**Neural Networks Project - Gesture Recognition**

* Sagar Zanwar (ML C39)

**Problem Statement:**

You 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

**Dataset Understanding:**

* 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

*Sample*:

**A picture containing photo, many, various, sitting

Description automatically generated**

Task:

* To train different models on the 'train' folder to predict the action performed in each sequence or video and which performs well on the 'val' folder as well

Suggested Solution Methods:

1. **3D Convolutional Neural Networks (Conv3D)**
   * 3D convolutions are a natural extension to the 2D convolutions
   * 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)
   * Cubic filter will '3D-convolve' on each of the three channels of the (100x100x30) tensor
2. **2D CNN + RNN:**
   * 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
   * To elaborate: 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).

**Pre-processing:**

**Data Generator:**

* In most deep learning projects you need to feed data to the model in batches
* This is done using Generators
* While we have Generators readily available in libraries such as Keras, we will build the generator from the scratch as we should be able to use this generator with other cases as well
* Enabled by python Yield statement called repeatedly, the generator will create batches and pass the last saved state of the parameters without wasting the space/memory
* Generator **yields a batch of data** and 'pauses' until the ‘fit\_generator’ calls \_\_next\_\_()

**Resizing and cropping of the images:**

* This is done to ensure that the NN only recognizes the gestures effectively rather than focusing on the other background noise present in the image
* Resizing and cropping also insures uniformity in the data

**Normalization:**

* With normalization of different channels (RGB) we can get rid of distortions, shadows extra lights

**Model Training:**

* As part of model training we have tried multiple experiments
  + Changing Model Architecture
  + Changing batch size
  + Increasing Epochs (to wait for model to be stable)
  + Use of Transfer Learning to use standard models as starting point
* It was observed that:
  + As we change architecture, the change in weights to train changes time to train model significantly
  + As we increase batch size, the GPU memory usage also increases
  + The increase in batch size decreased training time, however, we had to compromise on model accuracy
  + Use of Transfer learning significantly improved model accuracy as well as time taken for training (Likely because the weights are already trained to a great efficiency)
  + Reduce Learning Rate on plateau reduced learning rate and training time increased but accuracies improved

**Models Used:**

(Note: During the experimentation we have used many more models, and changed same models as need be. In table below, we have presented details only for the models that are present in final python file attached in zip

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Model | Accuracy | Details | # of parameters |
| Model 1 | Conv 3D | Epochs = 10  Batch Size = 30  Train: 0.59  Test: 0.35  Optimizer = Adam | Base Architecture: 3 Conv3D layers, , Flatten, 3 Dense Layers (last with SoftMax 5 class), with batch normalization and max-pooling  Model Underfit | 87,184,197 |
| Model 2 | Conv3D | Epochs = 10  Batch Size = 30  Train: 0.76  Test: 0.32  Optimizer = Adam | Base Architecture: 4 Conv3D layers, , Flatten, 2 Dense Layers (last with SoftMax 5 class), with batch normalization and max-pooling  Model Overfit | 5,103,637 |
| Model 3 | Conv3D | Epochs = 10  Batch Size = 10  Train: 0.90  Test: 0.81  Optimizer = Adam | Base Architecture: 4 Conv3D layers, , Flatten, 2 Dense Layers (last with SoftMax 5 class), with batch normalization and max-pooling  Number of channels/kernels reduced and reduced number of batches  Model Looks stable | 4,893,797 |
| Model 4 | Conv3D | Epochs = 20  Batch Size = 10  **Train: 0.97**  **Test: 0.90**  Optimizer = Adam | Base Architecture: 4 Conv3D layers, , Flatten, 2 Dense Layers (last with SoftMax 5 class), with batch normalization and max-pooling  Number of epochs increased | 4,893,797 |
| Model 5 | CNN + RNN | Epochs = 15  Batch Size = 10  **Train: 0.99**  **Test: 0.92**  Optimizer = SGD | Base Architecture: Transfer Learning + RNN:  Time distributed layer taking mobilenet input with imagenet weights followed by time dist flatten layer, GRU layer, 2 Dense layers (last with SoftMax 5 class) | 6,405,189 |

Observations:

* We have also tried hands on layers of LSTM and different batch and epoch size in transfer learning model, however, the presented combination ‘Model 5’ happens to be the best
* Transfer Learning model provided the stability to the model as well as greater accuracy at the cost of some increase in number of parameters to be trained.