Deep Learning Course Project - Gesture Recognition

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:

1. Thumbs up: Increase the volume
2. Thumbs down: Decrease the volume
3. Left swipe: 'Jump' backwards 10 seconds
4. Right swipe: 'Jump' forward 10 seconds
5. Stop: Pause the movie

Dataset:

The training data consists of 663 folders each having 30 frames(sequences) of a video performing one of the above-mentioned actions.

The validation data has 100 folders each having 30 frames(sequences) of a video performing one of the above-mentioned actions.

Two csv files train.csv and val.csv are given. Each row of the CSV file represents one video and contains three main pieces of information - the name of the subfolder containing the 30 images of the video, the name of the gesture and the numeric label (between 0-4) of the video.

Case Study:

Build a Deep Learning model using Neural Networks and train on the ‘train’ folder and validate the same on ‘val’ folder.

Try below approaches:

1. Convolution 3D Model.
2. CNN + RNN Model.

Steps/Approach on the Case Study:

The model building went through below steps for the above assignment:

1. Build the Generator Function for normal as well as augmented data.
2. Iteratively build Conv 3D and CNN+RNN models by updating the hyper parameters based on accuracy results.
3. Select one model file(*.h5) which has best accuracy of all.

Generator Function:

- The template for data generator function is in place with the starter code.
- Initially we have taken the image size as 160x160.
- As mentioned by professor in the video, we have performed mean normalization for all the images here.
- The given code will process through all the train data only of remainder of num_of_videos = num_batches*batch_size. Else if num_videos - num_batches*batch_size > 0, that many videos will be left over unprocessed.
- We have filled in the existing generator code and also added an additional check after the last batch processed for all if the left-over videos if any.
- For the augmented_generator function we used the same code with an extra parameter augment=False. If the data is to be augmented, this parameter needs to be set to true while calling the function.

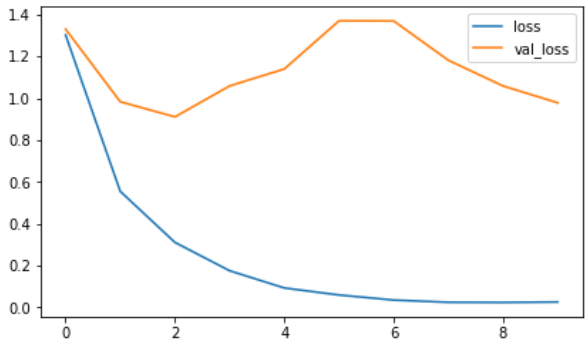
The initial batch size is 40.

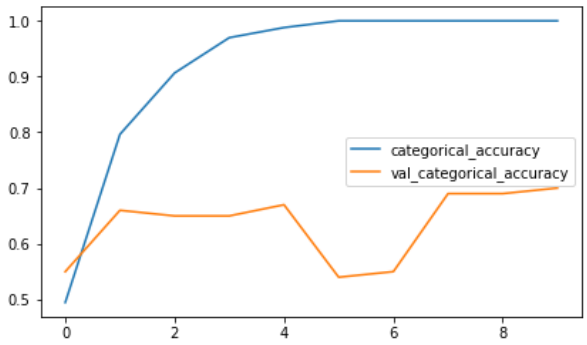
For all the models that are developed:

- Activation function for all hidden layers is relu.
- Activation function for output layer is softmax.
- Optimizer = Adam.
- Loss = Categorical Cross Entropy.
- Metrics = categorical_accuracy.
- We have Batch Normalization and Maxpooling3D((2,2)) for all the hidden Conv3D layers and some dense layers.
- We used ReduceLROnPlateau with factor =0.2 and patience = 0.4.

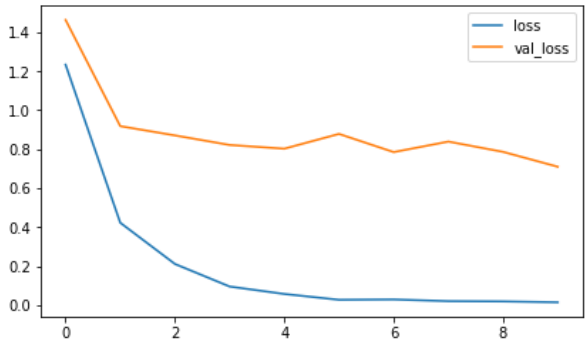
Convolutional 3D Model:

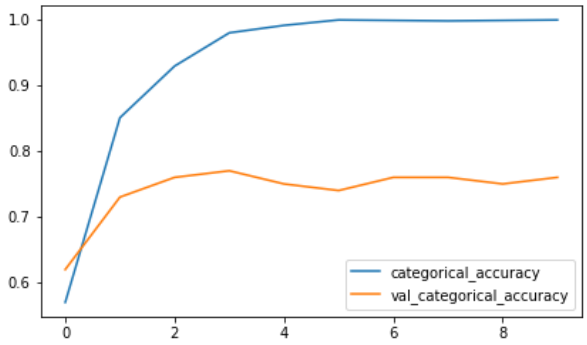
- We started off with building **first convolutional 3D model** with:
    - 4 Conv3D hidden layers with 8,16,32,64 neurons respectively and 1 Dense hidden layer.
    - Each layer has filter size of (2,2,2).
    - dropout of 0.2 for the final hidden layer.
    - Neurons in dense layer = 64.
- In the first run we saw the error for MaxPooling3D where the pooling window size and image size were not compatible. Then we updated the padding to ‘same’ such that the image will the padded for the pooling window to process it.
- Initially we tested the model with 3 epochs, and we saw the accuracy increasing for both train and validation data.
- Hence, we changed the epochs to 10 and ran the model.
- The first model results are as below:



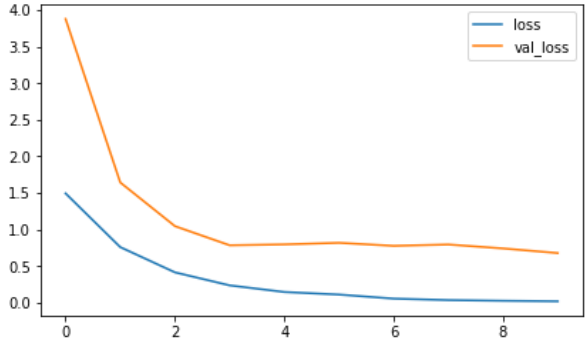


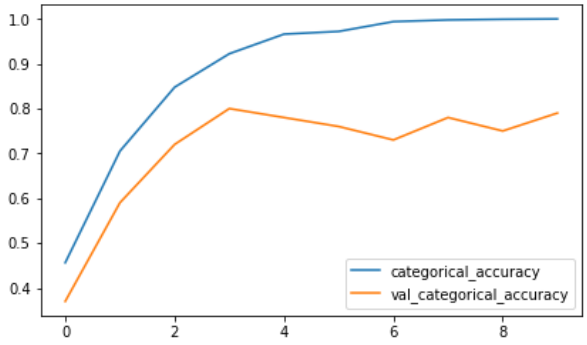
- As we see above, the model is over fit on training data with 100% accuracy from 5th Epoch. There is a huge difference between training and validation accuracy.
- We thought that this might be due to the data and hence we went for augmenting the data.
- After data augmentation this time we built a **second Conv3D model** with:
    - 5 Conv 3D hidden layers with 8,16,32,64,128 neurons respectively and 1 Dense layer with 128 neurons.
- We ran the model on augmented data and the results are as below:





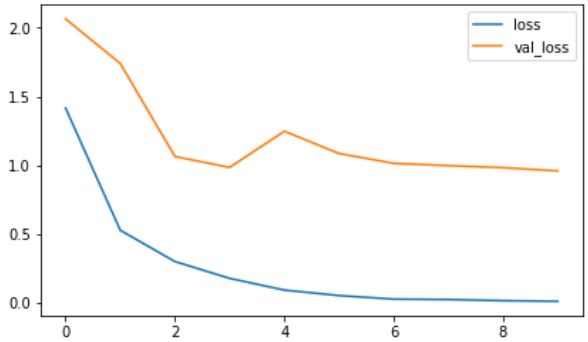
- Here, again the model started overfitting in Train Data from epoch 5. The validation accuracy lied around 70%-76% from 3rd Epoch.
- The best validation accuracy we got here is 76%.
- Since we have a small filter with (2,2,2), the model might capture more data from the image. Hence, we increased the filter size to (3,3,3).
- We re ran the model with filter size(3,3,3) and the results as below: - Can be treated as **Model 3**

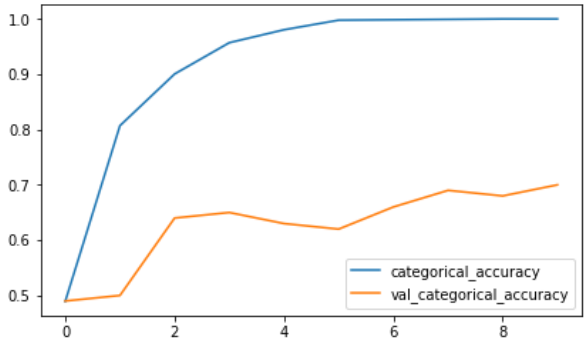




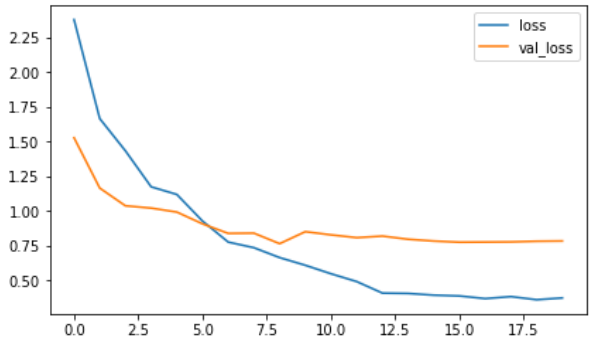
- We can see that the over fitting of train data has reduce a bit here.
- The validation accuracy also went up to 79% maximum.
- But still there is some over fitting of training data happening here.

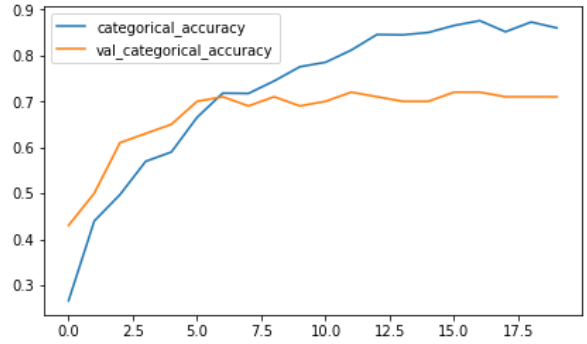
- We assumed that due to 5 conv3d layers there is lot of learning happening and decided to remove the last one.
- Hence, we choose to build **4th Conv3D model** without the last hidden Conv3D layer of 128 Neurons and filter size (3,3,3).
- Results:





- Here the model performance is degraded. Max Val Accuracy = 70% and Overfitting on Training data.
- There could be multiple reasons for overfitting like:
  - Dropout ratio is very less as 0.2. So, the model is not forgetting enough to be simple.
  - Filter size is unique across the layers, so the model may not learn the data missing in previous layer in subsequent layers.
  - Learning rate is 0.001, this could be high
- We have come up with **5th Conv 3D model** with:
  - 5 hidden Conv 3D layers with neurons 8,16,32,64,128.
  - 2 Dense hidden layers with 128 neurons
  - Filter size of (3,3,3) for first 4 layers and (2,2,2) for 5th hidden layer.
  - Dropout of 0.2 for 4th layer and 0.5 for last Dense hidden layer
  - Output layer: Dense with softmax activation for 5 classes.
  - 20 Epochs
- We ran the model and the results as below:





- We were able to achieve a reasonably good model where
  - There is no overfitting of train and val data.
  - Best Model's Accuracy on Train Data = 87.44%.
  - Best Model's Accuracy on Test Data = 72%.

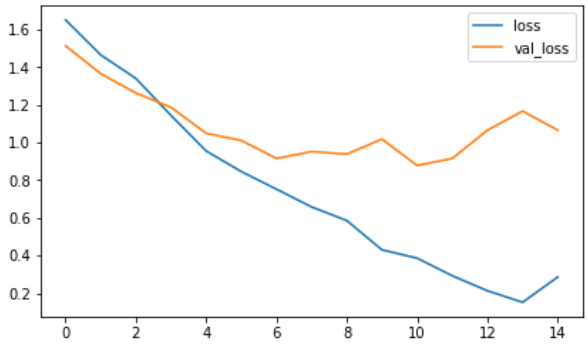
We stopped our investigation here. Best Model: model-00017-0.37124-0.87557-0.77778-0.72000.h5

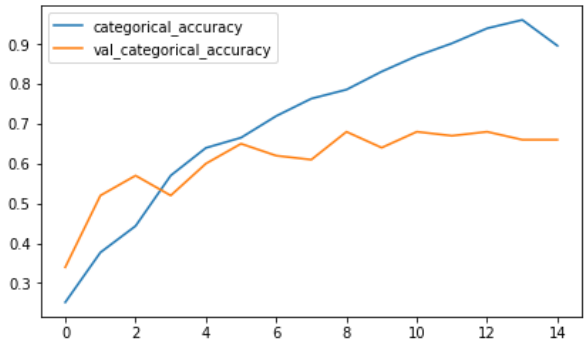
CNN + RNN Model:

For building a CNN + RNN model we chosen Transfer Learning option.

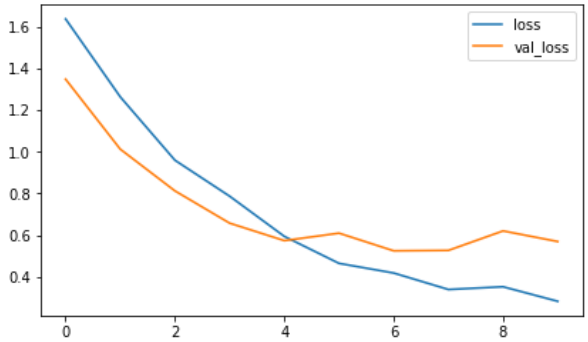
We kept the image size at (160x160) unchanged and batch size at 40 unchanged.

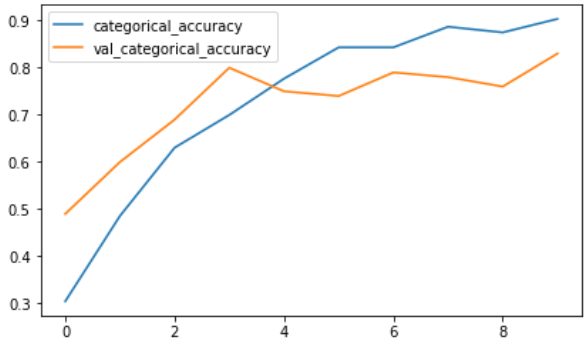
- We initially started with VGG Net model VGG16.
- We have built our **first model** with:
    - 1 Time Distributed Layer with Vgg16 model(include_top=False).
    - 1 Time Distributed Later with MaxPooling 2D(2,2) padding='same'
    - 1 GRU with 64 cells, dropout=0.25
    - 1 Dense layer with 64 neurons, dropout = 0.25
- When we ran the VGG16 model, we saw Out of Memory error. After multiple trial and error, we decided to keep the image size at 100x100 and batch size at 32.
- We ran our **first model** for 15 epochs and the results are as:



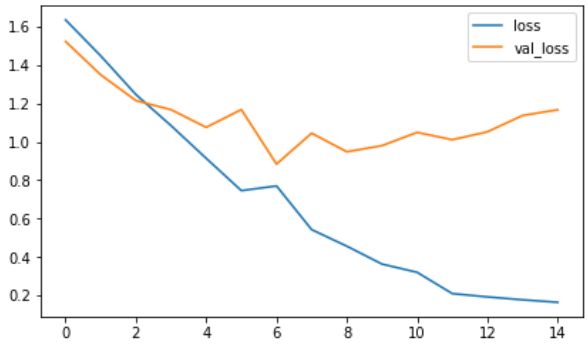


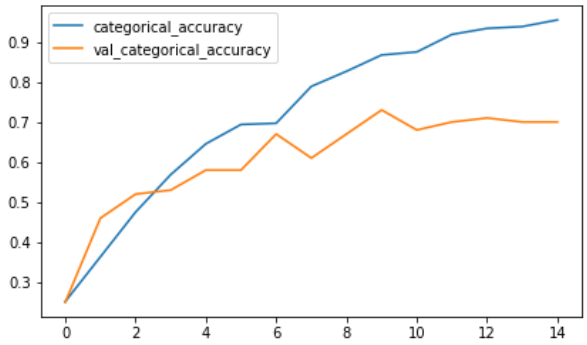
- We see a big difference with training and validation accuracy. Since we are using transfer learning, we expect far better results than Conv3D model.
- Hence, we decided to tryout **MobileNet**
- For our **second model** we have:
    - 1 Time Distributed Layer with mobilenet model(include_top=False).
    - 1 Time Distributed Later with MaxPooling 2D(2,2) padding='same'
    - 1 GRU with 64 cells, dropout=0.25
    - 1 Dense layer with 64 neurons, dropout = 0.25
- After running we understood that image size should be (128,128). We updated the image size. Batch size remains same as 32.
- We ran the second model and results as below:





- This time we got very good results.
    - Best Model's train data accuracy = 90.30%
    - Best Model's validation data accuracy = 83%
- This is ideally a good model as there is no overfitting and good accuracy on training and test data.
- Now, we decided to tryout the VGG16 and MobileNet models on augmented data.
- Our **third model** is with VGG16 + RNN on augmented data and results are as below:



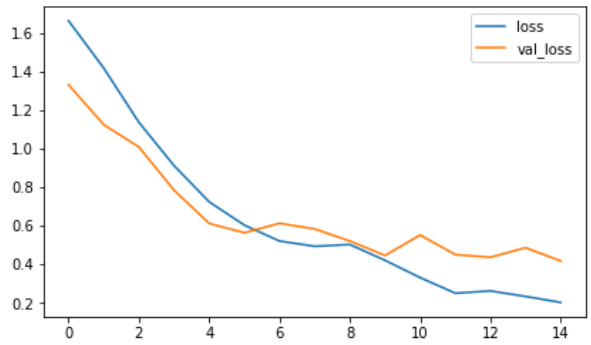


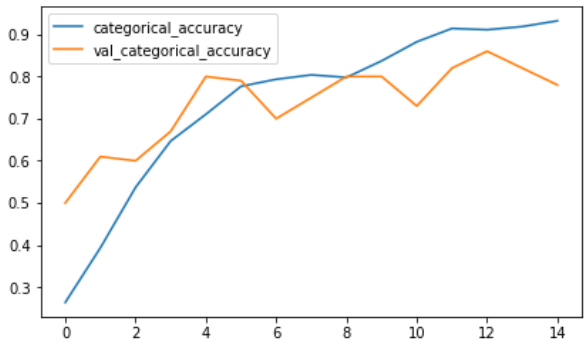
- We again got the best model with validation accuracy as 70%. This is not satisfying.
- We built our **final model** with augmented data using MobileNet + RNN

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
time_distributed_10 (TimeDis (None, 20, 4, 4, 1024)    3228864
_________________________________________________________________
time_distributed_11 (TimeDis (None, 20, 2, 2, 1024)    0
_________________________________________________________________
time_distributed_12 (TimeDis (None, 20, 4096)          0
_________________________________________________________________
gru_4 (GRU)                  (None, 128)               1622016
_________________________________________________________________
dropout_15 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_18 (Dense)             (None, 128)               16512
_________________________________________________________________
dropout_16 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_19 (Dense)             (None, 5)                 645
=================================================================
Total params: 4,868,037
Trainable params: 1,639,173
Non-trainable params: 3,228,864
```

We have 4,868,037 parameters out of which 1,639,173 are trainable. We did not train the MobileNet params

- Results are as below:





- We got a model with:
  - Train Accuracy: 91.10%
  - Validation Accuracy: 86.00%
  - Filename: model-00013-0.26162-0.91101-0.43656-0.86000.h5