**Gesture Recognition Case Study Analysis  
Arjun Desai, Dhaval Sadaiya  
  
Problem Statement**As a data scientist at a home electronics company that manufactures state of the art smart televisions, we want to develop a cool feature in the smart-TV that can recognize five different gestures performed by the user that will help users control the TV without using a remote.

1. Thumbs Up: Increase the volume

2. Thumbs Down: Lower the volume

3. Left Swipe: Fast Forward 10 seconds

4. Right Swipe: Fast Backward 10 seconds

5. Stop: Pause the movie

**Objectives**We intend to address the problem statement by achieving the following objectives:  
1. Generator: generators produce a stream of values, so yielding a single value doesn’t really qualify as a stream. To do this, we can actually put in multiple yield statements into a generator function. These yield statements form the sequence that the generator will output. Our generator takes a batch of videos as input without any error. Steps like cropping, resizing and normalization have been performed here.  
2. Model: We have develop models that are able to train without any errors on the total number of parameters and optimized them for the accuracy achieved on the validation dataset.

**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*.*

The data is in a zip file. The zip file contains a 'train' and a 'val' folder with 2 csv files each for 'train' and 'val' folder. These folders are in turn divided into subfolders which belong to a video/sequence of a particular gesture. These subfolders contain 30 images/frames for the gesture. Note that all the images in a particular video subfolder have the same dimensions but the different videos have different dimensions. These have dimensions either 360x360 or 120x160. Hence, it would require you to do some pre-processing.

Each row of the csv file has the name of the subfolder containing the frames/images of the particular gesture, the name of the gesture and the label assigned to the gesture. You have to develop a model that trains on the 'train' folder and gives a high accuracy on the 'val' folder. We have withheld the test folder for evaluation purposes. The train and validation folders, in turn, have folders corresponding to the individual videos.

**Pre-processing**

1. Crop and Resize: As there are two different dimensions of frames provided 360 X 360 and 140 X 160, we need to resize and crop images. We have handled this in each of the models during the experimentation in order to see the impact of the of the resizing on the model results.
2. Normalization: We have handled the normalization of the RGB values in order to get rid of distortions caused by lights and shadows in the images.
3. Augmentation: We have slightly rotated the pre-processed images of the gestures in order to increase the train data, however we rotated the images only to a certain extent in order to avoid changing the interpretation of the gestures.

**Model Architecture**

For analysing videos using neural networks, we have used two types of architectures. 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 that we have used to process videos is a natural extension of CNNs - a 3D convolutional network.

Convolutions + RNN: 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).

3D Convolutional Network, or 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.

**Results and Observations**

|  |  |  |  |
| --- | --- | --- | --- |
| **Experiment Number** | **Model** | **Result** | **Decision + Explanation** |
| 0 | Conv3D | Throws Generator error | Crop the images correctly, try to overfit on less amount of data |
| 1 | Conv3D | Training Accuracy: 0.99 Validation Accuracy: 0.79 | Base Model, it seems to be clearly overfitting |
| 2 | Conv3D | Training Accuracy: 0.85 Validation Accuracy: 0.76 | Adding dropout layers - Batch Size = 20 and No. of Epochs = 25 |
| 3 | Conv3D | Training Accuracy: 0.77 Validation Accuracy: 0.80 | Reduce filter size to (2,2,2) and image res to 120 x 120, - Batch Size = 30 and No. of Epochs = 25 |
| 4 | Conv3D | Training Accuracy: 0.82 Validation Accuracy: 0.79 | Adding more layers - Batch Size = 20 and No. of Epochs = 25 |
| 5 | Conv3D | Training Accuracy: 0.82 Validation Accuracy: 0.79 | Adding dropout at convolution layers |
| 6 | Conv3D | Training Accuracy: 0.79 Validation Accuracy: 0.71 | Reducing the number of parameters |
| 7 | CNN- LSTM | Training Accuracy: 0.88 Validation Accuracy: 0.74 | Base Model, we notice slight overfitting again |
| 8 | CNN- LSTM | Training Accuracy: 0.82 Validation Accuracy: 0.77 | Reducing network parameters and adding augmentation |
| Final Model | CNN- LSTM | Training Accuracy: 0.96  Validation Accuracy: 0.95 | Transfer Learning with GRU and training all weights |

We finalized the CNN LSTM with GRU model using Transfer Learning after adding drop-out convolution layers, reducing the number of parameters as well as adjusting the hyper-parameters on our base model. The final model:

* Returned a training accuracy of 96% and a validation accuracy of 95%
* Had a higher number of trainable parameters: 3,668,933

We used various model configurations, tuned hyper-parameters while shuffling the number of iterations, batch sizes, dimensions. We utilized batch normalization, added drop-out layers and pooling to address the challenge of overfitting on our base models. The final model utilizes transfer learning using a pre-trained Resnet before passing it through a Softmax layer for gesture recognition. The overall accuracy of the model was significantly boosted using Transfer Learning. Batch size was found to be inversely proportional to the model accuracy in several cases and directly proportional to the compute resources. Hence, we have finalized the batch size in order to strike a balance between model performance and accuracy. While using GRU instead of LSTM enabled faster computations, we observed better validation accuracies with the latter and hence finalized the model with a focus on accuracy.