Neural Networks Group Case Study: Hand Gesture Recognition Project

# Problem Statement:

This project involves building a 3D Convolutional Neural Network (CNN) to correctly recognize hand gestures by a user to control a smart TV.

The objective of this project is to build a hand gesture recognition model that can be hosted on a camera installed in a smart TV that can understand 5 gestures.

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

# About 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 - similar to what the smart TV will use.

The videos have two types of dimensions - either 360x360 or 120x160 (depending on the webcam used to record the videos).

Data Source : <https://drive.google.com/uc?id=1ehyrYBQ5rbQQe6yL4XbLWe3FMvuVUGiL>

# Neural Network Architectures Used:

For analysing videos using neural networks, two types of architectures are used commonly.

## 1. Convolutions + RNN

One is the standard CNN + RNN architecture in which you pass the images of a video through a CNN which extracts a feature vector for each image, and then pass the sequence of these feature vectors through an 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).

In place of generic RNN, LSTM and GRU has been used in our experiments.

The image network will just give some feature representation but the LSTM/GRU will be able to decipher the sequence information to classify them as one of the class.

Then dense layer output can be fed in sequence to LSTM/GRU to get the desired output.

An LSTM has 4 gates, while GRU has 3 gates. Using GRU will significantly reduce the training times as it needs to compute values for 3 gates and its performance is at par with the LSTMs.

Another advantage here is we can use the transfer learning here, since the state of the art networks are already available for the image classification, we can easily use the already trained weights of resNet or VGGNet and then we can use those networks to transform these images to give the image representation (eg dense layer output).

The dense layer which will be used will be standard models like resnet, VGGnet etc.

## 2. 3D convolutional network

The other popular architecture used to process videos is a natural extension of CNNs - a 3D convolutional network.

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 Ingestion Pipeline and Custom Generator:

As we already know, in most deep learning projects you need to feed data to the model in batches. This is done using the concept of generators.

Creating data generators is probably the most important part of building a training pipeline. Although libraries such as Keras provide built-in generator functionalities, they are often restricted in scope and you have to write your own generators from scratch. For example, in this problem, you need to feed batches of videos, not images. Similarly, in an entirely different problem such as 'music generation,' you may need to write generators which can create batches of audio files.

In this project, we have written our own batch data generator using the Python’s generator functions. A Python generator object requires very less memory as compared to a function which is of primary importance in deep learning models.

Generators have huge advantages of performance/memory/execution time for very large datasets. Also, we have better readability and has all features available with the python native objects.

The generator yields a batch of data and 'pauses' until the fit\_generator calls next(). Note that in Python-3, this functionality is implemented by \_next\_().

This is based on the concept of the lazy evaluation.

Yield statement in generator returns one value at a time. Generator helps us to bring that amount of data into memory to process stuffs. This helps us to do the batch wise gradient descent on a model.

We use our own custom data generator not the in-built image data generator which is available with the Keras. The reason is we have variety of data from multiple sources like text, images, csv files, audio etc.

Otherwise it might takes 100s of Gigs of memory for processing using ImageNet etc using the normal Keras.fit() .

| **Experiment Number** | **Model** | **Result (Last Epoch which was saved)** | **Decision + Explanation** |
| --- | --- | --- | --- |
| 1 | Conv3D | Training Accuracy = 96.83%  Validation Accuracy = 87.00% | The model showed continuous improvement, with a significant jump in validation accuracy at Epoch 14. The validation loss also improved significantly (0.32493), indicating effective learning. However, the model later showed signs of overfitting (training accuracy continued increasing, but validation accuracy stagnated or fluctuated). The decision would be to use early stopping around this epoch and possibly increase regularization (e.g., dropout or L2 weight decay) to prevent overfitting. |
| 2 | Conv3D | Training Accuracy = 98.04%  Validation Accuracy= 89.00% | The model shows strong performance, with a high validation accuracy of 89%, indicating good generalization. The loss decreased steadily, and learning rate reductions helped improve stability. The model is suitable for deployment or further fine-tuning with additional data or augmentation techniques. |
| 3 | Conv3D | Training Accuracy= 83.11% Validation Accuracy= 78.00% | The model demonstrated good learning progression with improving validation accuracy. However, after epoch 12, the validation loss started increasing (potential overfitting). The ReduceLROnPlateau reduced the learning rate at epoch 15, which helped stabilize but did not significantly improve validation performance. Consider early stopping at epoch 12 or introducing additional regularization to prevent overfitting. |
| 4 | Conv3D | Training Accuracy= 81.45% Validation Accuracy= 86.00% | The model performed well with a peak validation accuracy of 86%. However, validation loss started increasing after epoch 11, indicating overfitting. The early stopping mechanism restored the model to epoch 11 as it had the lowest validation loss (0.50669). The model learned well but started overfitting around epoch 12-14, suggesting either more regularization, dropout tuning, or an early stopping intervention at the right time. Reducing the learning rate after plateauing at epoch 14 was a good approach, but it didn’t improve the performance further. |
| 5 | CNN + GRU | Training Accuracy = 73.60% Validation Accuracy = 69% | Model shows improvement in validation accuracy over time but starts with poor generalization. Initially, the validation accuracy was quite low (starting at 0.18), and the model struggled to generalize. The learning rate reduction helped stabilize training, and by epoch 17, validation accuracy improved to 0.54. The model appears to benefit from more epochs, but further tuning (regularization, data augmentation, or more complex architecture) may help generalization. |
| 6 | CNN  + LSTM2D | Training Accuracy= 29.11% Validation Accuracy= 24% | The model is not learning well. The accuracy is quite low, suggesting underfitting. The learning rate was reduced twice (Epoch 10 and 17), but validation accuracy fluctuates around 20-30%. More architecture tuning is needed, possibly increasing the dataset size, tuning hyperparameters, or modifying the architecture to include attention mechanisms for better sequential learning. |
| 7 | CNN  + LSTM2D | Training Accuracy = 59.73% Validation Accuracy = 64% | The model is showing an improving trend, with validation accuracy surpassing training accuracy. This might indicate good generalization, but it could also hint at slightly unstable training. More epochs may be needed to confirm if performance plateaus or improves. Potential enhancements: increasing batch size, adjusting dropout rates, or fine-tuning learning rate reduction. |
| 8 | CNN  + LSTM2D | Training Accuracy= 39.67%, Validation Accuracy= 47.00% | The model is showing some improvement, but the accuracy is still relatively low, indicating potential issues such as underfitting or inadequate learning. Consider adjusting hyperparameters, increasing training data, or using data augmentation. |
| 9 | CNN + GRU | Training Accuracy= 51.89% Validation Accuracy= 58% | The model shows a steady improvement in validation accuracy, reaching 50% at Epoch 26. However, there is a minor fluctuation in validation loss, suggesting potential overfitting beyond this point. Further training might not significantly improve accuracy. Consider early stopping or tuning hyperparameters like learning rate or regularization. |
| Final Model | MobileNet + GRU | Training Accuracy = 96.83% Validation Accuracy = 95.00% | The model shows strong generalization, with high validation accuracy close to training accuracy. The decreasing loss across epochs indicates good learning. However, the minor fluctuation in validation accuracy (e.g., drop in Epoch 19) suggests some potential overfitting, but overall, the model performs well. |