## Neural Networks Project - Gesture Recognition

# **Problem Statement**

As a data scientist at a home electronics company which manufactures state of the art **smart televisions**, I wanted 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.

Below are the gestures, and associated commands for the same.

* 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 of the Assignment**

Train different models (of Conv3D, and CNN+RNN), on the 'train' folder which performs well on the validation data ( 'val') folder as well.

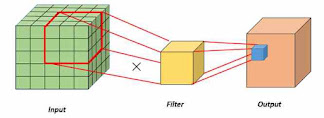
# **Model Architecture**

2 Types of architectures are recommended.

1. **3D Convolutional Network, or Conv3D**

3D convolutions are a natural extension to the 2D convolution. Just like in 2D conv, where filter is moved in two directions, in 3D conv, the filter will be moved 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.

Below figure depicts a sample conv 3D operation, where filter is applied in all 3 directions.



**Fig 1: Conv3D Filter Operation**

****

Output

Dense

Flatten

Conv3D

**Fig 2: Conv3D Architecture**

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).

Output

Dense

RNN Model  
(LSTM/GRU)

CNN Model  
(Conv2D)

****

Flatten

**Fig 3: CNN + RNN Architecture**

**Note:** Please note that, in both the above architectures, the error / backpropagation is not explicitly indicated, as it will be inherently taken care by the architectures.

# **Preprocessing**

* **Crop the images:**
  + As the provided images are of different sizes, cropping of the images is done to ensure, always the images of same so that NN can recognize the images effectively. In general, cropping was done to select the maximum possible square for the given image dimensions.
* **Resizing the images:**
  + Resize the images to defined size as per the model, so that all images are provided with same size for the model to train.
* **Normalizing the images:**
  + Normalizing the values of RGB vectors (to divide by 255), so that the it can be computationally more efficient, and control the effect of outliers.
* **Augmentation (optional):**
  + Augmentation is tried (though limited), to augment the images with blur filter (to make blurring effect), edge filter (increasing contrast on edges), sharpen filter (to enhance brighter pixels) etc.,

# **Usage of Data Generator**

As the input is video (sequence of image), the size is relatively huge, and loading / processing all the images keeping them in disk is memory intense, and it impacts performance. So, data generators are used to feed a batch of images at a time(based on batch size) and run through the network, and process next batch etc., to make it computationally effective. As part of Generator logic, cropping, resizing, and normalization of images are also performed.

# **Model Experiments**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Experiment**  **Number** | **Model** | **Total**  **Parameters** | **Result** | **Decision + Explanation** |
| **1** | **Conv3D** | 1,744,709 | Training Accuracy: 0.98  Training Loss: 0.0840  Validation Accuracy: 0.21  Validation Loss: 9.0810 | * Model was overfitting. * Reduce the Batch Size, to 30, and increase batch size to avoid overfitting problems. * As there is lot of oscillation of the val loss, lower the learning rate to 0.0003 |
| **2** | **Conv3D** | 1,948,997 | Training Accuracy: 0.92  Training Loss: 0.2981  Validation Accuracy: 0.17  Validation Loss: 3.3682 | * Though the gap between the training, and validation loss is reduced, model was still overfitting. * Number of frames reduced to 20, batch size reduced to 25, and increase the Dropout percentage. |
| **3** | **Conv3D** | 1,948,997 | Training Accuracy: 0.87  Training Loss: 0.4051  Validation Accuracy: 0.26  Validation Loss: 2.4843 | * The gap between the training, validation loss is reduced, and the validation accuracy was improved, but model was still overfitting. * Reduce the Resolution to (130,130), batch size to 20, and increase the epochs, increase dense layer neurons to 256, increasing dropouts to 0.40 (to avoid overfitting)5 |
| **4** | **Conv3D** | 2,458,309 | Training Accuracy: 0.89  Training Loss: 0.3388  Validation Accuracy: 0.74  Validation Loss: 0.5986 | * Training accuracy is slightly improved as compared to previous model, and validation accuracy is considerably increased. Moreover, the gap between training and validation loss is also reduced. Still the model is slightly overfitting. * No. of parameters are increased significantly. So, reducing the image resolution to reduce the parameters |
| **5** | **Conv3D** | 1,762,501 | Training Accuracy: 0.95  Training Loss: 0.1286  Validation Accuracy: 0.61  Validation Loss: 1.2040 | * Though training accuracy improved significantly, the validation accuracy is dropped down, and moreover, the gap between, training and validation loss is increased considerably. Model again started overfitting. * Try with Data Augmentation |
| **6** | **Conv3D**  **With Data Augmentation** | 1,762,501 | Training Accuracy: 0.52  Training Loss: 1.0849  Validation Accuracy: 0.7400  Validation Loss: 0.7240 | * Model started underfitting, as the training accuracy is dropped down in this case. * Revert to model 4, and increasing dropout rate to avoid overfitting problems. |
| **7** | **Conv3D** | 2,254,021 | Training Accuracy: 0.83  Training Loss: 0.44  Validation Accuracy: 0.76  Validation Loss: 0.64 | * Relatively better model, with the training and validation accuracy, as well as training and validation loss getting much closer. Though the model is slightly overfitting and there are mode training parameters. * Reduce the resolution further to (120, 120), and select relatively smaller frames (18), increase the epochs to allow model to stabilize. |
| **8** | **Conv3D**  **[Recommended Model]** | 1,762,501 | Training Accuracy: 0.87  Training Loss: 0.38  Validation Accuracy: 0.82  Validation Loss: 0.61 | * The best Model with relatively smaller parameters, and the accuracy of training, validation looks much closer. Even the loss also getting relatively closer. |
| **9** | **ConvLSTM** | 1,657,445 | Training Accuracy: 0.82  Training Loss: 0.51  Validation Accuracy: 0.66  Validation Loss: 0.92 | * Model is overfitting. * Increasing dropout rate to 0.35, reducing number of frames (18), and increasing the number of epochs. |
| **10** | **ConvLSTM (LSTM)** | 1,657,445 | Training Accuracy: 0.77  Training Loss: 0.64  Validation Accuracy: 0.69  Validation Loss: 0.76 | * Though, the training accuracy is slightly reduced as compared to previous model, the validation accuracy improved. Same is the case wrto loss. * Still the model is slightly overfitting. Try to check with GRU |
| **11** | **ConvGRU**  **RECOMMENDED**  **MODEL** | **1,346,405** | **Training Accuracy: 0.84**  **Training Loss: 0.45**  **Validation Accuracy: 0.78**  **Validation Loss: 0.62** | * **This produced relatively better model, and overfitting problems are also taken care of. The performance is relatively comparable with Experiment 8, with relatively fewer parameters. So, this was the recommended model** |

**Note:** Both Experiment 8, 11 are having better performance where overfitting problems are addressed. Though Experiment 8 has slightly better training and validation accuracy, and training and validation losses, as compared to Model 11, the number of parameters is significantly less in Model 11. **Hence the recommended Model is Experiment 11.**

# **Observations**

* CNN + RNN (GRU) model has better performance considering the number of parameters as compared to Conv3D, CNN + RNN(LSTM).
* As a generic observation, the number of trainable parameters has an impact on the training time, which was evident across multiple models.
* Larger batch size was always overfitting. Reducing the batch size, has improved the performance slightly better.
* The augmentation method, did not really help improving the performance (though it was tried in a limited way).

# **Scope for Improvement**

* Data Augmentation with other options (like edge enhancement, smoothen filter) etc., just to ensure better impact.
* There could have been more tuning of the hyper parameters in terms of padding stride, filter length variations (only 2, 3 kernel size filters are used in the model) to extract more features.
* Only Adam optimizer was used, tuning with help of other hyperparameters, AdaGrad etc., to improve model performance.
* Usage of transfer learning