## **Deep Learning Project- Gesture Recognition**

**Sandeep Malagi** ( Note: Since my partner is not here as he shifted to other batch I am submitting alone )

# **Problem Statement:**

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.

# **Understanding the Dataset:**

* The training dataset is composed of several hundred short videos, each falling under one of five distinct categories. These videos, generally lasting between 2 to 3 seconds, are broken down into sequences of 30 individual frames. Diverse individuals have contributed to this dataset by recording themselves performing one of the five designated gestures using a webcam setup akin to that of a smart TV's camera

# **Objective:**

* Our objective is to develop models that can accurately identify the specific action depicted in each video from the training set. The effectiveness of these models will then be gauged on their ability to predict the actions in the validation set. The ultimate test of the model's accuracy will be determined using a separate 'test' dataset, which is reserved for the final evaluation.

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

**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 100 x 100 x 3, for example, the video becomes a 4D tensor of shape 100 x 100 x 3 x 30 which can be written as (100 x 100 x 30) x 3 where 3 is the number of channels. Hence, deriving the analogy from 2D convolutions where a 2D kernel/filter (a square filter) is represented as (f x f) x c where f is filter size and c is the number of channels, a 3D kernel/filter (a 'cubic' filter) is represented as (f x f x f) x c (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 (100 x 100 x 30) tensor.

**2. 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).

# **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**

Image Processing Techniques for Gesture Recognition Neural Network:

1. **Resizing and Cropping:**

* Resizing is changing the image dimensions to maintain consistency across all inputs to the neural network.
* Cropping helps to remove extraneous background details, allowing the network to concentrate on the hand gesture itself.

2. **Normalization:**

* This process adjusts the pixel intensity values across the RGB channels to mitigate distortions from variable lighting and shadows, standardizing the input data for the network.

3. **Data Augmentation:**

* Data augmentation increases data diversity by making slight modifications to existing images, enhancing the model’s ability to generalize.
* For instance, in our case by rotating images of hand gestures, the model learns to recognize gestures in various orientations, which is essential for real-world application where hand positioning varies.

# **NN Architecture development and training**

Enhancements in Neural Network Training:

1. **Model Configurations and Hyperparameter Tuning:**

* Engaged in extensive testing with various model structures and hyperparameters. Batch sizes, image resolutions, filter dimensions, padding, and stride lengths were methodically varied to optimize performance.
* Adjusted learning rates dynamically using ReduceLROnPlateau, which lowered the learning rate when no change in validation loss (val\_loss) was observed between epochs, enhancing the training efficacy.

2. **Optimizer Selection:**

* Compared Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation (Adam) optimizers. Adam was chosen for its efficacy in reducing high variance in the model parameters, which contributed to an uptick in accuracy.
* Opted against Adagrad and Adadelta, despite their dynamic learning rate capabilities, due to our computational constraints that make their convergence impractically slow.

3. **Combatting Overfitting:**

* Incorporated Batch Normalization, pooling, and dropout layers to mitigate overfitting. This issue was flagged by a discrepancy between high training accuracy and subpar validation accuracy.
* Utilized dropout layers to randomly deactivate certain neurons during training, preventing them from co-adapting too much.

4. **Early Stopping Implementation:**

* Implemented early stopping to cease training when val\_loss plateaued or when further epochs ceased to yield performance improvements, thus avoiding unnecessary computation and potential overfitting.

# **Observations**

**Neural Network Training Observations:**

1. **Trainable Parameters and Training Time:**

* A correlation was noted where an increase in the number of trainable parameters resulted in longer training durations. More parameters mean more computations during the forward and backward passes, which inherently extends the training time.

2. **Batch Size and Computational Resources:**

* Batch size is constrained by the memory capacity of the computing hardware, specifically the GPU. An excessively large batch size can exceed the GPU's memory, causing an Out of Memory (OOM) error. Therefore, finding an optimal batch size that balances computational resource utilization without causing errors is crucial.

3. **Batch Size versus Training Time and Accuracy:**

* While a larger batch size can significantly decrease the training time by processing more data at once, it can adversely affect model accuracy. This is due to the reduced number of updates to the model per epoch, which can impede the model's ability to converge to the most accurate parameters.

4. **Overcoming Overfitting with Data Augmentation and Early Stopping:**

* Data Augmentation introduces variability into the training data, which helps prevent the model from learning noise and specifics of the training set that do not generalize to new data.
* Early Stopping is a regularization technique where training is halted once the model's performance on a validation set ceases to improve, preventing it from learning idiosyncrasies of the training data that do not apply to the broader data distribution.

5. **Enhanced Accuracy with Transfer Learning:**

* Utilizing the MobileNet architecture, known for its efficiency and small footprint, facilitated a leap in accuracy. Transfer learning leverages a pre-trained network that has already learned a robust feature set from a large and diverse dataset, which can be fine-tuned to the specific task at hand, thus improving performance compared to training a model from scratch.
* The choice of MobileNet over other architectures like VGG16, AlexNet, and GoogleNet was based on its balance of speed and performance, essential for real-time applications and situations where computational resources are limited.

For comprehensive data and technical specifics, refer to the information provided in the accompanying Python notebook.

Github link :