**Deep Learning Course Project – Gesture Recognition**

Abhishek Jain

Ramya Sree Amaravathi

**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.

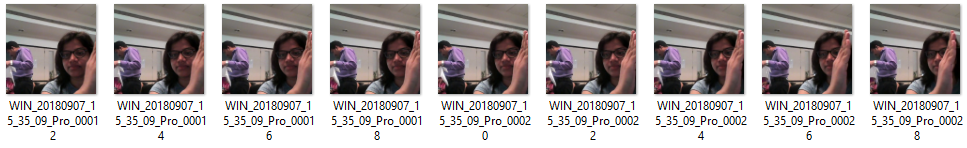
The gestures are continuously monitored by the webcam mounted on the TV. Each gesture corresponds to a specific command:

| **Gesture** | **Corresponding Action** |
| --- | --- |
| 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. |

Each video is a sequence of 30 frames (or images).

**Understanding the dataset:**

The training data consists of a few hundred videos categorized 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 webcam – similar to what the smart TV will use.



**Objective:**

* **Generator**: 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.
* **Model**: Develop a model that can train without any errors which will be judged on the total number of parameters (as the inference(prediction) time should be less) and the accuracy achieved. As suggested by Snehansu, start training on a small amount of data and then proceed further.
* **Write up**: This should contain the detailed procedure followed in choosing the final model. The write up should start with the reason for choosing the base model, then highlight the reasons and metrics taken into consideration to modify and experiment to arrive at the final model.

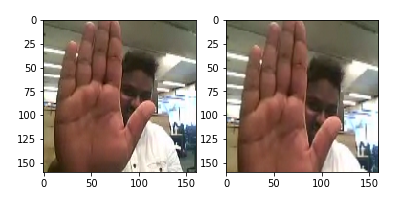
**Commonly used architectures for analysing videos:**

* **Convolutions+RNN**: The conv2D 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 .
* **3D Convolutional Network or Conv3D**:3D convolutions are a natural extension to the 2D convolutions. In 3D we move the convolutions in three directions. Mostly used on 3D image data.

**Steps followed:**

* Loading necessary libraries and defining data path.
* **Generator**: Initialized path of the data file, image properties, batch size, frames to sample and number of epochs.
* Created sample model with given batch size and other properties.
* **Data Preprocessing**: The steps below are performed during data preprocessing.

1. Resizing and cropping of the images. This was mainly done to ensure that the NN only recognizes the gestures effectively rather than focusing on the other background noise present in the image.
2. Normalization of the images. Normalizing the RGB values of an image can at times be a simple and effective way to get rid of distortions caused by lights and shadows in an image.



* **Experimenting with image resolution, batch size and number of frames to use**: Image resolution and number of frames in sequence have more impact on training time than batch\_size, So we can consider the Batch Size around 15-40 and We changed the resolution 160X160, 120X120 according the model performance.
* It was taken into consideration that we don’t rotate images to a greater extent as this would change the meaning of the gestures completely.

**NN Architecture development and training:**

* 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 also played around with different learning rates and ReduceLROnPlateau was used to decrease the learning rate if the monitored metrics (val\_loss) remains unchanged in between epochs.
* We experimented with SGD() and Adam() optimizers but went forward with Adam() as it lead to improvement in model’s accuracy by rectifying high variance in the model’s parameters.
* 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
* Early stopping was used to put a halt at the training process when the val\_loss would start to saturate / model’s performance would stop improving

**Observations:**

* It was observed that as the Number of trainable parameters increase, the model takes much more time for training.
* Increasing the batch size greatly reduces the training time but this also has a negative impact on the model accuracy. This made us realise that there is always a trade-off here on basis of priority -> If we want our model to be ready in a shorter time span, choose larger batch size else you should choose lower batch size if you want your model to be more accurate
* CNN+LSTM based model did not perform better than Conv3D. As per our understanding, this is something which depends on the kind of data we used, the architecture we developed and the hyper-parameters we chose
* The table below shows the complete details of experiments with results, explanation, parameters, and best epochs for each model.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Experiment** | **Result** | **Explanation** | **# Parameters** | **Best Epoch** |
| **CONV3D** | **0** | **Error** | **Reduce the batch Size and Reduce the # if neurons in dense layer** | **-** | **-** |
| **1** | **Train Accuracy: 99% Validation Accuracy: 77%** | **Model is Clearly Over fitting, lets add dropouts** | **1,117,061** | **10** |
| **2** | **Train Accuracy: 87% Validation Accuracy: 82%** | **Overfitting is reduced but significant  increase in number of parameters, Lowering learning rate** | **3,638,981** | **22** |
| **3** | **Train Accuracy: 76% Validation Accuracy: 75%** | **Overfitting has reduced but accuracy is also reduced, adding more layers in next step** | **1,762,613** | **25** |
| **4** | **Train Accuracy: 76% Validation Accuracy: 77%** | **Didn’t see much improvement in performance, adding dropouts next** | **2,556,533** | **15** |
| **5** | **Train Accuracy: 89% Validation Accuracy: 77%** | **not seeing improvement, reducing no of parameters next** | **2,556,533** | **14** |
| **6** | **Train Accuracy: 89% Validation Accuracy: 76%** | **Comparable accuracy with a much lighter model, reducing more parameters** | **696,645** | **19** |
| **7** | **Train Accuracy: 86% Validation Accuracy: 74%** | **Very similar accuracy with last model still not best accuracy** | **504,709** | **20** |
| **CNN+LSTM** | **8** | **Train Accuracy: 91% Validation Accuracy: 76%** | **CNN LSTM best accuracy of 76% it’s not performing well, applying some data augmentation tricks with conv3D** | **1,657,445** | **18** |
| **CONV3D** | **9** | **Train Accuracy: 79% Validation Accuracy: 77%** | **Very less difference in Train and test values is good learning power** | **3,638,981** | **15** |
| **10** | **Train Accuracy: 81.7% Validation Accuracy: 81.0%** | **By far the best set of accuracy with moderate number of parameters** | **1,762,613** | **23** |

**Final Comments:**

* Finally we choose the 10th model of Conv3D kind as it had the best accuracy.
* We can further improve by experimenting more with CNN+LSTM/GRU and Transfer learning .
* Improvement can also be done by more augmentation techniques on the data and also doing more hyper parameter tuning.

--End--