Neural Network Project - Gesture Recognition

12-01-2022

Created by:
Anik Chakraborty (waytoanik@outlook.com)
Manish Kumar (manish.arya21@gmail.com)

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 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

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 a webcam - like what the smart TV will use.
Task is to train a model on the 'train' folder which performs well on the 'val' folder as well (as usually done in ML projects). We have withheld the test folder for evaluation purposes - your final model's performance will be tested on the 'test' set.

Approach:

Two architectures used:

For analyzing videos using neural networks, two types of architectures are used commonly.

- ### 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. For video data, each data point is IID, but in each datapoint images are in sequence. We perform Conv2D on images, here we have temporal dimension as the third dimension. Just like in 2D conv, we move the filter in two directions (x and y), in 3D conv, we move the filter in three directions (x, y and z). So, for 3D conv cubic or 3-D kernals are used.

  Considering a video of 30FPS (Frame per seconds). One second of the video can be represented as 30 sequences of images. If each image or frame is of shape 100x100x3 (3 is no. of channels), then the video becomes a 4-D tensor of shape

  100x100x3x30 which can be written as (30x100x100)x3 where 3 is the number of channels and 30 is no. of sequences in each datapoint. This is the input tensor shape of Conv 3D network. Cubic filters are used to convolve and create the feature maps. There are 3D Pooling layers to reduce the dimension. Followed by flattening and then it goes to a Feed Forward network with softmax activation at outer layer for classification scenarios.

Data Generator Function

We have created custom data generator function. It works like a generator object. It takes batch size as input and instead of loading all training datapoints at once, it reads datapoints based on the value of the batch size and then performs required image pre-processing and yields data for training. We have performed required resizing of images. As we have different image sizes in dataset, we resized all images to 100x100 with 3 channels. Performed Normalizations for each channel separately using min-max scaling. Tested the data generator function to check if it yields the data properly.

Model Building and Experiments:

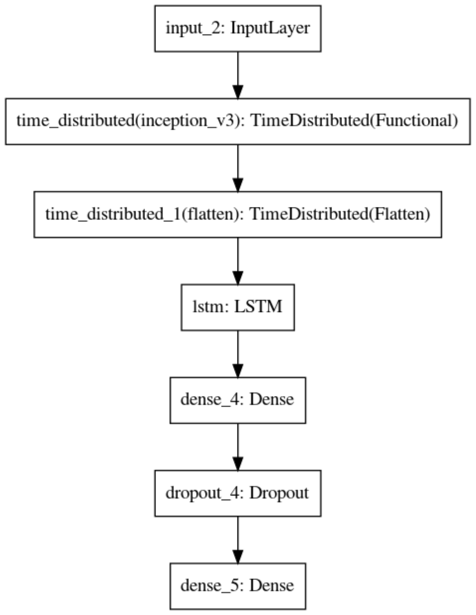
Experiment No.	Model No.	Model Architecture	Decision + Explanation	Best Training Accuracy	Best Validation Accuracy	Total Training Epochs	Best Weights Epoch	Final Batch Size	Total Parameters	Overall Model Rank
1			Finetuned InceptionV3 model, used imagenet weights and trained from layer 249 to end layer.							
2			Tried to increase batch size and able to train with batch size 128							
3			Usedhe_normal weight initialization for dense layers having ReLU activation. It help to converge gradients faster and makes the network stable.							
4	Model 1	InceptionV3 + LSTM	Used ReduceLROnPlateau and ModelCheckpoint callbacks and trained for 75 epochs with adam as optimizer. Training loss became almost zero, model shows sign of overfitting.	0.998	0.72	75	39	128	24297253	6
5			Instead of LSTM, now used GRU with InceptionV3. Kept other things as it is.							
6	Model 2	InceptionV3 + GRU	two callbacks. Training stoppped after 17th epoch. Training accuracy reached 1 and training loss almost 0, but no significant improvements in validation accuracy. Model is still overfitting.	1	0.59	17	17	128	23707941	9
7			Used previous model architecture and added higher Dropout rate, and L2 Regularization in dense layers and initially trained for 30 epochs.							
8	Model 3	InceptionV3 + GRU	Validation accuracy was improving, so later on loaded the model from saved checkpoint model file of 30th epoch and trained for another 30 epochs. Validation accuracy is bit better than previous, but still showing sign of overfitting.	0.989	0.68	60	30	128	23707941	7
9			Got OOM error during first run.							
10	Model 4	ResNet50V2 + GRU	Reduced batch size to 64 and re-run for 50 epochs. Fine-tuned ResNet50V2 model, used imagenet weights and trained layers from 154 to end layer. Got validation accuracy better than previous models/experiments. Still there is some overfitting and room for improvement.	0.998	0.81	50	26	64	36266245	5
11			Got OOM error during first run, so reduced batch size to 32							
12	Model 5	Xception + LSTM	Finetuned InceptionV3 model, used imagenet weights and trained from layer 114 to end layer, added Dropout and L2 Regularization. Got bestter categorical_accuracy in compared to previous CNN+RNN based models.	0.95	0.85	30	7	32	40133165	4
13			Built a Conv3D architecture with BatchNormalization, Dropout, Dense layers with he_normal initialization and trained for 30 epochs.							
14	Model 6	Conv3D	Validation accuracy was improving, so later on loaded the model from saved checkpoint model file of 30th epoch and trained for another 30 epochs. Even after 60 epochs model did not overfit also validation categorical_accuracy is not that good. We might need to recheck our Conv3D net architecture.	0.8	0.66	60	16	48	35923957	8
15			Added more conv3D layers, incraesed no. of filters, added more non linearity by using more ReLu layers, reduced model parameters by using more MaxPooling3D.							
16			Added one more dense layer Added dropout layers after CNN part. After redesigning the architecture trained the model for 30 epoch. Network performance looked better than previous network.							
17	Model 7	Conv3D	Accuracy was still improving and loss was decreasing, so later on loaded the model from saved checkpoint model file of 30th epoch and trained for another 30 epochs. This architecture shown the best categorical_accuracy and minimum loss. This network architecture has least number of parameters (almost 1/10th of CNN+RNN architectures) also produced better accuracy.	0.95	0.93	60	55	64	2817477	1
18			Model 7 with Nadam optimizer. Trained for 60 epochs.							
19	Model 8	Con3D	Loaded 60th epoch weight and further trained for 30 more epochs	0.97	0.9	90	61	64	2817477	2
20	Model 9	Con3D	Model 7 with RMSprop as optimizer	0.98	0.88	90	45		2817477	3

Model Architectures in details:

Approach 1: Conv2D + RNN

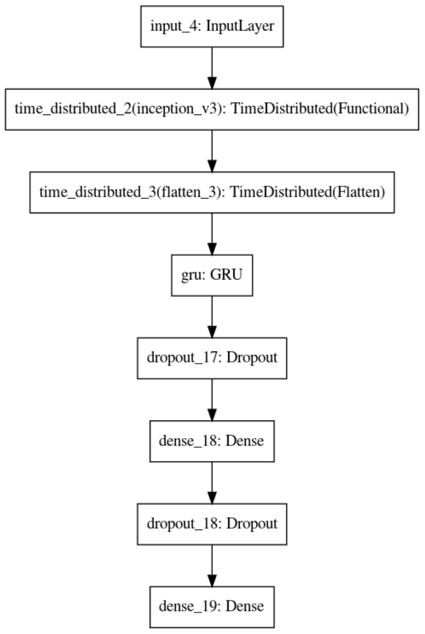
Model 1: InceptionV3 (Finetuning) + LSTM

- Total params: 24,297,253
- Trainable params: 13,609,349
- Non-trainable params: 10,687,904
- **Best Epoch:** 39
- **Training: loss= 0.006, categorical_accuracy= 0.9985**
- **Validation: loss= 1.457, categorical_accuracy= 0.72**


input_2: InputLayer
time_distributed(inception_v3): TimeDistributed(Functional)
time_distributed_1(flatten): TimeDistributed(Flatten)
lstm: LSTM
dense_4: Dense
dropout_4: Dropout
dense_5: Dense

Model 2: InceptionV3 (Finetuning) + GRU

- Total params: 23,707,941
- Trainable params: 13,020,037
- Non-trainable params: 10,687,904
- **Best Epoch:** 17
- **Training: loss= 8.4316e-04, categorical_accuracy= 1.00**
- **Validation: loss= 2.2937, categorical_accuracy= 0.59**

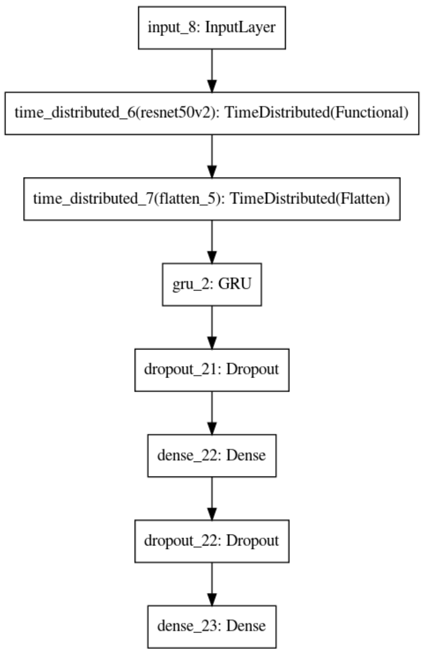

input_4: InputLayer
time_distributed_2(inception_v3): TimeDistributed(Functional)
time_distributed_3(flatten_3): TimeDistributed(Flatten)
gru: GRU
dropout_17: Dropout
dense_18: Dense
dropout_18: Dropout
dense_19: Dense

Model 3: InceptionV3 (Finetuning) + GRU with higher Dropout, L2

- Total params: 23,707,941
- Trainable params: 13,020,037
- Non-trainable params: 10,687,904
- **Best Epoch:** 30 of first run
- **Training: loss= 0.8340, categorical_accuracy= 0.9894**
- **Validation: loss= 2.1870, categorical_accuracy= 0.6800**

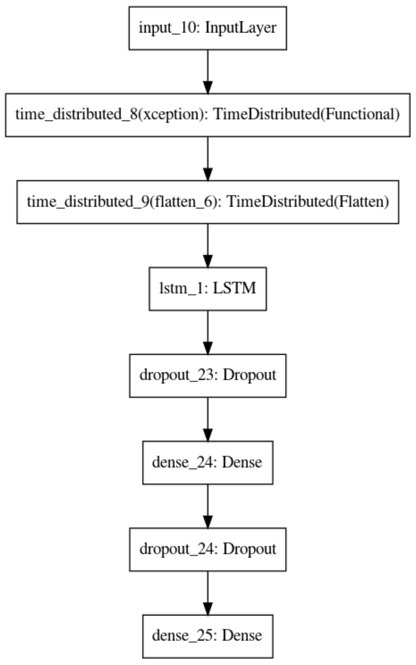
Model 4: ResNet50V2 (Finetuning) + GRU with Dropout and L2 Regularization

- Total params: 36,266,245
- Trainable params: 27,672,325
- Non-trainable params: 8,593,920
- **Best Epoch:** 26
- **Training: loss: 0.7449, categorical_accuracy: 0.9985**
- **Validation: val_loss: 1.4857, val_categorical_accuracy: 0.810**


input_8: InputLayer
time_distributed_6(resnet50v2): TimeDistributed(Functional)
time_distributed_7(flatten_5): TimeDistributed(Flatten)
gru_2: GRU
dropout_21: Dropout
dense_22: Dense
dropout_22: Dropout
dense_23: Dense

Model 5: Xception (Finetuning) + LSTM with Dropout and L2 Regularization

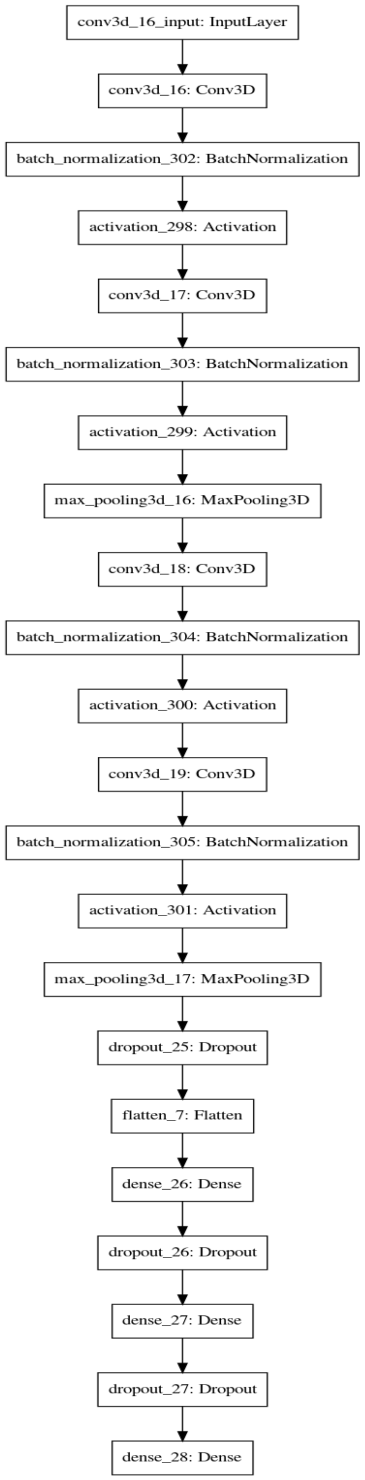
- Total params: 40,133,165
- Trainable params: 26,061,525
- Non-trainable params: 14,071,640
- Best Epoch: 7
- **Training: loss: 2.0037 - categorical_accuracy: 0.9548**
- **Validation: val_loss: 2.2650 - val_categorical_accuracy: 0.8500**


input_10: InputLayer
time_distributed_8(xception): TimeDistributed(Functional)
time_distributed_9(flatten_6): TimeDistributed(Flatten)
lstm_1: LSTM
dropout_23: Dropout
dense_24: Dense
dropout_24: Dropout
dense_25: Dense

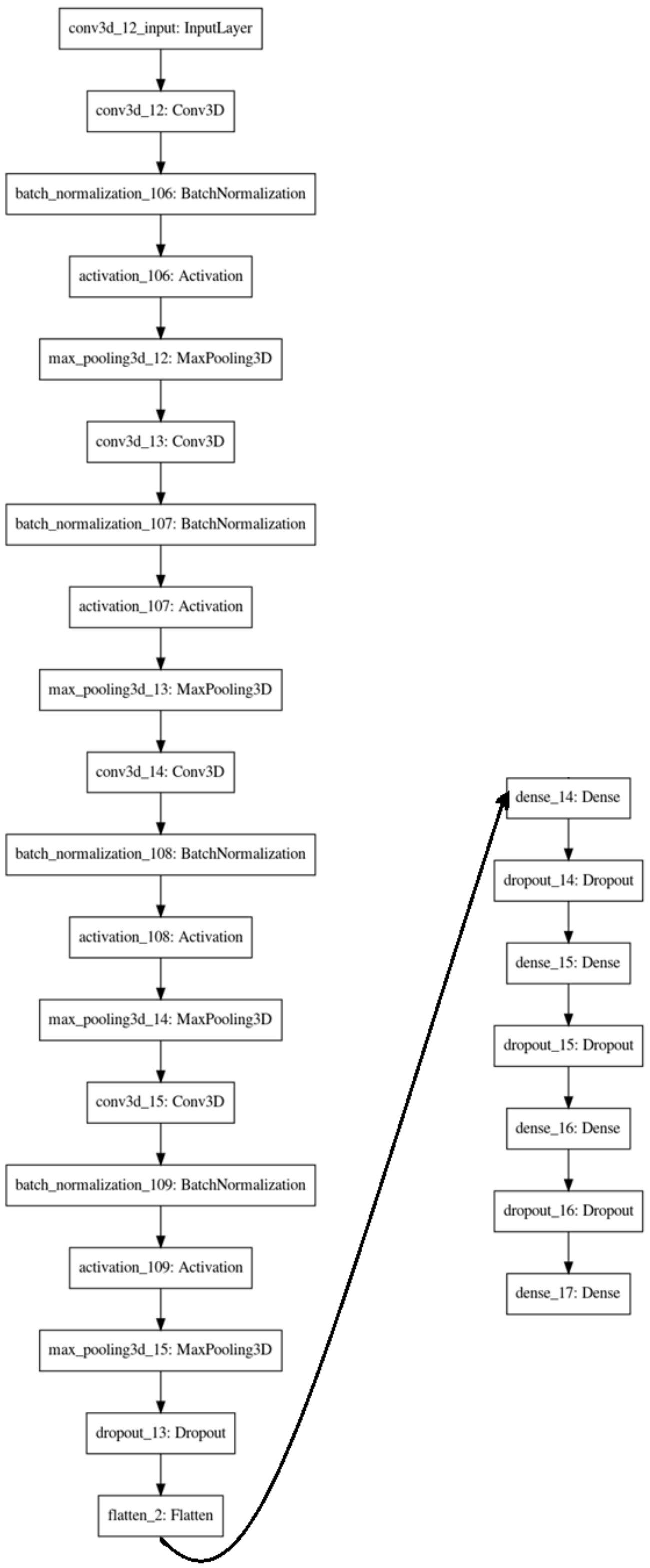
Approach 2: Conv3D

Model 6: Conv3D with Batch Normalization, Dropout

- Total params: 35,923,957
- Trainable params: 35,923,765
- Non-trainable params: 192
- **Best Epoch:** 16
- **Training: 0.5909 - categorical_accuracy: 0.7994**
- **Validation: val_loss: 1.1632, val_categorical_accuracy: 0.6600**


conv3d_16_input: InputLayer
conv3d_16: Conv3D
batch_normalization_302: BatchNormalization
activation_298: Activation
conv3d_17: Conv3D
batch_normalization_303: BatchNormalization
activation_299: Activation
max_pooling3d_16: MaxPooling3D
conv3d_18: Conv3D
batch_normalization_304: BatchNormalization
activation_300: Activation
conv3d_19: Conv3D
batch_normalization_305: BatchNormalization
activation_301: Activation
max_pooling3d_17: MaxPooling3D
dropout_25: Dropout
flatten_7: Flatten
dense_26: Dense
dropout_26: Dropout
dense_27: Dense
dropout_27: Dropout
dense_28: Dense

Model 7: Conv3D with higher Dropout and Dense layers (reduced params)


conv3d_12_input: InputLayer
conv3d_12: Conv3D
batch_normalization_106: BatchNormalization
activation_106: Activation
max_pooling3d_12: MaxPooling3D
conv3d_13: Conv3D
batch_normalization_107: BatchNormalization
activation_107: Activation
max_pooling3d_13: MaxPooling3D
conv3d_14: Conv3D
batch_normalization_108: BatchNormalization
activation_108: Activation
max_pooling3d_14: MaxPooling3D
conv3d_15: Conv3D
batch_normalization_109: BatchNormalization
activation_109: Activation
max_pooling3d_15: MaxPooling3D
dropout_13: Dropout
flatten_2: Flatten
dense_14: Dense
dropout_14: Dropout
dense_15: Dense
dropout_15: Dropout
dense_16: Dense
dropout_16: Dropout
dense_17: Dense

- Total params: 2,817,477
- Trainable params: 2,816,997
- Non-trainable params: 480
- **Best Epoch:** 25 of second run (55th Epoch)
- **Training: loss: 0.1759 - categorical_accuracy: 0.9502**
- **Validation: val_loss: 0.2894 - val_categorical_accuracy: 0.930**

Model 8: Conv3D (Model 7 with Nadam as optimizer)

- Total params: 2,817,477
- Trainable params: 2,816,997
- Non-trainable params: 480
- Best Epoch: 1st of second run (61th Epoch)
- **Training: loss: 0.0856, categorical_accuracy: 0.9744**
- **Validation: val_loss: 0.5302, val_categorical_accuracy: 0.9000**

Model 9: Conv3D (Model 7 with RMSprop as optimizer)

- Total params: 2,817,477
- Trainable params: 2,816,997
- Non-trainable params: 480
- Best Epoch: 45
- **Training: loss: 0.0491, categorical_accuracy: 0.9834**
- **Validation: val_loss: 0.4420, val_categorical_accuracy: 0.8800**

Conclusion:

We have performed experiments with different types of architectures. For Conv2D+ RNN types of models, we have explored different transfer learnings and fine-tuned the pre-trained weights to extract features and then fed it to RNN-based networks like LSTM, RNN. This type of architectures have lot of model parameters and it took good amount of time to train these kinds of models. We explored 5 such models and got best validation accuracy of .85 using Xception(Finetuned) + LSTM architecture (Model 5). The model has 40133165 parameters.

Then we tried exploring different network architectures of Conv3D. Training time required for these models are far lesser than Conv2D+ RNN type models. Explored Adam, Nadam, RMSprop as optimizers in different experiments. Adam and Nadam gave faster convergence and better validation accuracy. We got best validation accuracy of .93 (Model 7). This model has only 2817477 parameters (almost 15 times lesser than best Conv2D+RNN type model). Model 7 (model-00025-0.17594-0.95023-0.28937-0.93000.h5) considered as our final model.