**Gesture Recognition**

**By Pramod S and Rakshith K R**

**Problem Statement:**

A home electronics company wants to develop a cool feature to their smart TV that can recognize five different gestures performed by the user which helps user control the TV without remote. A model which can accurately detect the five gestures has to be developed.

The five gestures are as follows:

* 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:**

Dataset consists of total 773 videos, 663 for train and 100 for validation. These videos are categorized into one of the five classes. Each video is divided into sequence of 30 frames. 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. Specifically, videos have two types of dimensions - either 360x360 or 120x160.

**Methodology:**

Generator function:

Creating a function that generates required batch of sequences (videos) with required number of frames per video. The frames are preprocessed as follows inside the generator function

1. Cropping the image to capture most part of the gesture. There are two types of video sizes 360X360 and 120X160. Cropping has been done differently for each of the video types after inspecting few frames from each of them respectively
2. The images are resized to a common size. Different image sizes like 81X81X3, 80X80X3, 112X112X3 & 120X120X3 were tried to get the best model. For final model the frame size is set to 80X80X3 (same can be seen in the jupyter notebook)
3. After resizing, the images are finally normalized

Model Building and Training:

Conv3D and CNN+RNN are the two types of architectures used for video classification tasks. 3D convolutions apply a 3-dimensional filter to the dataset and the filter moves in 3-directions (x, y, z) to calculate the low-level feature representations. CNN+RNN uses 2D convolutional layers followed by a GRU or LSTM layers. In case of CNN+RNN, both custom CNN and transfer learning like VGG16 with imagenet weights are used to extract features before passing it to RNN.

Initially a baseline model was built to check if the generator function is functional and model is running with no errors. Baseline model had two 3D convolutional layers with 32 feature maps and 64 feature maps respectively and one dense layer followed by an output softmax layer. Many experiments were conducted to get an optimal model in terms of good performance with less parameters and memory. In these experiments mainly image size, batch size, filter/kernel sizes and architectures were changed to get the optimal model. Batch size was also limited after few experiments due to low availability of memory (out of memory). Also, number of epochs were fixed to 30due to huge computational time. Total of 18 frames per video were taken starting from 6th to 24th frame of each video (This was taken after looking at few of the videos and gesture transition timing). The table below shows the brief summary of each of the experiment conducted

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Experiment | Batch size | Image dimension | Model | Total trainable parameters | Decision | Remarks | Train acc. | Val acc. |
| 1 | 32 | 81X81 | **Conv3D** | 2,049,605 | Check if model runs | Generator is functional and model is able to learn. Model is overfitting for just 10 epochs | 0.9 | 0.24 |
| 2 | 64 | 81X81 | 2,049,605 | Adding dropouts to reduce overfitting. Increasing the epochs to 30 and batch size to 64. Changed solver from Adam to SGD | Time taken per epoch is almost same compared to batch size of 32. Dropouts reduced overfitting a bit but, validation accuracy is not yet improved | 0.65 | 0.36 |
| 3 | 128 | 80X80 | 3,558,085 | Adding one more layer and changing filters sizes. Increasing batch size to 128 | Out of memory error |  |  |
| 4 | 64 | 80X80 | 3,558,085 | Changing batch size back to 64 | Stopped at epoch 21/30 since validation accuracy was not increasing. Adding a layer and changing filters did not help in improving accuracy | 0.64 | 0.25 |
| 5 | 64 | 120X120 | 7,738,781 | Changed image size. Removed extra layer and modified filter sizes | Out of memory error |  |  |
| 6 | 32 | 120X120 | 7,738,781 | Reduced batch size to 32 | Accuracy of both training and validation has improved | 0.73 | 0.71 |
| 7 | 32 | 112X112 | 4,875,845 | Reduced image size and changed kernel sizes to reduce parameters | Accuracy has increased. But the difference between train and validation is around 7% | 0.89 | 0.82 |
| 8  (Final Model) | 32 | 80X80 | 3,745,733 | Reduced image size to reduce the parameters and changed dropout value from 0.3 to 0.35 to reduce overfitting | Now the model is generalized and has a decent accuracy with a smaller number of parameters compared to previous model | 0.85 | 0.83 |
| 9 | 32 | 80X80 | **CNN+RNN (GRU)** | 301,701 | CNN (2 Convolutional layers) and RNN (1 GRU layer) | Model has overfitted | 0.93 | 0.7 |
| 10 | 32 | 80X80 | 301,701 | Adding dropouts to reduce overfitting | Highest accuracy was seen in 26/30 epoch. Overfitting has reduced but, validation accuracy has not improved | 0.83 | 0.7 |
| 11 | 32 | 80X80 | 1,202,437 | Increased number of feature maps for convolutional layers from 16 to 32 for 1st layer and 32 to 64 for 2nd layer | Validation accuracy slightly increased. But model is overfitting | 0.92 | 0.72 |
| 12 | 32 | 80X80 | 4,801,029 | Further increased the feature maps and one more dropout layer | Adding more feature maps didn’t work out | 0.85 | 0.65 |
| 13 | 32 | 80X80 | **CNN+RNN (LSTM)** | 1,300,229 | Reduced feature maps and using LSTM instead of GRU | Validation accuracy has not improved | 0.86 | 0.69 |
| 14 | 32 | 80X80 | **Transfer Learning (VGG16) + RNN** | 725,509 | Using VGG16 + RNN | Validation accuracy has improved but, training accuracy is lower than validation accuracy | 0.74 | 0.81 |
| 15 | 32 | 80X80 | 3,007,503 | Making last one layer of VGG16 trainable | Highest validation accuracy of 85% seen in 14/30 epoch | 0.92 | 0.85 |

Model selection:

Transfer learning followed by RNN has the highest validation accuracy among the architectures (experiment 15 highlighted in orange). Even though the number of trainable parameters is low (3,007,503) the memory of the final model is ~98mb. The model with second highest accuracy is conv3D with 3,745,733 trainable parameters and model memory of ~29mb (experiment 8 highlighted in green). In case of custom CNN followed by RNN, the highest validation accuracy obtained is around 72% (experiment 11 highlighted in blue).

It is observed that even though the transfer learning (VGG16) followed by RNN is performing better than other models, the memory used by the model is very high. It can be seen that the memory is almost three times more compared to the conv3D model with just 2% of difference in validation accuracy. The conv3D has better performance in terms of accuracy and memory which is very much suitable for our application. Hence, conv3D (experiment 8 model) is taken as the final model for this application. The architecture for the final model is

conv3D-32🡪maxpool3D🡪conv3D-64🡪maxpool3D🡪dense-192🡪dense-5-softmax

Observations and Conclusion:

* Image size increases the number of trainable parameters. These parameters can be reduced by changing the filter sizes
* For the same model, the batch size of 32 and 64 had similar run time per epoch
* Due to Out of Memory error batch size was later limited to 32
* Number of epochs can also be experimented to increase the accuracy of models. Due to computational time number of epochs were fixed to 30
* CNN+RNN uses fewer parameters (experiments 9 to 13) and memory. But the validation accuracy is not good compared to other models. This can be improved by trying different CNN or conv2D layers which extracts better features before passing to RNN
* Transfer learning followed by RNN surely increases the accuracy with memory trade off. Other transfer learning CNNs apart from VGG16 can be tried out

Note:

The final juypter notebook has the conv3D model (experiment 8 final model), CNN+RNN model (experiment 13) and transfer learning + RNN model (experiment 15).