# Gesture Recognition Project

# (on github: <https://github.com/that1girlCyborg/gesture_recognition> )

# Two types of architectures suggested for analysing videos using deep learning:

1. **3D Convolutional Neural Networks (Conv3D)**

*3D convolutions* are a natural extension to the 2D convolutions you are already familiar with. Just like in 2D conv, you move the filter in two directions (*x* and *y*), in 3D conv, you move the filter in three directions (*x*, *y* and *z*). In this case, the input to a 3D conv is a video (which is a sequence of 30 RGB images). If we assume that the shape of each image is *100 x 100 x 3*, for example, the video becomes a 4D tensor of shape *100 x 100 x 3 x 30* which can be written as *(100 x 100 x 30) x 3* where *3* is the number of channels. Hence, deriving the analogy from 2D convolutions where a 2D kernel/filter (a square filter) is represented as *(f x f) x c* where *f* is filter size and *c* is the number of channels, a 3D kernel/filter (a *'cubic'* filter) is represented as *(f x f x f) x c* (here *c = 3* since the input images have three channels). This cubic filter will now *'3D-convolve'* on each of the three channels of the *(100 x 100 x 30)* tensor

.

1. **CNN + RNN architecture**

The *conv2D* network will extract a feature vector for each image, and a sequence of these feature vectors is then fed to an RNN-based network. The output of the RNN is a regular softmax (for a classification problem such as this one).

# Data Generator

This is one of the most important part 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* and *120 x 160*) as well as create a batch of video frames. The generator should be able to take a batch of videos as input without any error. Steps like cropping, resizing and normalization should be performed successfully.

# Data Pre-processing

* ***Resizing* and *cropping* of the images.** This was mainly done to ensure that the NN only recognizes the gestures effectively rather than focusing on the other background noise present in the image.
* ***Normalization* of the images.** Normalizing the RGB values of an image can at times be a simple and effective way to get rid of distortions caused by lights and shadows in an image.
* At the later stages for improving the model’s accuracy, we have also made use of ***data augmentation***, where we have ***slightly rotated*** the pre-processed images of the gestures in order to bring in more data for the model to train on and to make it more generalizable in nature as sometimes the positioning of the hand won’t necessarily be within the camera frame always.

# NN Architecture development and training

* Experimented with different model configurations and hyper-parameters and various iterations and combinations of batch sizes, image dimensions, filter sizes, padding and stride length were experimented with. We also played around with different learning rates and *ReduceLROnPlateau* was used to decrease the learning rate if the monitored metrics (*val\_loss*) remains unchanged in between epochs.
* We experimented with *SGD()* and *Adam()* optimizers but went forward with *Adam()* as it lead to improvement in model’s accuracy by rectifying high variance in the model’s parameters. We were unsupportive of experimenting with *Adagrad()* and *Adadelta()* due to the limited computational capacity as these take a lot of time to converge because of their dynamic learning rate functionalities.
* We also made use of *Batch Normalization*, *pooling* and *dropout* *layers* when our model started to overfit, this could be easily witnessed when our model started giving poor validation accuracy inspite of having good training accuracy ☺.
* *Early stopping* was used to put a halt at the training process when the *val\_loss* would start to saturate / model’s performance would stop improving.

# Observations

* It was observed that as the Number of trainable parameters increase, the model takes much more time for training.
* **Batch size ∝ GPU memory / available compute.** A large batch size can throw *GPU Out of memory error,* and thus here we had to play around with the batch size till we were able to arrive at an optimal value of the batch size which our GPU could support ( NVIDIA Tesla K80 GPU with 12GB memory provided by nimblebox.ai platform.)
* Increasing the batch size greatly reduces the training time but this also has a negative impact on the model accuracy. This made us realise that there is always a trade-off here on basis of priority -> If we want our model to be ready in a shorter time span, choose larger batch size else you should choose lower batch size if you want your model to be more accurate.
* *Data Augmentation* and *Early stopping* greatly helped in overcoming the problem of overfitting which our initial version of model was facing.
* *CNN+LSTM* based model with *GRU* cells had better performance than *Conv3D.* As per our understanding, this is something which depends on the kind of data we used, the architecture we developed and the hyper-parameters we chose.
* *Transfer learning* **boosted** the overall accuracy of the model. We made use of the [*MobileNet*](https://arxiv.org/abs/1704.04861) Architecture due to it’s light weight design and high speed performance coupled with low maintenance as compared to other well-known architectures like VGG16, AlexNet, GoogleNet etc.
* For detailed information on the Observations and Inference, please refer Table 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MODEL** | **EXPERIMENT** | **RESULT** | **DECISION + EXPLANATION** | **PARAMETERS** |
| **Conv3D** | **1** | **OOM Error** | **Reduce the batch size and Reduce the number of neurons in Dense layer** | **-** |
| **2** | **Training Accuracy : 0.99**  **Validation Accuracy : 0.81** | **Overfitting ☹**  **Let’s add some Dropout Layers ☺** | **1,117,061** |
| **3** | **Training Accuracy : 0.65**  **Validation Accuracy : 0.52**  ***(Best weight Accuracy,Epoch:6/25)*** | **Val\_loss didn’t improve from 1.24219 so early stopping stop the training process. Let’s lower the learning rate to 0.0002.** | **3,638,981** |
| **4** | **Training Accuracy : 0.76**  **Validation Accuracy : 0.72**  ***(Best weight Accuracy,Epoch:12/25)*** | **Overfitting has reduced but accuracy hasn't improved. *Let's trying adding more layers*** | **1,762,613** |
| **5** | **Training Accuracy : 0.83**  **Validation Accuracy : 0.76** | ***Don’t see much performance improvement. Let's try adding dropouts.*** | **2,556,533** |
| **6** | **Training Accuracy : 0.84**  **Validation Accuracy : 0.69** | **Overfitting Increase, adding dropouts has further reduced validation accuracy. Let's try to reduce the parameters** | **2,556,533** |
| **7** | **Training Accuracy : 0.84 Validation Accuracy : 0.74** | **Overfitting reduced, but validation accuracy low. Let's try to reduce the parameters.**  **Val Accuracy: 0.49, Train Accuracy: 0.54** | **696,645** |
| **8** | **Training Accuracy : 0.82 Validation Accuracy : 0.73** | **Accuracy remains below same. Let’s switch to CNN+LSTM.** | **504,709** |
| **CNN+LSTM** | **9**  **(Model-8 on Notebook)** | **Training Accuracy : 0.93 Validation Accuracy : 0.85** | **CNN - LSTM model - we get a best validation accuracy of 85%.** | **1,657,445** |
| **Conv3D** | **Let's apply some Data Augmentation techniques & check the model performance** | | | |
| **10** | **Training Accuracy : 0.78 Validation Accuracy : 0.82** | **(3, 3, 3) Filter & 160 x 160 image resolution** | **3,638,981** |
| **11** | **Training Accuracy : 0.72 Validation Accuracy : 0.75** | **(2, 2, 2) Filter & 120 x 120 image resolution. Increase epoch count to 20. Network is generalizing well.** | **1,762,613** |
| **12** | **Training Accuracy : 0.87 Validation Accuracy : 0.78** | **Adding more layers.** | **2,556,533** |
| **13** | **Training Accuracy : 0.65 Validation Accuracy : 0.25** | **Very low performance. Let’s reduce the network parameters.** | **2,556,533** |
| **14** | **Training Accuracy : 0.89 Validation Accuracy : 0.78** | **After reducing network parameters, model’s performance is quite good.** | **696,645** |
| **15** | **Training Accuracy : 0.88 Validation Accuracy : 0.81** | **Reducing network parameters again.** | **504,709** |
| **CNN LSTM with GRU** | **16** | **Training Accuracy : 0.98 Validation Accuracy : 0.77** | **Overfitting is considerably high, not much improvement.** | **2,573,541** |
| **Transfer Learning(Optional)** | **17** | **Training Accuracy : 0.85 Validation Accuracy : 0.58** | ***We are not training the MobileNet weights that can see, validation accuracy is very poor.*** | **3,840,453** |
| **Transfer Learning with GRU &(Optional)** | **18** | **Training Accuracy : 0.98 Validation Accuracy : 0.93** | **Awesome result !!** | **3,692,869** |

After doing all the experiments, we finalized **Model 8– CNN+LSTM**, which performed well.

**Reason:**

* (Training Accuracy: 93%, Validation Accuracy: 85%)
* Number of Parameters (1,657,445) less according to other models’ performance
* Learning rate gradually decreasing after some Epochs

# Further suggestions for improvement:

* **Using Transfer Learning**:
* **Using GRU**
* **Deeper Understanding of Data**
* **Tuning hyperparameters**