation loss did not improve from 1.5. Hence let us modify the batch size. |
| 2 | Conv3D | 20 | 120 x 120 | 10 | Train Accuracy: 0.91  Validation Accuracy: 0.80 | This model seems to be overfitting. The validation loss did not improve much from 0.52. |
| 3 | Conv3D | 30 | 120 x 120 | 10 | Train Accuracy: 0.89  Validation Accuracy: 0.81 | The validation accuracy has marginally improved from the previous model. The validation loss did not improve much from 0.48. |
| 4 | Conv3D | 20 | 120 x 120 | 10 | Train Accuracy: 0.89  Validation Accuracy: 0.75 | The validation loss went up in this model. It was 0.65. And also we can observe that the training accuracy was retained as before, but the validation accuracy has dropped. The time is increased to 243s. Let’s try CNN + GRU model, as Conv3D is not giving us desired validation accuracy. |
| 5 | MobileNet  Transfer  Learning  Model + GRU | 20 | 120 x 120 | 10 | Train Accuracy: 0.997  Validation Accuracy: 0.91 | Validation loss has dropped to 0.11. Let us increase the no. of frames to see if val\_accuracy would increase further. |
| 6 | MobileNet  Transfer  Learning  Model + GRU | 30 | 120 x 120 | 10 | Train Accuracy: 0.99  Validation Accuracy: 0.97 | Validation loss improved from 0.11 to 0.055. Also, we can see the improvement in the validation accuracy in this mode. This is by far the best model we have got.  This model is selected as the final model, as we have got 97% Validation accuracy. |
| 7 | MobileNet  Transfer  Learning  Model + LSTM | 20 | 120 x 120 | 10 | Train Accuracy: 0.99  Validation Accuracy: 0.89 | With LSTM model, we can see that the validation loss did not improve from 0.133. But the training time is less when compared to GRU models. |
| 8 | MobileNet  Transfer  Learning  Model + LSTM | 30 | 120 x 120 | 10 | Train Accuracy: 0.98  Validation Accuracy: 0.95 | With LSTM model, we can see that the validation loss did improve from 0.133 to 0.108. But these figures are big when compared to GRU’s validation loss. |
| **Final Model** | MobileNet  Transfer  Learning  Model + GRU | 30 | 120 x 120 | 10 | Train Accuracy: 0.99  Validation Accuracy: 0.97 | Validation loss improved from 0.11 to 0.055.  This model is selected as the final model, as we have got 97% Validation accuracy. |

**Table -1: Analysis and Results**

**Additional Recommendations for Enhancements :**

* Using Transfer Learning: Leveraging a pre-trained model such as ResNet50, ResNet152, or Inception V3 to extract initial feature vectors could enhance the model's performance. These feature vectors can be passed to an RNN to capture sequence information before the final classification through a softmax layer, potentially improving gesture recognition accuracy.
* Deeper Understanding of Data: The videos were recorded in varied backgrounds, lighting conditions, and by different individuals using different cameras. A more thorough analysis of the images could reveal additional insights, adding diversity to the dataset. Incorporating this extra information in the generator function could increase both the stability and accuracy of the model.
* Tuning Hyperparameters: Further experimentation with different combinations of hyperparameters, such as activation functions and optimizers (e.g., Adagrad() and Adadelta()), could result in the development of better, more accurate models. Additionally, fine-tuning hyperparameters like filter size, padding, stride length, batch normalization, and dropout rates may further enhance the model's performance.