**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

**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:**

To anticipate the action taken in each sequence or video, we must train various models on the 'train' folder that also perform well on the 'val' folder. The performance of the final model will be evaluated using the 'test' set; the final test folder is being withheld for evaluation.

**ARCHITETURES USED IN THE PROJECT:**

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.

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.

**Data Generator:**

One of the most crucial components of the code is this. Due to the fact that we have photographs in 2 distinct sizes (360 × 360 and 120 x 160), we will pre-process the images in the generator and produce a batch of video frames. A batch of videos should be able to be entered into the generator without any problems. Successful completion of procedures like cropping, resizing, and normalisation is required.

**Model building:**

Here you make the model using different functionalities that Keras provides. Remember to use Conv3D and MaxPooling3D and not Conv2D and Maxpooling2D for a 3D convolution model. You would want to use Time Distributed while building a Conv2D + RNN model. Also remember that the last layer is the SoftMax. Design the network in such a way that the model is able to give good accuracy on the least number of parameters so that it can fit in the memory of the webcam.

* Experimented with different model configurations and hyper-parameters and various iterations and combinations of batch sizes, image dimensions, filter sizes, padding and stride length were experimented with.
* We experimented with *SGD()*optimizers and used them in the code and then used it create the train\_geberator and the Val generator which was used in .fit generator.
* We also made use of *Batch Normalization*, *pooling* and *dropout* *layers* when our model started to overfit, this could be easily witnessed when our model started giving poor validation accuracy inspite of having good training accuracy

|  |  |  |  |
| --- | --- | --- | --- |
| **Experiment Number** | **Model** | **Result** | **Decision + Explanation** |
| **1** | **CNN + RNN with 5 epochs** | **Throws Generator error** | **Crop the images correctly, try to overfit on less amount of data** |
| **2** | CNN + RNN with Trainable VGGnet layers for 10 epochs | **Accuracy:**  **0.6100** | **Model can learn from data the accuracy is low trying the next model with higher epochs** |
| **3** | **CNN + RNN with Trainable VGGnet layers for 20 epochs** | **Accuracy:**  **0.8500** | **We can see the increase in the accuracy we are dropout layers in next experiment** |
| **4** | CNN + RNN without trainable VGGnet layers with additional dropout for 20 epochs | **Accuracy:**  **0.4400** | **When we add a dropout layers, we can see a dip in accuracy so in the next experiment we try to add without trainable VGGNet Conv layers for 20 epochs** |
| **5** | CNN + RNN without trainable VGGnet layers for 20 epochs | **Accuracy:**  **0.5900** | **We see the improvement in accuracy compared to the previous model** |
| **6** | Conv3D architecture with 25 epochs | **Accuracy**  **0.8200** | **Model has similar accuracy compared to that of model 3 and it is the second highest accuracy obtained among all of them** |
| **Final model (model 3 in notebook:**  **(**model-00020-0.42394-0.95023-0.54457-0.85000.h5**)** | CNN + RNN with Trainable VGGnet layers for 20 epochs | **Accuracy: 0.8500** | **By experimenting all the model with various hyperparameter tuning, (experiment 3 in write up and notebook) has the highest accuracy with minimal parameters** |
|  |  |  |  |

The best weights of CNN-LSTM: (model-00020-0.42394-0.95023-0.54457-0.85000.h5) Experiment 3 - CNN + RNN with Trainable VGGnet layers for 20 epochs. we considered this weight for model testing.

GESTURE RECOGNITION PROJECT AND WRITE UP BY,

Harika Guntur

Allauddin

Dilpreet

------------------------------------- END OF THE WRITE UP ----------------------------------------------