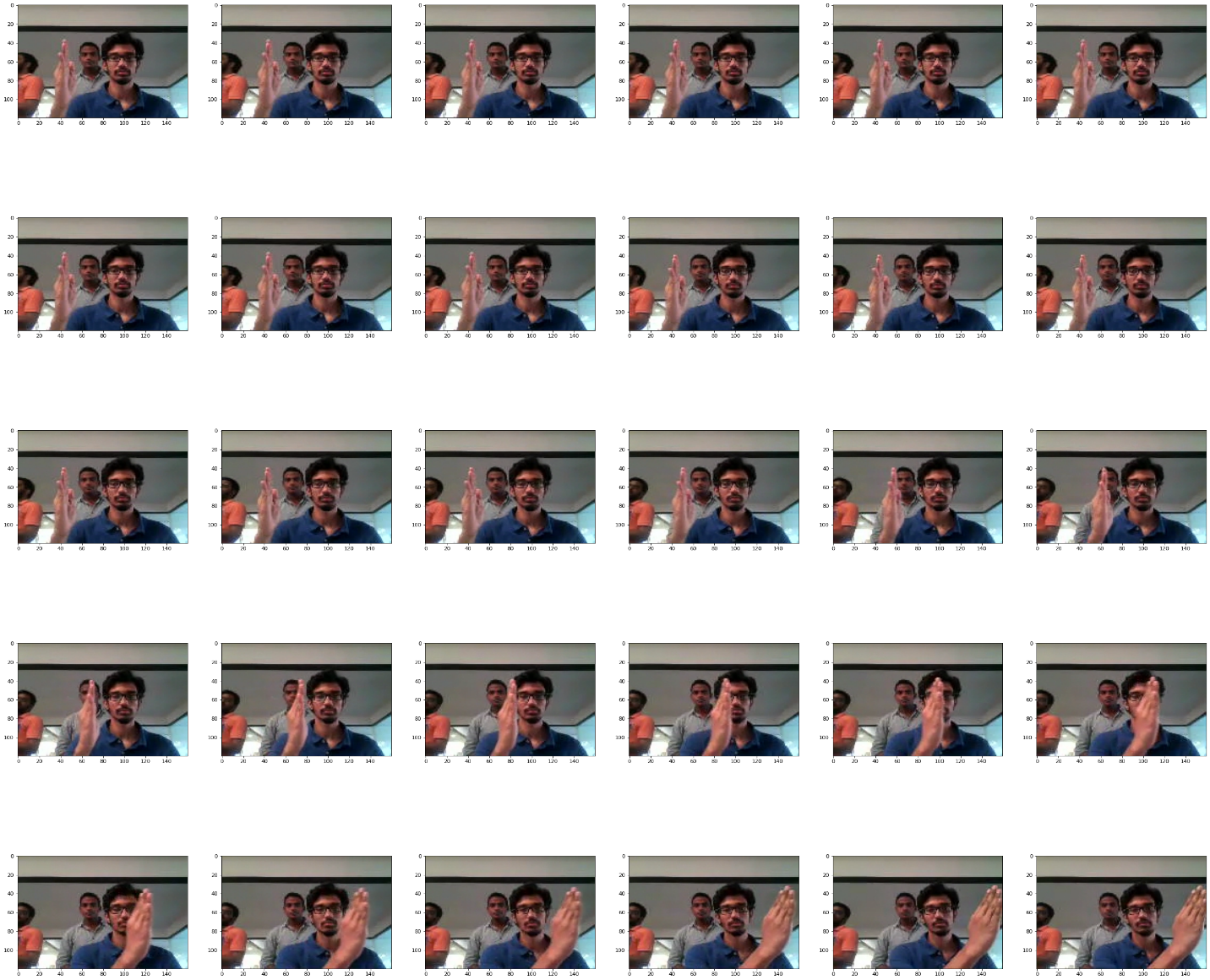
Problem Statement

*As a data scientist at a home electronics company that 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.*

|  |  |
| --- | --- |
| Thumbs up ‎‎ | Increase the volume |
| Thumbs down | Decrease the volume |
| Left swipe | Backward 10 seconds |
| Right swipe | Forward 10 seconds |
| Stop | Pause |

Preview



Dataset Reading

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 - similar to what the smart TV will use.

Objective

* *To train different models on the 'train' folder to predict the action performed in each sequence or video and which performs well on the 'Val' folder as well.*
* *The final model should have a smaller number of parameters (lightweight) possible so that it can be implemented on the webcam mounted on top of the TV.*

Generator Function

*This is one of the most important parts of the code. In the generator, we are going to pre-process the images as we have images of 2 different dimensions (360 x 360 & 120 x 160) as well as create a batch of video frames. The generator should be able to make a batch of videos as input without any error. Steps like cropping, resizing and normalization should be performed successfully.*

Data Pre-processing:

* *Resizing: As Input images are of two different dimensions, we have to resize the images to a common shape (image height x image width). The output image dimension is decided by the argument passed to the generator function. The output image dimension can be adjusted by the hardware available to train the model further there is a trade-off between the output image dimension and the batch size.*

* *Normalization: Normalization is a technique by which we can scale the values without losing out on the information. It plays a vital role in speeding up the training process. We read the images via OpenCV we convert the image from BGR to RGB, we resize it then finally we will normalize each channel separately.*
* *Batch creation: This is the curtail part of the generator function. It creates a batch of images and labels and returns them to the caller function unlike other functions generator function works differently by the technique called yielding it does not process every batch in a single go instead it yields only one batch of data when the generator function is called once.*

Model Setup

*Before experimenting with different models, it’s good to develop all the required functions needed for the model-building process so that we can avoid writing codes again and again for each model we build. So, in our case, we have created 3 inner functions and 1 outer function.*

* *model\_params: While building the model we would call only the outer function called ‘model\_params’ and that outer function would call all the inner functions generator\_function, callback, and steps\_per\_epoc. This outer function also returns all the data acquired from the inner functions such as train\_generator, val\_generator, callbacks\_list, steps\_per\_epoch, and validation\_steps to the caller function.*
* *generator\_function: This generator function will further call our main generator function for the train and validation batch and get the generator object and store it in the train\_generator and val\_generator variable and returns it to the main function.*
* *callback\_function: The callback function creates a call-back list and returns to the model\_params. The returned list will have two keras inbuild callback functions ModelCheckpoint and ReduceLROnPlateau.*
* *steps\_per\_epocs\_function: This is the function to calculate how many steps the model should run to complete one epoch. This is also known as the number of batches in one epoch. This function returns two values i.e.) steps per epochs for training and the same for validation.*

Model build

*Here we make the model using different functionalities that Keras provides. We will experiment with different models with Conv3D and MaxPooling3D. We also want to experiment with* TimeDistributed while building a Conv2D + RNN model and transfer learning. The last layer will be SoftMax. We will design the network in such a way that the model can give good accuracy on the least number of parameters so that it can fit in the memory of the webcam

Base Model:

*We have successfully built our base model to test the model setups and generator function. We ran only for two epochs and the model is overfitted as expected. The Base model we have built is with Con3D architecture.*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Image Resolution* | *Batch size* | *Number of Epochs* | *Conv3D layers* | *MaxPool3D layer* | *Dense* | *Total Parameters* | *Training accuracy* | *Validation accuracy* | *Result* |
| 64 x 64 | 8 | 2 | 2 | 2 | 2 | 307,333 | 0.68 | 0.14 | Overfit |

* *Batch Normalization is used followed by the Conv3D layers and Dense layers except for the Dense with SoftMax activation*
* *Activation function is ‘relu’*

Model 1:

*As our base model is getting successfully overfitted as expected we experiment with the model parameter to come up with a wiser one. We experimented with increasing the Image Resolution, batch size, and several other parameters.*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Image Resolution* | *Batch size* | *Number of Epochs* | *Conv3D layers* | *MaxPool3D layer* | *Dense* | *Total Parameters* | *Training accuracy* | *Validation accuracy* | *Result* |
| 128 x 128 | 64 | 2 | 3 | 3 | 2 | 6,315,077 | - | - | Error |

* *Batch Normalization is used followed by the Conv3D layers and Dense layers except for the Dense with SoftMax activation*
* *Activation function is ‘relu’*

*We got 'ResourceExhaustedError' as the hardware we have is not efficient enough to handle the batch size of 64 with an image resolution of 128 x 128*

Model 2:

*After many experiments and the trade-off between the image resolution and the batch size, we got to the conclusion to choose the image Resolution as 64 x 64 and the batch size as 8.*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Image Resolution* | *Batch size* | *Number of Epochs* | *Conv3D layers* | *MaxPool3D layer* | *Dense* | *Total Parameters* | *Training accuracy* | *Validation accuracy* | *Result* |
| 64 x 64 | 8 | 2 | 3 | 3 | 2 | 1,596,485 | 0.73 | 0.17 | Overfit |

* *Batch Normalization is used followed by the Conv3D layers and Dense layers except for the Dense with SoftMax activation*
* *Activation function is ‘relu’*

*So, by choosing Image Resolution and Batch size as 64 x 64 & 8 we can build the model, but the model is overfitted as we ran only for 2 epochs.*

Model 3 – Final Model:

*We also tweaked the architecture of our previous model a little bit by adding one more Conv3D layer, MaxPooling3D layer, and dense layer to get better accuracy. As increased in MaxPooling3D layer also results in a lesser number of training parameters which is very crucial in our case. Also, we train the model for 25 epochs.*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Image Resolution* | *Batch size* | *Number of Epochs* | *Conv3D layers* | *MaxPool3D layer* | *Dense* | *Total Parameters* | *Training accuracy* | *Validation accuracy* | *Result* |
| 64 x 64 | 8 | 25 | 4 | 4 | 3 | 360,133 | 0.90 | 0.88 |  |

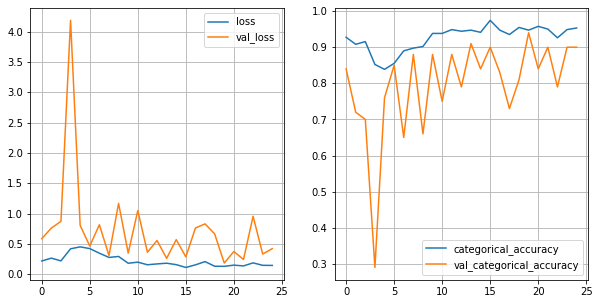
* *Batch Normalization is used followed by the Conv3D layers and Dense layers except for the Dense with SoftMax activation*
* *Activation function is ‘relu’*
* *Dropout with 50% is used after the 1st Dense layer.*
* *Dropout with 25% is used after the 2nd Dense layer.*

As a result, the above model has achieved 88% validation accuracy and 90% training accuracy for 25 epochs. We trained the model for further 25 epochs to see if the performance increases.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Image Resolution* | *Batch size* | *Number of Epochs* | *Conv3D layers* | *MaxPool3D layer* | *Dense* | *Total Parameters* | *Training accuracy* | *Validation accuracy* | *Result* |
| 64 x 64 | 8 | 25+25 | 4 | 4 | 3 | 360,133 | 0.95 | 0.94 |  |

As expected, we got a very good result after 50 epochs. Though the model is built for 50 epochs the best result has been achieved at the 45th epoch.

* Training accuracy increased to 0.947
* Validation accuracy increased to 0.940
* Model also has lesser parameters of around 360K

**

*The model is our final model with higher accuracy in the validation dataset however we did several other experiments as well and those are as follows.*

* CNN2D-GRU
* *Transfer learning – Inception V3 combined with GRU*

CNN2D-GRU:

*We build GRU Model with Time distributed Conv2D and the other layers below are the result.*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Image Resolution* | *Batch size* | *Number of Epochs* | *TD Conv2D layers* | *TD MaxPool2D layer* | *GRU* | *Dense* | *Total Parameters* | *Training accuracy* | *Validation accuracy* | *Result* |
| 64 x 64 | 8 | 25 | 4 | 4 | 1(128) | 2 | 951,909 | 0.97 | 0.86 | overfit |

*The model is overfitting!*

Transfer learning InceptionV3-GRU:

*We build GRU Model in combination with the InceptionV3 (Transfer learning). Inception model followed by the batch norm, MaxPoll2D, flatten and dropout with 50%. We also added dropout with 50% after GRU (32) and dense (64) as well.*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Image Resolution* | *Batch size* | *Number of Epochs* | *TD Conv2D layers* | *TD MaxPool2D layer* | *GRU* | *Dense* | *Total Parameters* | *Training accuracy* | *Validation accuracy* | *Result* |
| 128 x 128 | 12 | 25 | 0 | 1 | 1(32) | 2 | 22,013,285 | 0.82 | 0.67 | overfit |

*The model is still overfitting! Maybe we need to add more layers or try with a little tweak of the architecture however since we are doing transfer learning here, the Model might end up getting more parameters which we don't want as need to implement this model in webcam we need to have less number of parameters in our model but we already have 22,013,285 parameters in this inception\_gru model.*

Conclusion

*As per the business requirement, we need to build the model with higher accuracy and a lower number of parameters possible to detect the 5 gestures correctly so that we can implement the same in the webcam mounted on top of the TV to take control of the frequently usable action using simple gestures.*

***Model-3*** *which is based on Conv3D architecture has a lesser number of parameters and it has both train and test accuracy as 94% can satisfy the business requirements.*