Deep Learning Case Study – Gesture Recognition

Pradeep Kumar S – Group Facilitator
Manoj Romina

Problem Statement

Imagine we are working as a data scientist at a home electronics company which 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 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

We have been provided with both training and validation datasets as separate folders. The training data consists of a several videos categorized into one of the five gesture classes. Each video is typically around 2-3 seconds long and 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 train and validation folders have 2 CSV files each. These folders are sub divided in to folders where each sub folder represents a video of a specific gesture with 30 frames each.

Objective

The main objective of this case study is to train different models on the train folder and validate on the validation folder to check the performance of the same. The trained model is then is used to predict the action performed in each sequence or video. The Final test folder for evaluation is withheld and the final model is tested on it.

Architectures

The two suggested architectures in this case study for video analysis using deep learning networks are:

1. 3D Convolutional Neural Networks (Conv3D)
2. CNN + RNN architecture

3D Convolutional Neural Networks (Conv3D)

3D convolutions are a natural extension to the 2D convolutional neural networks. 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).

CNN + RNN Architecture

The 2D convolutional network (conv2D) network will extract a feature vector for each image, and a sequence of these feature vectors is then fed to a recurrent neural network (RNN)-based network. The output of the RNN is a regular SoftMax function.

The commonly used recurrent networks are Long-Short Term Memory networks (LSTM) and Gated Recurrent Units (GRU).

Modelling

The approach of model building is listed below:

Generators

The data ingestion pipeline is set up using the generators. Building a data generator is the most important part of building a data pipeline. In the generator, we are going to pre-process the images as we have images of 2 different dimensions of 360x360 and 120x160. This generator helps us in taking a batch of videos as input without any errors and memory overloading. Few important pre-processing like resizing, cropping and normalisation is performed successfully in this step.

Data Pre-Processing

- **Resizing and cropping** - Resizing and cropping of the images is mainly done to ensure that the network only recognizes the gestures effectively rather than focusing on the other background noise present in the image.
- **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.
- **Data Augmentation** – This is an important step in the later stages of model development for improving the model’s accuracy. We have slightly rotated the pre-processed images of the gestures in order to increase data for the model to train on and make it more generalizable in nature as the positioning of the hand won’t necessarily be within the camera frame always.

Model Building

3D Convolutional Networks

- The Base model was first built with simplified convolutional and dense layers in order to get it overfitted. This is done in order to check if the model is able to learn all the patterns from the data provided.

- We then proceed to experiment with various layer combinations or model configurations along with hyper parameters tuning. The model is executed for various iterations (epochs) and combinations of batch sizes, image dimensions, filter sizes, padding and stride length.
- We also experimented with varied learning rates using 'ReduceLROnPleateau' which was decreasing the learning rate based on the monitoring of validation loss.
- We chose Adam as the model optimizer as it led to the improvement in model's accuracy by rectifying high variance in the model's parameters.
- We used pooling and Dropout layers to prevent the model from overfitting.
- We used early stopping to stop the training process when the validation loss started saturating without any further improvements.

The Final best performing model had 4 convolutional layers and 2 dense with 2 dropout layers. This model had 1.9 million parameters which resulted in the best **training accuracy of 96.8%** and **validation accuracy of 91%**.

CNN + RNN

- The Base model was first built with simplified convolutional, GRU and dense layers in order to get it overfitted. This is done in order to check if the model is able to learn all the patterns from the data provided.
- We then proceed to experiment with various layer combinations or model configurations along with hyper parameters tuning. The model is executed for various iterations (epochs) and combinations of batch sizes, image dimensions, filter sizes, padding and stride length.
- We also experimented with varied learning rates using 'ReduceLROnPleateau' which was decreasing the learning rate based on the monitoring of validation loss.
- We chose Adam as the model optimizer as it led to the improvement in model's accuracy by rectifying high variance in the model's parameters.
- We used pooling and Dropout layers to prevent the model from overfitting.
- We used early stopping to stop the training process when the validation loss started saturating without any further improvements.

The Final best performing model had 4 convolutional layers, 1 GRU and 2 dense with 2 dropout layers. This model had 0.3 million parameters which resulted in the best **training accuracy of 92.9%** and **validation accuracy of 86%** for second last epoch.

Observations

3D Convolutional Networks

- We observed that a model of simple configuration performed much better than complex models with higher number of hidden layers.
- We observed that the higher the parameters of the model, higher the training time.
- We observed that increasing the batch size resulted in the decrease of the accuracy. Lowe batch size performed pretty well. The batch size we fixed was 20.
- Overfitting issue was tackled by data augmentation and early stopping.
- The best combination of image properties was identified as 120x120 resolution, choosing 20 frames instead of the entire 30 frames resulted in higher accuracy of our model.
- The best hyper parameters identified were training iterations or epochs of 20-30 and batch size of 20 which resulted in higher accuracy and stable model.

- We observed that Batch Normalisation was significantly impacting our model's performance. Removal of the same resulted in some massive improvements in terms of validation accuracy.

The table of the models tried are given below:

Model	Data Augmentation	Model Parameters	Results – Categorical Accuracy		Comments and Observations
Base Model (Conv3DModelBase)	No	9,901,701	Train	93.06%	Base model is Overfitting. We start experimenting with hyperparameters tuning.
			Val	75.00%	
Model A (conv3d_exp_2)	No	9,901,701	Train	61.24%	Changing number of frames didn't improve the model.
			Val	43.00%	
Model B (conv3d_exp_4)	No	9,901,701	Train	20.66%	This was the worst performing model with batch size of 30.
			Val	21.00%	
Model C (conv3d_exp_6)	No	9,901,701	Train	89.89%	Change in image resolution to 120x120 and batch to 20 made significant improvement.
			Val	90.00%	
Model D (conv3d_exp_8)	Yes	9,905,352	Train	71.72%	Data Augmentation showing some great improvements in the model with same parameters.
			Val	63.00%	
Model E (conv3d_exp_9)	Yes	9,905,352	Train	69.46%	Overfitting on augmented data if batch size is increased to 40.
			Val	53.00%	
Model F (conv3d_exp_10)	Yes	9,905,352	Train	74.74%	Significant improvement in model with augmented data with same parameters as model C.
			Val	74.00%	
Model 1 (Conv3D1Model)	No	3,757,701	Train	100%	Adding a dropout later, slightly reduced overfitting but still can be improved.
			Val	88.00%	
Model 2 (Conv3D2Model)	No	3,757,701	Train	99.70%	Adding an extra dense layer didn't improve the model. Going for augmented data.
			Val	89.00%	
Model 2a (Conv3D2Model - augmentation)	Yes	3,757,701	Train	85.60%	Using augmented data with previous model significantly improved the accuracy overall.
			Val	78.00%	
Model 3 (Conv3D3Model)	No	3,708,149	Train	99.95%	Reducing filter size resulted in the model overfitting again.
			Val	84.00%	
Model 4 (Conv3D4Model)	Yes	693,637	Train	89.89%	Adding an extra convolutional layer reduced overfitting.
			Val	81.00%	
Model 5 **(Conv3D5Model)**	**Yes**	**1,931,269**	**Train**	**97.36%**	**Adding an extra dense layer gave us the best performing model.**
			Val	**95.00%**	

CNN + RNN

- We observed that a model of simple configuration performed much better than complex models with higher number of hidden layers.
- We observed that the higher the parameters of the model, higher the training time.
- We observed that increasing the batch size resulted in the decrease of the accuracy. Lowe batch size performed pretty well. The batch size we fixed was 30.
- Overfitting issue was tackled by data augmentation and early stopping.
- The best combination of image properties was identified as 120x120 resolution, choosing 20 frames instead of the entire 30 frames resulted in higher accuracy of our model.
- The best hyper parameters identified were training iterations or epochs of 20-30 and batch size of 30 which resulted in higher accuracy and stable model.
- We observed that Batch Normalisation was significantly impacting our model's performance. Removal of the same resulted in some massive improvements in terms of validation accuracy.

Model	Data Augmentation	Model Parameters	Results – Categorical Accuracy		Comments and Observations
Base Model (CNN_RNN_ModelBase)	No	1,243,269	Train	99.10%	Base model is Overfitting. We start experimenting with hyperparameters tuning.
			Val	71.00%	
Model A (cnn_rnn_exp_1)	No	1,243,269	Train	75.57%	Changing number of frames didn't improve the model.
			Val	60.00%	
Model B (cnn_rnn_exp_2)	No	705,669	Train	64.60%	Change in image resolution to 120x120 and batch to 20 made significant improvement.
			Val	58.00%	
Model C (cnn_rnn_exp_3)	No	705,669	Train	71.89%	Overfitting if batch size is increased to 40.
			Val	51.00%	
Model D (cnn_rnn_exp_4)	Yes	705,669	Train	62.72%	Data Augmentation showing some great improvements in the model with same parameters.
			Val	57.00%	
Model E (cnn_rnn_exp_5)	Yes	9,905,352	Train	98.946%	Increasing epoch to 20 resulted in overfitting
			Val	77.00%	
Model F (cnn_rnn_exp_5a)	Yes	705,669	Train	74.28%	Increasing epoch to 20 and augmenting data resulted in reducing overfitting
			Val	64.00%	
Model 1 (CNN_RNN1_Model)	No	348,421	Train	98.94%	Adding an extra CNN and a dropout later for Dense, didn't reduced overfitting.
			Val	76.00%	
Model 2	No	348,421	Train	97.29%	

(CNN_RNN2_Model)			Val	79.00%	Adding Dropout for GRU Layer didn't improve the model
Model 3 **(CNN_RNN3_Model)**	**No**	**383,493**	**Train**	**92.91%**	**Adding A Dense Layer with Dropout and Removed Dropout from GRU Layer. Got best result for 19th epoch.**
			Val	**86.00%**	
Model 4 **(CNN_RNN4_Model)**	No	324,357	Train	83.71%	Adding a CNN layer reduced overfitting.
			Val	70.00%	

Final Observation – Best Model

The 3D convolutional neural network (**Conv3D5Model**) model with 4 convolution layers and 2 dense with accompanying dropout layers produced the best performing model using image properties like a resolution of 120x120, batch size of 20 and sampling frames of 20.

This resulted in an overall **training accuracy of 96.8%** and **validation accuracy of 91%.**

Further Improvements

- **Hyperparameters Tuning**: Using various functions like ReLU, Leaky ReLU, tanh and sigmoid functions along with optimizers like AdaDelta, RMSprop etc., Tuning padding and stride length also can significantly improve model performance.
- **Using Transfer Learning**: Using pre trained models like MobileNet, GoogleNet, ResNet provides significant boost to the model due to pre trained weights that we use associated with these models.
- **Gated Recurrent Units**: Using GRU instead of LSTM will significantly improve the training time with lesser number of parameters even though there might be a negligible decrease in the accuracy.