Neural Networks Project - Gesture Recognition

- **Kriti Pawar**
- **Kuldeep Lodha**

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.

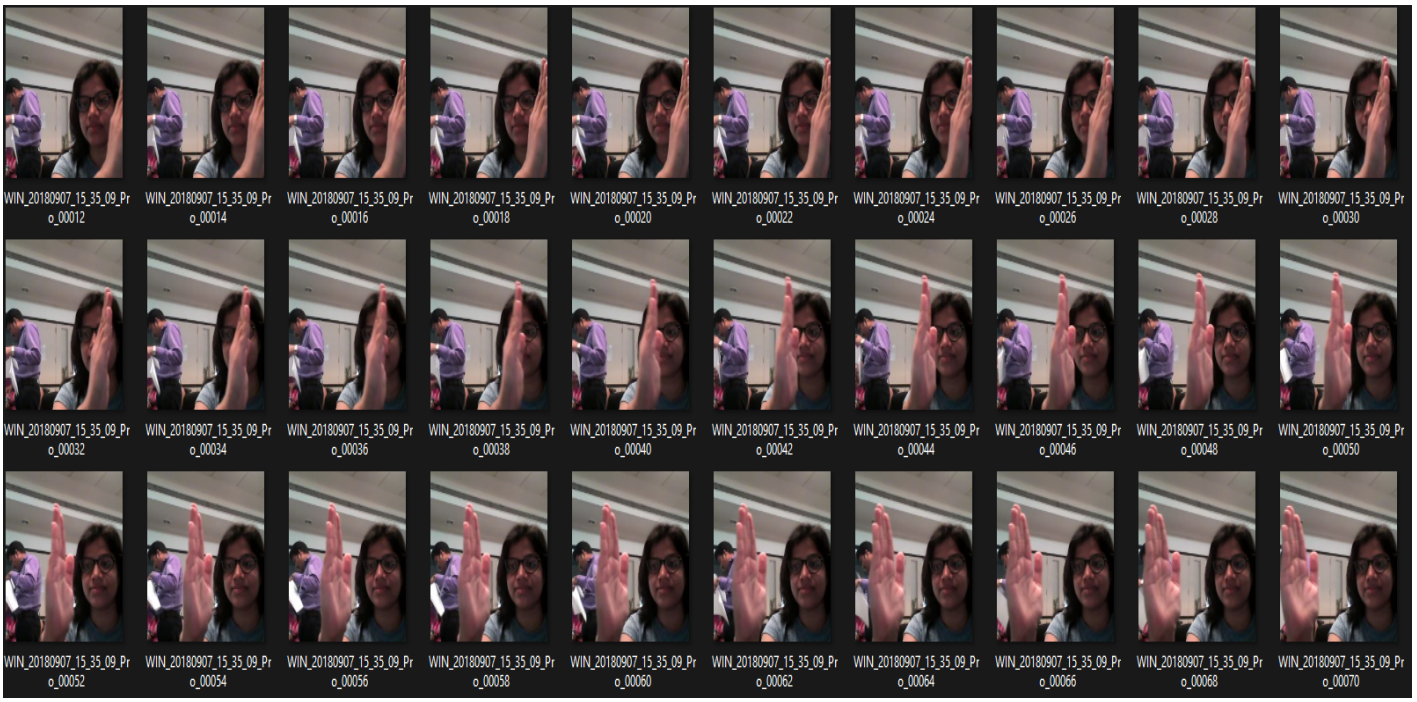
The gestures are continuously monitored by the webcam mounted on the TV. Each gesture corresponds to a specific command:

- **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**

Here's the data: https://drive.google.com/uc?id=1ehyrYBQ5rbQQe6yL4XbLWe3FMvuVUGiL

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 - similar to what the smart TV will Use.



Objective

Your task is to train a model on the 'train' folder which performs well on the 'val' folder as well (as usually done in ML projects). We have withheld the test folder for evaluation purposes - your final model's performance will be tested on the 'test' set.

Model Architecture :-

For analysing videos using neural networks, two types of architectures are used commonly. 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 other popular architecture used to process videos is a natural extension of CNNs - a 3D convolutional network. In this project, you will try both these architectures.

1. **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 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.

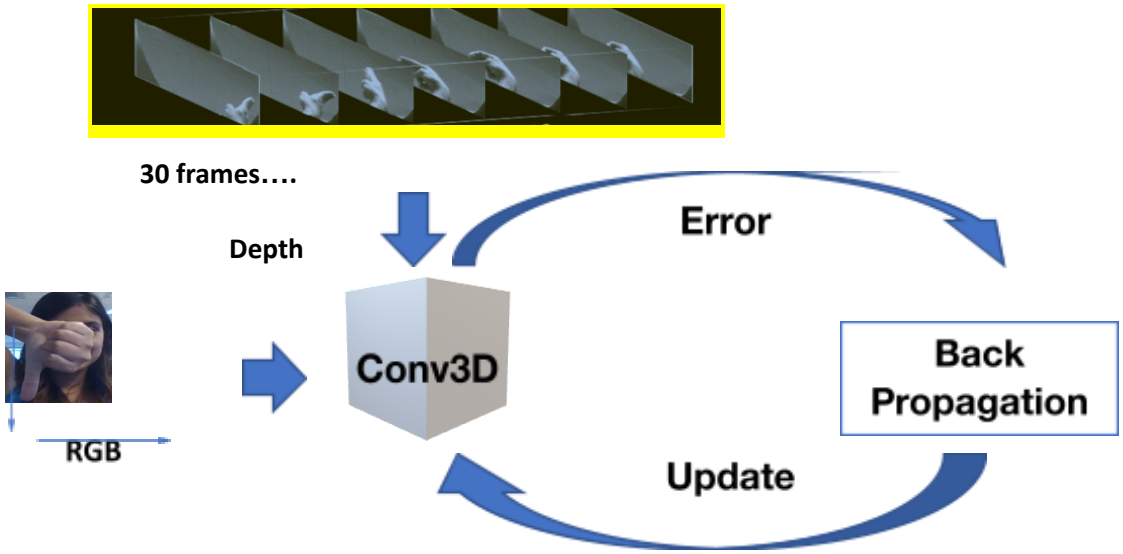


Figure 1: A simple representation of working of a 3D-CNN

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).

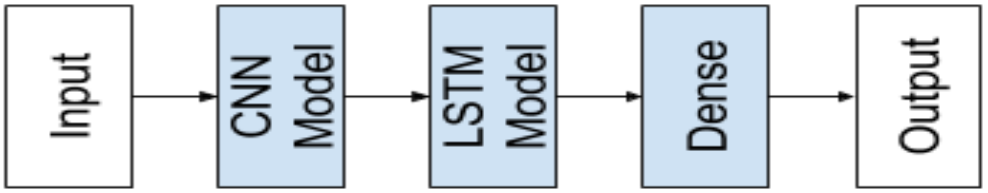


Figure 2: A simple representation of an ensembled CNN+LSTM Architecture

Data Generator

This is one of the most important part of the code. In the generator, we are going to pre-process the images as we have images of 2 different dimensions (*360 x 360* and *120 x 160*) 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* and *cropping* of the images.** This was mainly done to ensure that the NN only recognizes the gestures effectively rather than focusing on the other background noise present in the image.
- ***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.

NN Architecture development and training

- Explored various model configurations and hyperparameters. Conducted multiple iterations involving batch sizes, image dimensions, filter sizes, padding, and stride lengths. Also experimented with different learning rates, implementing ReduceLROnPlateau to decrease the learning rate in case the monitored metrics (val_loss) remained unchanged 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.

- Additionally, we applied Batch Normalization, pooling, and dropout layers to counter overfitting. This became evident when the model displayed poor validation accuracy despite 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 ∝ GPU memory / available compute.** A large batch size can throw *GPU Out of memory error,* 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
- Raising the batch size significantly reduced the training duration, albeit with a decline in model accuracy. This observation led us to understand the inherent trade-off: opt for a larger batch size for faster model preparation, or choose a smaller batch size for a more accurate model.
- *Regularizaation* and *Early stopping* greatly helped in overcoming the problem of overfitting which our initial version of model was facing.
- *Transfer learning* **boosted** the overall accuracy of the model. We made use of the *MobileNet* 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.

Table 1: Observations and Results for numerous tested NN architectures

MODEL	EXPERIMENT	RESULT	DECISION + EXPLANATION	TOTAL PARAMETERS
Conv3D	1	Training Accuracy : 0.93 Validation Accuracy : 0.76	Overfitting. Let's try to use Adam() optimizer instead of SGD()	9,731,333
	2	Training Accuracy : 0.62 Validation Accuracy : 0.66	The accuracy is decreased when we use Adam() optimizer.	9,731,333
	3	Training Accuracy : 0.98 Validation Accuracy : 0.67 (Best weight Accuracy,Epoch:6/25)	We added three more layers with Batchnormilization and MaxPooling but the Val_loss didn't improve from 0.67 so early stopping stop the training process.	2,302,725

	4	Training Accuracy : 0.92 Validation Accuracy : 0.69 (Best weight Accuracy,Epoch:12/25)	Overfitting has reduced but accuracy hasn't improved. Let's trying adding more layers	2,302,725
	5	Training Accuracy : 0.8326 Validation Accuracy : 0.81	*Upon resizing the image dimensions to 120x120, significant improvements in accuracy were observed across both the training and validation sets.*	27,335,429
Conv2D with GRU	6	Training Accuracy : 0.93 Validation Accuracy : 0.62	*The model exhibits strong training accuracy but appears to face challenges with generalization to the validation dataset.*	2,613,013
Conv2D + LSTM	7	Training Accuracy : 0.8492 Validation Accuracy : 0.70	*The model exhibits strong training accuracy but appears to face challenges with generalization to the validation dataset..*	787,957
Conv2D + LSTM (with Regularization)	8	Training Accuracy : 0.7149 Validation Accuracy : 0.60	Model's current performance indicates suboptimal results with high losses and significant training-validation metric gaps. Further analysis and model refinements are needed to improve generalization for unseen data	787,957
Transfer Learning + LSTM	9 (Model-8 on Notebook)	Training Accuracy : 0.98 Validation Accuracy : 0.86	We use transfer lerning with mobilenet and LSTM. In epoch 28, the model achieved outstanding results. We will choose this model as our final model, as it is giving the best performance.	4,959,173

Further suggestions for improvement:

- **Using Transfer Learning**: Utilized pre-trained ResNet50/ResNet152/Inception V3 for extracting initial feature vectors. These were then processed through a recurrent neural network (RNN) for sequence information and subsequently to a softmax layer for gesture classification.
- **Deeper Understanding of Data:** The video clips were recorded in diverse environments with variations in lighting, individuals, and camera sources. Further exploration and inclusion of

these images can diversify the dataset, potentially offering more insights. Leveraging this added information within the generator function can enhance the model's stability and accuracy.

- **Tuning hyperparameters:** Exploring various combinations of hyperparameters, such as different activation functions (ReLU, Leaky ReLU, mish, tanh, sigmoid).