GESTURE RECOGNITION CASE STUDY

Write up to choose the final model.

NOTE: LINK FOR BEST MODEL-3 h5 FILE:

https://drive.google.com/file/d/1zHmca0Mu69Cjj-_vOe6BmDRAI-a-5XJ1/view?usp=sharing

(Since, platform is not letting us to upload files size greater than 50MB, we have uploaded in our drive and shared the link as per TA's suggestion in discussion forum. However, for submission purpose we have also uploaded our memory footprint model-14 h5 file along with writeup and python notebook. But, please consider the file provided in link as our final best h5 file)

Discussion forum link : https://learn.upgrad.com/v/course/593/question/345116

We started with data generator function. We have two types of data generator. One without augmentation and one with augmentation. We mostly used first generator and even our final model is built with generator without data augmentation.

Common steps performed in both types of generators:

- ✓ Resized image using "imresize(image,(y,z))" function. We have used default resampling filter. *PIL.Image.NEAREST*. When tried with different sampling filters, we were more satisfied with the default parameter.
- ✓ Normalizing image by dividing by 255.Since all our images are natural images, we chose this method.

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.

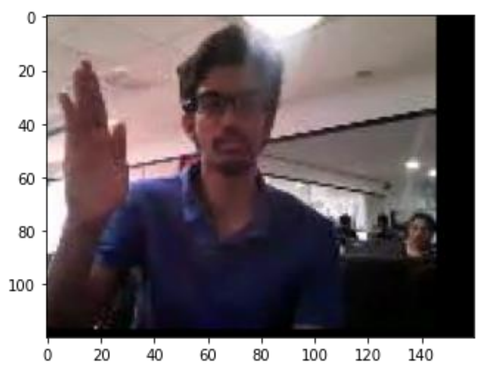

Image-1

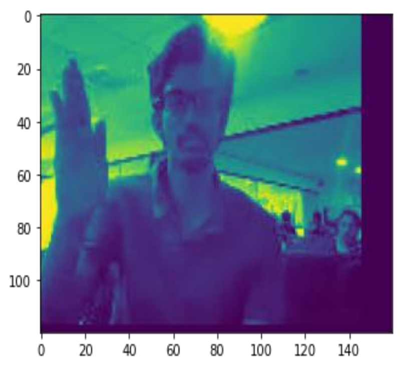

Image-2

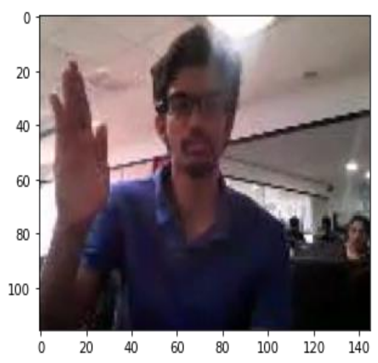

Image-3

Model 01: Image size=120*120, batch_size=40, num_epochs=15

- Best Training Accuracy – 86.85 % Best Validation Accuracy – 75.00 %

This is the most initial model, it is difficult to conclude that this will be our final model without hyper parameter tuning, so 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.

- Best Training Accuracy - 75.4 % Best Validation Accuracy - 71.67 %

From our model 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
- Indicates high bias in the neural network

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

Model 03: Out of above 2 models, model-1 performs better. So, Tweaked model-1 by changing num_epochs=25

- ***Best Training Accuracy – 85.5 % Best Validation Accuracy - 82.00 %***

We can clearly see that increasing epoch have increased accuracy even though computational time/training time was slightly high.

The above are best values we got in model 3. Going with epoch-25's 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.

Currently, we have obtained our best h5 model file.

Model 04: Model-3 is best among above. Tweaking it by increasing image size=160*160

OOM Error - ResourceExhaustedError: OOM when allocating tensor with shape [40,16,30,160,160] and type float on /job:localhost/replica:0/task:0/device: GPU:0 by allocator GPU_0_bfc

- Best Training Accuracy - None %   Best Validation Accuracy - None %

Model 05 : Model-3 is best among above. Tweaking it by increasing image size=140*140

OOM Error - ResourceExhaustedError: OOM when allocating tensor with shape [40,16,30,140,140] and type float on /job:localhost/replica:0/task:0/device: GPU:0 by allocator GPU_0_bfc

- Best Training Accuracy - None %   Best Validation Accuracy - None %

Model 06 : Model-3 is best among above. Tweaking it by increasing batch size=50

OOM Error - ResourceExhaustedError: OOM when allocating tensor with shape[50,16,30,120,120] and type float on /job:localhost/replica:0/task:0/device:GPU:0 by allocator GPU_0_bfc

- Best Training Accuracy - None %   Best Validation Accuracy - None %

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

- Best Training Accuracy – 43.14 % Best 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 08 : CNN + LSTM - 120X120, Batch size 40,num_epochs=25

- Best Training Accuracy – 78.89 % Best 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 09 : With GRU model

- Best Training Accuracy – 93.08 %   Best 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

- Best Training Accuracy – 82.01 % Best 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

- Best Training Accuracy – 20.76 %. Best Validation Accuracy – 20.67 %

The dropout layers increase the number of parameters that are assigned as 0 (dropping out 25% neurons in most of layers) and this underfits the model, as a result the model does not learn properly, and it leads to a very poor accuracy.

Hence, we continue with Model 3 as our Base Model

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

- Best Training Accuracy – 54.33 %. Best 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 is not achieved with this model.

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

Model 13: To reduce memory footprint of Model 3

- Best Training Accuracy – 93.77 %. Best Validation Accuracy – 81.67 %

The model is overfitting as there is a huge difference between the training and validation accuracy. Model ends up learning a greater number of parameters than are needed 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.

Hence, we continue with Model 3 as our Base Model

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

- Best Training Accuracy – 80.00 %. Best Validation Accuracy – 75.00 %

With a smaller filter size, i.e. (2*2*2) the computational cost and weight sharing decreases greatly. This leads 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. As, mentioned above in the note, we are submitting this h5 file in zip folder. But performance wise, model-3 is best still. Since, we are more emphasizing on performance now and not computation time, let's still use 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.

```
Layer (type)                 Output Shape              Param #   
=================================================================
conv3d_11 (Conv3D)           (None, 30, 120, 120, 8)   656       
_________________________________________________________________
batch_normalization_11 (Batc (None, 30, 120, 120, 8)   32        
_________________________________________________________________
activation_11 (Activation)   (None, 30, 120, 120, 8)   0         
_________________________________________________________________
conv3d_12 (Conv3D)           (None, 30, 120, 120, 16)  3472      
_________________________________________________________________
activation_12 (Activation)   (None, 30, 120, 120, 16)  0         
_________________________________________________________________
batch_normalization_12 (Batc (None, 30, 120, 120, 16)  64        
_________________________________________________________________
max_pooling3d_9 (MaxPooling3 (None, 15, 60, 60, 16)    0         
_________________________________________________________________
conv3d_13 (Conv3D)           (None, 15, 60, 60, 32)    4128      
_________________________________________________________________
activation_13 (Activation)   (None, 15, 60, 60, 32)    0         
_________________________________________________________________
batch_normalization_13 (Batc (None, 15, 60, 60, 32)    128       
_________________________________________________________________
max_pooling3d_10 (MaxPooling (None, 7, 30, 30, 32)     0         
_________________________________________________________________
conv3d_14 (Conv3D)           (None, 7, 30, 30, 64)     16448     
_________________________________________________________________
activation_14 (Activation)   (None, 7, 30, 30, 64)     0         
_________________________________________________________________
batch_normalization_14 (Batc (None, 7, 30, 30, 64)     256       
_________________________________________________________________
max_pooling3d_11 (MaxPooling (None, 3, 15, 15, 64)     0         
_________________________________________________________________
conv3d_15 (Conv3D)           (None, 3, 15, 15, 128)    65664     
_________________________________________________________________
activation_15 (Activation)   (None, 3, 15, 15, 128)    0         
_________________________________________________________________
batch_normalization_15 (Batc (None, 3, 15, 15, 128)    512       
_________________________________________________________________
max_pooling3d_12 (MaxPooling (None, 1, 7, 7, 128)      0         
_________________________________________________________________
flatten_3 (Flatten)          (None, 6272)              0         
_________________________________________________________________
dense_7 (Dense)              (None, 1000)              6273000   
_________________________________________________________________
dropout_5 (Dropout)          (None, 1000)              0         
_________________________________________________________________
dense_8 (Dense)              (None, 500)               500500    
_________________________________________________________________
dropout_6 (Dropout)          (None, 500)               0         
_________________________________________________________________
dense_9 (Dense)              (None, 5)                 2505      
=================================================================
Total params: 6,867,365
Trainable params: 6,866,869
Non-trainable params: 496
_________________________________________________________________
```