Gesture Recognition using Neural Networks

By

Khan Imran Jabbar

Problem Statement

An electronics company which manufactures state of the art smart televisions wants to develop a 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.

This TV will continuously monitor the user via a webcam and will classify the input it receives, into 5 categories using a classification model. The TV will then perform the necessary action based on the classification result.

Scope and limitations

The creation of a neural network based video classification model that will help classify the action into the 5 pre-determined classes. All models will be limited to Neural Network models only.

Data Available

663 videos split into 30 frames, each video belonging to a 1 of the 5 class. Also available are 100 videos for validation split into 30 frames, each video belonging to 1 of the 5 classes. There are 2 CSV also provided that have relevant information on the data provided.

Model building thought process

We will choose a few models from this initial list and then refine them until we think we have acceptable results.

Pre-processing steps used/experimented with

1) Image size : Size of the image to be fed to the Neural Network. The dimensions experimented with were :
    a. 120 – Used as most frame available were already had 120 as the lowest dimension. This dimension was finally chosen and was used for all the models created
    b. 240 – Experimented with: but there was no apparent improvement over using 120, hence was discarded

2) Normalization: Techniques used to "Normalize" the image before feeding to the Neural Network. The techniques experimented with were:
    a. (image−np.min(image))/(np.max(image)−np.min(image))
        i. This normalization resulted in model performance degradation over option b and hence was discarded
    b. (image−np.percentile(image,5))/(np.percentile(image,95)−np.percentile(image,5))
        i. This normalization did not negatively affect the model performance and hence was chosen as the normalization method
    c. Image/255
        i. This normalization degraded the model performance, lowering the training accuracy by almost 20% in the test run and hence was discarded

Other Experiments

1) Optimizer
    a. SGD
        i. SGD was tested for the test run, but ended up giving "NAN" as loss which was due to exploding gradients. Skipped it to in favor of ADAM
    b. ADAM
        i. Adaptive optimizer worked without a hitch, so it was used as an optimizer in the model building

Batch size as a parameter to control hardware resource

a. As the code was being implemented across 4 different machine and 2 cloud platform, fixing the batch size was a problem due to different hardware of each physical machine. Error trapping was implemented to avoid stoppage and the batch size was reduced programmatically so the code can run with user interference

Please note: Batch size as model hyper parameter has been discussed later

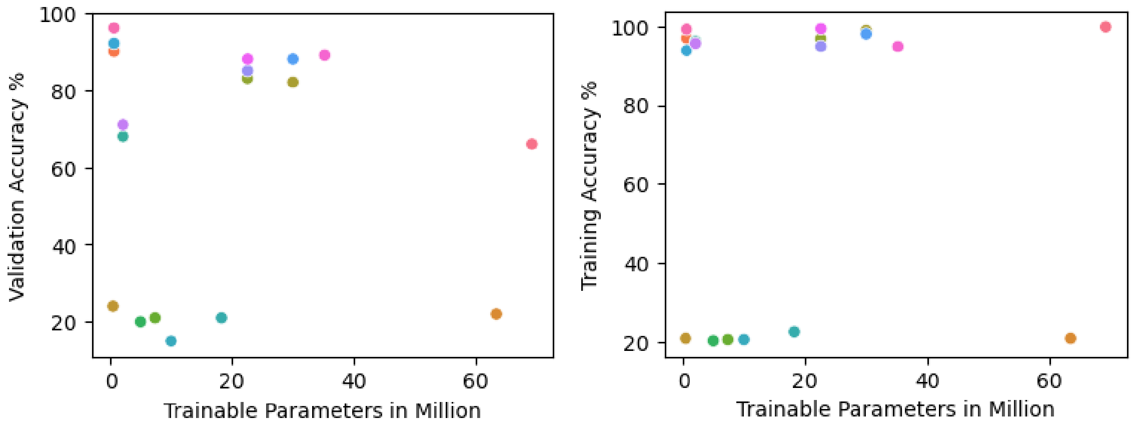
Modelling

19 models were built and tested. The number includes the all the models that were re-trained after tuning parameters or architecture.

Please note: All final layers were softmax layer with 5 neurons and all models ran for 20 epochs

A Quick look at what we observed

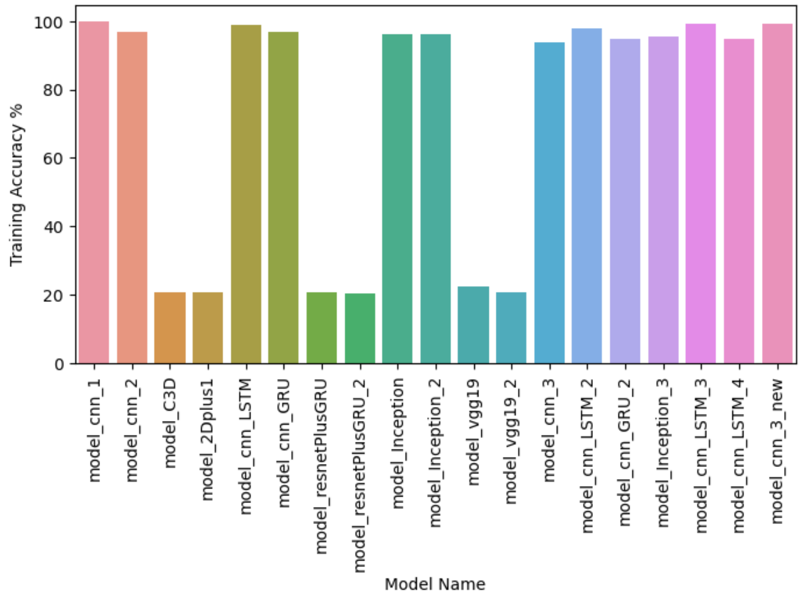
Trainable parameters by accuracy score



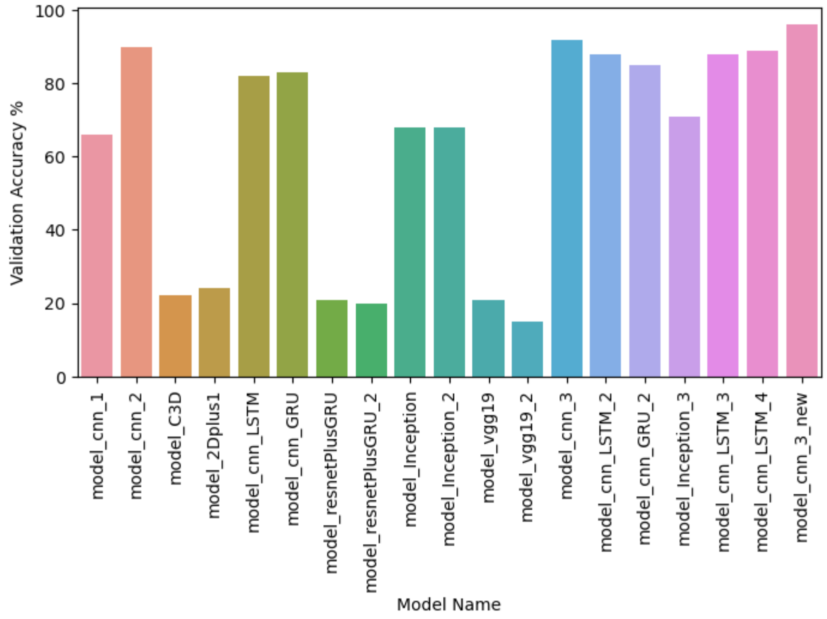
An increase in trainable parameters did not translate to higher accuracy score for the models tested given that they were trained for a limited number of epochs

Accuracy Score by Model

Training


Training Accuracy %
0
20
40
60
80
100
model_cnn_1
model_cnn_2
model_C3D
model_2Dplus1
model_cnn_LSTM
model_cnn_GRU
model_resnetPlusGRU
model_resnetPlusGRU_2
model_Inception
model_Inception_2
model_vgg19
model_vgg19_2
model_cnn_3
model_cnn_LSTM_2
model_cnn_GRU_2
model_Inception_3
model_cnn_LSTM_3
model_cnn_LSTM_4
model_cnn_3_new
Model Name

Validation


Validation Accuracy %
0
20
40
60
80
100
model_cnn_1
model_cnn_2
model_C3D
model_2Dplus1
model_cnn_LSTM
model_cnn_GRU
model_resnetPlusGRU
model_resnetPlusGRU_2
model_Inception
model_Inception_2
model_vgg19
model_vgg19_2
model_cnn_3
model_cnn_LSTM_2
model_cnn_GRU_2
model_Inception_3
model_cnn_LSTM_3
model_cnn_LSTM_4
model_cnn_3_new
Model Name

First Iteration

1) Non- Pretrained models

Model Name	Type	Notes
model_cnn_1	Conv3D	1st experiment. Low Performance
model_cnn_2	Conv3D	Maxpooling , batch normalization and dropout were used. High Performance
model_C3D	Conv3D	Very low performance Idea based on the paper "Learning Spatiotemporal Features with 3D Convolutional Networks" https://arxiv.org/pdf/1412.0767.pdf
model_2Dplus1	Conv3d with skip connection. Keras functional API's used to build custom model	Took the longest to build and run and gave one of the lowest performances among all the built models. Idea based on the paper "A Closer Look at Spatiotemporal Convolutions for Action Recognition" https://arxiv.org/pdf/1711.11248v3.pdf
model_cnn_LSTM	Time distributed Conv2D layers that were flatten and fed to a LSTM layer of 1024 neurons	Simple to build architecture and model had high performance
model_cnn_GRU	Time distributed Conv2D layers that were flatten and fed to a GRU layer of 1024 neurons	Simple to build architecture and model had high performance

Some more information on the above models

Model Name	Trainable Parameters (In Million)	Batch Size	Training Accuracy %	Validation Accuracy %
model_cnn_1	69.149685	4	99.85	66
model_cnn_2	0.598117	8	96.98	90
model_C3D	63.366197	4	20.81	22
model_2Dplus1	0.442645	8	20.81	24
model_cnn_LSTM	29.991557	8	98.94	82
model_cnn_GRU	22.522501	8	96.83	83

2) Using Pretrained models

Model Name	Type	Notes
model_resnetPlusGRU	Resnet50 with 10 trainable layers passed to a GRU layer with 512 neurons	Very low performance
model_resnetPlusGRU_2	Resnet50 with 5 trainable layers passed to a GRU layer with 512 neurons	Very low performance
model_Inception	InceptionV3 with 10 trainable layers with an added Globalaveragepooling	Dropped GRU as it didn't seem to work with first 2 pretrained model. Performance average
model_Inception_2	InceptionV3 with 5 trainable layers with an added Globalaveragepooling	There was no performance improvement seen from the previous inception model
model_vgg19	VGG19 with 10 trainable layers with an added Globalaveragepooling	Very Low Performance
model_vgg19_2	VGG19 with 5 trainable layers with an added Globalaveragepooling	Even lower performance than model_vgg19

Some more information on the above models

Model Name	**Trainable Parameters (In Million)**	**Batch Size**	**Training Accuracy %**	**Validation Accuracy %**
model_resnetPlusGRU	7.354373	8	20.51	21
model_resnetPlusGRU_2	4.992517	8	20.21	20
model_Inception	2.103493	8	96.23	68
model_Inception_2	2.103301	8	96.08	68
model_vgg19	18.229253	8	22.47	21
model_vgg19_2	9.969669	8	20.51	15

Among the pretrained model InceptionV3 was the only one that had somewhat acceptable results, but both inception models were overfitting.

Second Iteration

We chose the top 3 top performing non-pretrained model from the above list to fine tune it experimentally to check if we could achieve better the results

Tuning Batch size

During the test / experimental process we saw that batch size had a huge impact on the accuracy. For the top models chosen from the 1st iteration, we reduced the batch previously used to half, and ran the model for 20 epochs.The model architecture remained exactly as the original ones

Model Name	Derived from	Notes
model_cnn_3	model_cnn_2	Validation accuracy increased from 90% to 92% and the model overfit lesser than the previous version (Training accuracy dropped to 93.38% from 96.98%)
model_cnn_LSTM_2	model_cnn_LSTM	Validation accuracy increased from 82% to 88% but the training accuracy hovered around 98% for both indicating a slight overfit
model_cnn_GRU_2	model_cnn_GRU	Validation accuracy increased from 83% to 85% and the training accuracy dropped from 96.83% to 94.87% indicating the model was starting to generalize more than the previous iteration

Second iteration for pretrained model

Given that almost no pre-trained model had acceptable results, we chose only model_inception_2 to see if we could squeeze any performance. In the first iteration we saw that freezing more layers gave us a performance bump, so we did duplicate the architecture but froze all but the top layer and reduced batch size to half(model_inception_3). There was a slight improvement in the validation but the model was still overfitting. The decision to drop all the pre-trained from any further refinement process

Some more information about models in the second iteration

Model Name	Trainable Parameters (In Million)	Batch Size	Training Accuracy %	Validation Accuracy %
model_cnn_3	0.598117	4	93.82	92
model_cnn_LSTM_2	29.991557	4	98.04	88
model_cnn_GRU_2	22.552501	4	94.87	85
model_Inception_3	2.103301	4	95.63	71

Third iteration

In this we took the top 2 performing models from second iteration and experimented to see if we could get any performance increase

Model Name	Derived from/Changes	Notes
model_cnn_LSTM_3	model_cnn_LSTM_2 -Added 2 more Time distributed Conv2d layers before the LSTM and increased the LSTM from 1024 to 2048 Dropped batchsize to 2	Validation accuracy (88%) didn't increase but training accuracy increased from 98.04% to 99.4% indicating that further training without more data could lead to over fit
model_cnn_LSTM_4	model_cnn_LSTM_3 -Added another LSTM layer of 1024 after the LSTM 2048 layers from the previous model Increased batch size to 4	Validation accuracy increased from 88% to 89% and training accuracy decreased to 94.84 from 99.4%
model_cnn_3_new	Model_cnn_3 No changes to any parameters	This was our best performing model among all iteration and it also has very few trainable parameters compared to other. So instead of changing anything we decided to refine it by training the model_cnn_3 for 20 more epochs (40 total)

The results of the last iterations:

Model Name	Trainable Parameters (In Million)	Batch Size	Training Accuracy %	Validation Accuracy %
model_cnn_LSTM_3	22.564741	2	99.4	88
model_cnn_LSTM_4	35.146629	4	94.84	89
model_cnn_3_new	0.598117	4	99.25	96

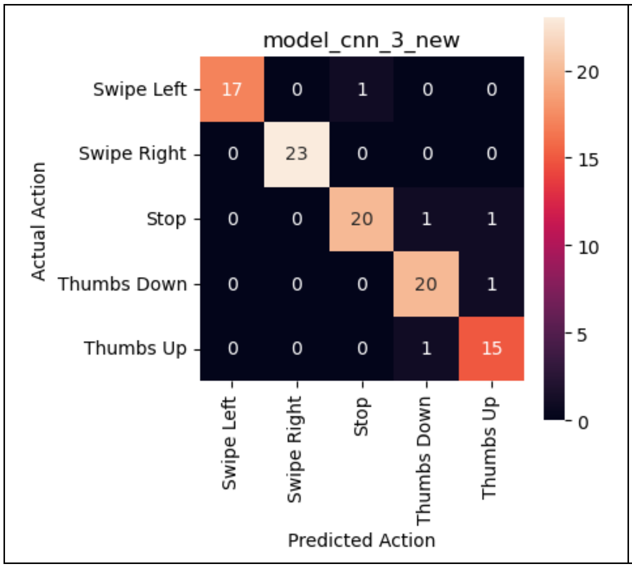
Final choice

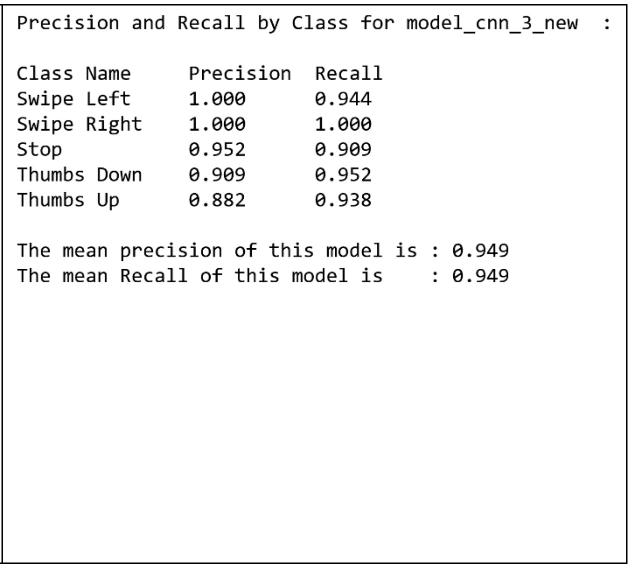
Model_cnn_3_new is the obvious winner as it has the highest accuracy with less very few trainable parameters and model file size of 6.93MB.

Final Evaluation

The top 3 models from all models created were 1) cnn_model_3_new 2) cnn_model_3 and 3) cnn_model_2. We tested these models against the validation data and these were the results

1) cnn_model_3_new



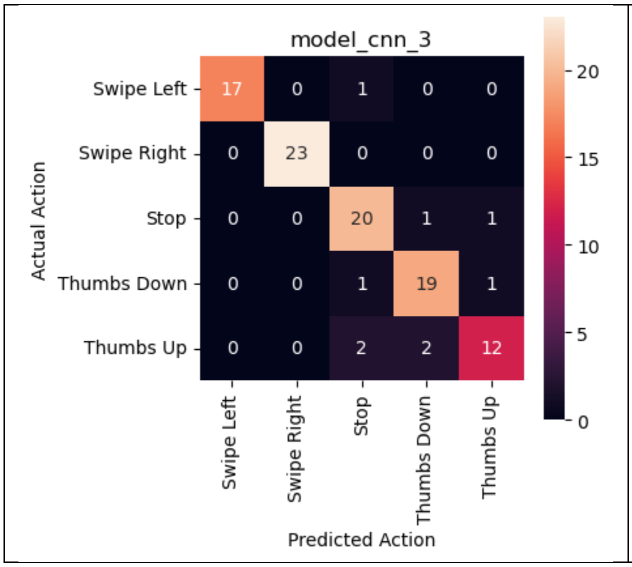


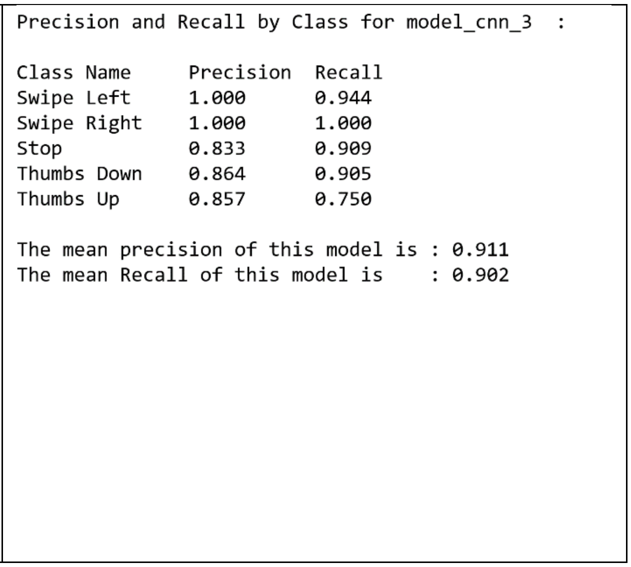
```
Precision and Recall by Class for model_cnn_3_new  :

Class Name     Precision  Recall
Swipe Left     1.000      0.944
Swipe Right    1.000      1.000
Stop           0.952      0.909
Thumbs Down    0.909      0.952
Thumbs Up      0.882      0.938

The mean precision of this model is : 0.949
The mean Recall of this model is    : 0.949
```

2) cnn_model_3



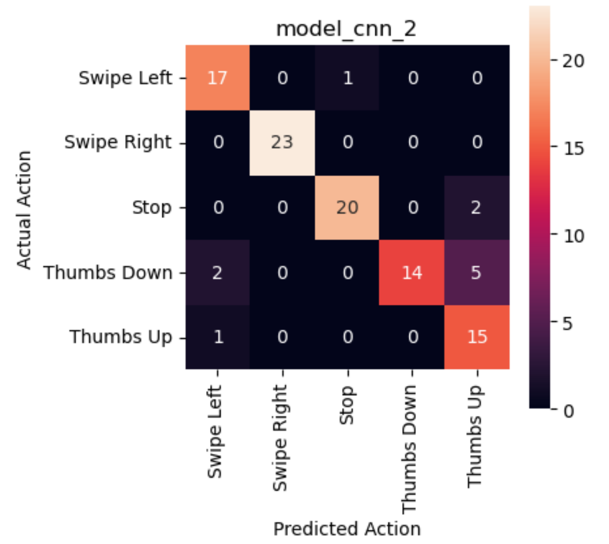


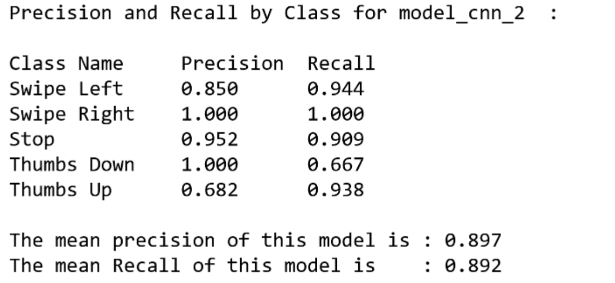
```
Precision and Recall by Class for model_cnn_3  :

Class Name     Precision  Recall
Swipe Left     1.000      0.944
Swipe Right    1.000      1.000
Stop           0.833      0.909
Thumbs Down    0.864      0.905
Thumbs Up      0.857      0.750

The mean precision of this model is : 0.911
The mean Recall of this model is    : 0.902
```

3) cnn_model_2





```
Precision and Recall by Class for model_cnn_2  :

Class Name      Precision  Recall
Swipe Left      0.850      0.944
Swipe Right     1.000      1.000
Stop            0.952      0.909
Thumbs Down     1.000      0.667
Thumbs Up       0.682      0.938

The mean precision of this model is : 0.897
The mean Recall of this model is    : 0.892
```