GESTURE RECOGNITION CASE STUDY

.h5 File Google Drive URL :-

https://drive.google.com/drive/folders/1blgp96YdALGD13k8FIvBSnL2UTFx4yRe?usp=sharing

Because of 50MB restriction by upgrad we cannot upload .h5 file in zip

We started with data generator function. Two types of data generator were used. One without augmentation and one with augmentation. The first generator was mostly used. And even our final model is built with generator without data augmentation.

Steps specific to data augmentation generator:

We have used cv2.warpAffine transformation (Image-1). Since the direction of gesture is important, we have used this transformation as all parallel lines in the original image will still be parallel in the output image. We have converted to gray image (image-2) and cropped the black part of image (image-3) whichever we got after transforming and performed above mentioned common steps.

MODEL DETAILS

Model 1: Image size=120*120, batch_size=40, num_epochs=15

Training Accuracy – 86.85 %

Validation Accuracy – 75.00 %

This is the first model. In upcoming models, we will experiment with different hyperparameters. We have used 'relu' as activation function for all layers except output layer (used softmax function here). Used Adam optimizer with learning rate 0.001

Model 02: Same as model-1 but we changed activation function to 'elu' instead of 'relu' and SGD optimizer instead of Adam.

Training Accuracy - 75.4 %

Validation Accuracy - 71.67 %

We can see that our validation accuracy is considerably higher than training accuracy in most of the cases.

This could be due to high dropouts since we are using 0.5

Very basic model with inadequate amount of data

For all the above reasons, it is better to tune more hyperparameters with the initial model (Model - 1)

Model 03: model-1 performs better. So, Tweaked model-1 by increasing num_epochs=25

Training Accuracy – 85.5 %

Validation Accuracy - 82.00 %

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv3d_20 (Conv3D)           (None, 30, 120, 120, 8)   656

batch_normalization_20 (Bat  (None, 30, 120, 120, 8)  32
chNormalization)

activation_20 (Activation)   (None, 30, 120, 120, 8)   0

conv3d_21 (Conv3D)           (None, 30, 120, 120, 16)  3472

activation_21 (Activation)   (None, 30, 120, 120, 16)  0

batch_normalization_21 (Bat  (None, 30, 120, 120, 16)  64
chNormalization)

max_pooling3d_16 (MaxPoolin  (None, 15, 60, 60, 16)   0
g3D)

conv3d_22 (Conv3D)           (None, 15, 60, 60, 32)    4128

activation_22 (Activation)   (None, 15, 60, 60, 32)    0

...
Trainable params: 6,866,869
Non-trainable params: 496
```

Increasing epoch have increased accuracy.

Even though computational time/training time was slightly high.

The above are best values we got in 3rd Model. Going with epoch-25 values as the difference between training and validation accuracy is <5%. The computation time increases with the number of epochs; however, the accuracy also increases and gradually the model runs better.

Model 4: Model-3 is best among above. Tweaking it by increasing image size=160*160

Model 5 : Model-3 is best among above. Tweaking it by increasing image size=140*140

Model 6 : Model-3 is best among above. Tweaking it by increasing batch size=50

For Model-4,5,6, When we increase the memory, the tensor size increases, and the GPU outputs results in OOM exception, the resources are not enough to run the tensor.

Model 7 : Model-3 is best among above. Tweaking it by decreasing batch size=15

Training Accuracy – 43.14 %

Validation Accuracy – 36.67 %

By decreasing the number of video sequences in each batch, i.e batch size the model is not able to learn the data, and the overall accuracy is very poor to be selected as the base model.

The difference between training accuracy and validation accuracy is also very poor. The model is not able to generalize well which indicates model have not learnt enough. The model also has high bias. So, we cannot use it to predict unseen data. Thus, rejecting this model as base model.

Model 8 : CNN + LSTM - 120X120, Batch size 40,num_epochs=25

Training Accuracy – 78.89 %

Validation Accuracy – 68.33 %

The above training accuracy is decent but fails in case of validation accuracy, this could be because the LSTM model used is very simple and introduction of more dense layers can help.

Model 9 : With GRU model

Training Accuracy – 93.08 %

Validation Accuracy – 70.00 %

The difference between Training and validation accuracy is very huge (around 20%) and this indicates Overfitting i.e the model does not fit well for unseen data.

Model 10: Hyper parameter tuned on Base Model (Model 3) - Data Augmentation

Training Accuracy – 82.01 %

Validation Accuracy – 73.33 %

The training accuracy increases gradually throughout the model but fluctuates highly for validation accuracy, and this indicates it is not a stable model and would not perform well on unseen data.

Data augmentation increases computation time and is not a suitable model.

Hence, we continue with Model 3 as our Base Model.

Model 11: Hyperparameter Tuned : Model Architecture - Added Dropouts

Training Accuracy – 20.76 %.

Validation Accuracy – 20.67 %

The dropout layers increase the number of parameters that are assigned as 0 and this underfits the model. As a result the model does not learn properly, and it leads to a decrease in accuracy.

Hence, we continue with Model 3 as our Base Model

Model 12: Hyperparameter Tuned : Model Architecture - Added more dense layers

Training Accuracy – 54.33 %.

Validation Accuracy – 45.00 %

The model is underfitting as well as complex. The foremost objective of training machine learning based model is to keep a good trade-off between simplicity of the model and the performance accuracy which cannot achieved with this model.

Model 13: To reduce memory footprint of Model 3

Training Accuracy – 93.77 %.

Validation Accuracy – 81.67 %

The model is overfitting.

Model ends up learning a greater number of parameters than required to solve your problem.

The model is not able to generalize the trend and will not able to predict fluctuations in the test data if any, so it is better to not use this for /unseen data.

we continue with Model 3 as our Base Model

Model 14: Hyperparameter Tuned : Filter size and Dense neurons (128)

Training Accuracy – 80.00 %.

Validation Accuracy – 75.00 %

With a smaller filter size, i.e. (2*2*2) the computational cost and weight sharing decreases greatly thus leading to lesser weights during back propagation.

2x2 and 4x4 are generally not preferred because odd-sized filters symmetrically divide the previous layer pixels around the output pixel. And if this symmetry is not present, there will be distortions across the layers.

3x3 is the optimal choice to be followed. This add one more reason for choosing model-3.

Best Model to be used in case there are memory constraints as it can achieve the same accuracy at a lower number of parameters. We are submitting this h5 file in zip folder. But performance wise, model-3 is best. Since, we are more emphasizing on performance now and not computation time, we will be using model-3 as our final model.

Refer below model-3 summary. The metric values used to take a decision is as follows:

loss='categorical_crossentropy', metrics=['categorical_accuracy']

We experimented by performing lot of hyperparameter tuning and finally decided model-3 as best model in terms of performance, optimal number of parameters, computational time, etc. We run same model more than twice to ensure that we are getting good accuracy every time.