## **Deep Learning Case Study - Gesture Recognition**

# **Problem Statement**

As a data scientist at a home electronics company which manufactures state of the art smart televisions. We want to develop a cool feature in the smart-TV that can recognise five different gestures performed by the user which will help users control the TV without using a remote.

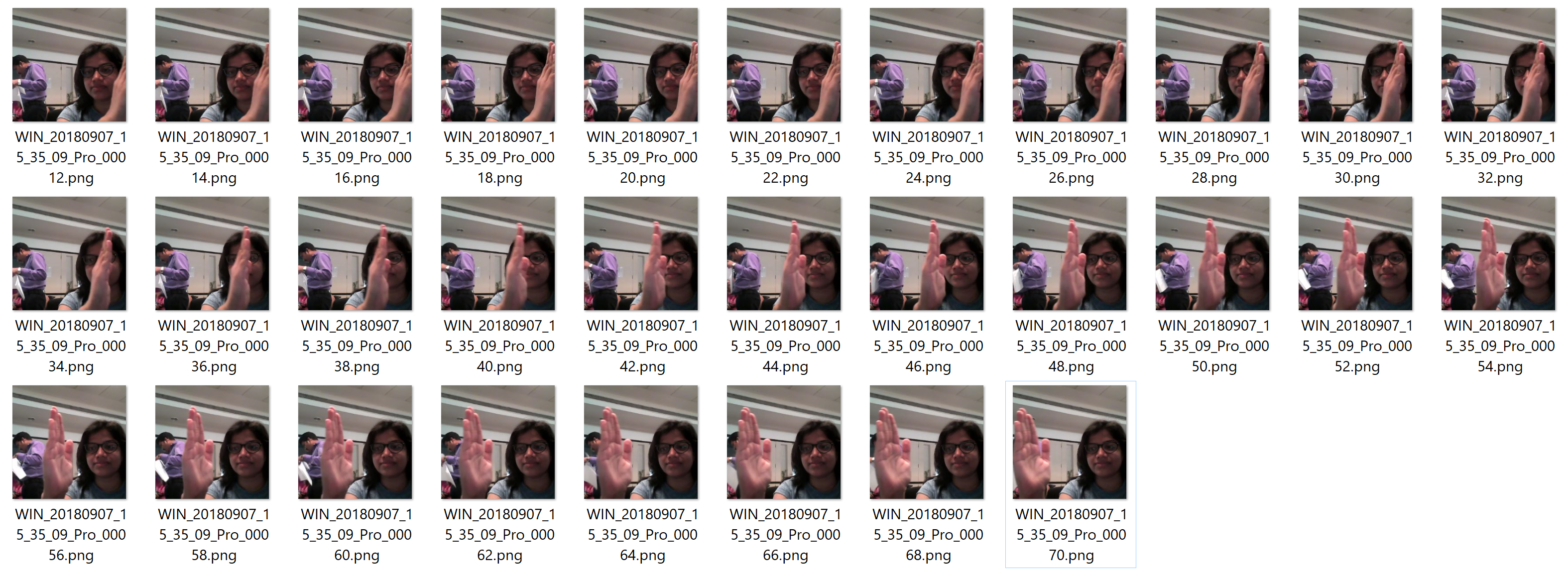
* Thumbs up :  Increase the volume.
* Thumbs down : Decrease the volume.
* Left swipe : 'Jump' backwards 10 seconds.
* Right swipe : 'Jump' forward 10 seconds.
* Stop : Pause the movie.

# **Understanding the Data**

**Link to data:** <https://drive.google.com/uc?id=1ehyrYBQ5rbQQe6yL4XbLWe3FMvuVUGiL>

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.

Sample of frames showing Right Swipe in the dataset is shown below.



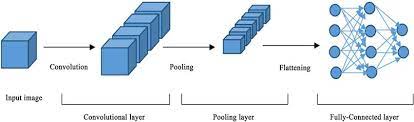
# **Objective**

Our task is 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 test folder for evaluation is withheld - final model's performance will be tested on the 'test' set.

# **Architectures suggested for analysing videos using deep learning**

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

*3D convolutions* are a natural extension to the 2D convolutions. 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.

****

**Simple representation of Conv3D CNN architecture**

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).

A close up of a sign

Description automatically generated

**Simple representation of CNN+RNN architecture**

# **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 *160 x 120*) 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 model development and training method**

* Before creating models, we experimented with different model configurations and hyper-parameters, various iterations, combinations of batch sizes, image dimensions, filter sizes, padding and stride length were experimented with. We also used different learning rates and *ReduceLROnPlateau* was used to decrease the learning rate if the monitored metrics (*val\_loss*) remains unchanged in between epochs.
* We used *Adam ()* optimizer as it led to improvement in model’s accuracy by rectifying high variance in the model’s parameters.
* We also made use of *Batch Normalization*, *pooling* and *dropout* *layers* when our model shown overfitting results. We also witnessed that our models started giving poor validation accuracy despite of having good training accuracy.
* *Early stopping* was used to put a halt at the training process when the *validation loss* would start to saturate / model’s performance would stop improving.
* *We executed the code on local machine as the environment to develop and test the models is common.*

# **Observations**

* It was observed that as the Number of trainable parameters increase, the model takes much more time for training.
* **Batch size Vs GPU memory -** 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 system could support.
* Number of samples, Image resolution have greater impact on training time compared to batch size. So, we tried to keep the batch size (allowed on our system) fixed and changed the number of samples and image size to get better performance of given model.
* *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 its light weight design and high-speed performance coupled with low maintenance as compared to other well-known architectures like VGG16, AlexNet, GoogleNet etc.

**Summary of results with various experimental models**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model Type** | **Model No** | **Result**  **(Best training accuracy & corresponding validation accuracy)** | **Remarks** | **Number of parameters(total)** |
| **Conv3D(CNN)** | **Base Model** | **OOM error** | **Resources exhausted for batch size=40 and frames to sample=30 for image resolution 160x160** | **1,736,389** |
| **1** | **Training Accuracy: 0.93**  **Validation Accuracy: 0.76** | **Overfitting model Use Data Augmentation & see** | **1,117,061** |
| **2** | **Training Accuracy :0.80**  **Validation Accuracy: 0.88** | **Overfitting is slightly reduced.**  **Reduce image size and filer size for stable, better performance** | **3,638,981** |
| **3** | **Training Accuracy :0.83**  **Validation Accuracy: 0.87** | **Reduced the parameters to 50% compared to earlier model.**  **No overfitting.**  **Add more layers to improve the performance further.** | **1,762,613** |
| **4** | **Training Accuracy :0.78**  **Validation Accuracy :0.89** | **No major improvement in the performance. Performance is not stable and resulted in underfitting in some epochs. Try changing the image resolution and see** | **2,556,533** |
| **5** | **Training Accuracy :0.85**  **Validation Accuracy :0.90** | **Decent performance in low footprint models.**  **Perform additional rotations in data augmentation and see if performance improves further.** | **504,709** |
| **6** | **Training Accuracy :0.84**  **Validation Accuracy :0.93** | **Model is not giving consistent result. It is underfitting in some epochs. Reduce learning rate, image size along with additional data augmentation** | **3,638,981** |
| **7** | **Training Accuracy :0.79**  **Validation Accuracy :0.87** | **Underfitting is reduced.**  **Decent performance.**  **Add additional layers and dropouts and see if performance improves further.** | **1,762,613** |
| **8** | **Training Accuracy :0.78**  **Validation Accuracy :0.39** | **Model is overfitting.**  **Addition of layers and dropouts didn’t help.**  **Reduce network parameters.** | **2,556,533** |
| **9** | **Training Accuracy :0.84**  **Validation Accuracy :0.89** | **No overfitting.**  **Decent accuracy for bot training and validation sets**  **Use CNN+RNN model.** | **230,949** |
| **CNN+RNN** | **10** | **Training Accuracy :0.97**  **Validation Accuracy :0.89** | **Overfitting model.**  **Use data augmentation and reduce overfitting.** | **1,657,445** |
| **CNN+LSTM with GRU** | **11** | **Training Accuracy :0.99**  **Validation Accuracy :0.91** | **Continues to be an overfitting model.**  **No major improvement. Let us try transfer learning.** | **2,573,541** |
| **CNN+RNN with transfer learning** | **12** | **Training Accuracy :0.99**  **Validation Accuracy :0.98** | **Model seems to be stable with less fluctuations in accuracy.**  **Decent accuracy** | **3,692,869** |
| **CNN+RNN+**  **transfer learning with GRU** | **13** | **Training Accuracy :0.98**  **Validation Accuracy :1.0** | **Looks like the model is overfitting. Perform hyperparameter tuning and see if it improves the performance and decreases overfitting.** | **3,446,533** |
| **CNN+RNN+**  **transfer learning with GRU** | **14** | **Training Accuracy :0.95**  **Validation Accuracy :0.96** | **Decent performance in transfer learning models.** | **3,446,533** |

**Decent Models**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model Type** | **Model No** | **Result**  **(Best training accuracy & corresponding validation accuracy)** | **Remarks** | **Number of parameters(total)** |
| **Conv3D** | **9** | **Training ccuracy:0.84**  **Validation accuracy:0.89** | **No major overfitting and stable model in low foot print models** | **230,949** |
| **Conv3D** | **3** | **Training accuracy:0.83**  **Validation accuracy:0.87** | **Decent model with low image resolution, learning rate and augmented data. No overfitting.** | **1,762,613** |
| **CNN+RNN+**  **transfer learning with GRU** | **14** | **Training Accuracy :0.95**  **Validation Accuracy :0.96** | **Good performance in transfer learning models.** | **3,446,533** |

**Final Model**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model Type** | **No** | **Result**  **(Best training accuracy & corresponding validation accuracy)** | **Remarks** | **Number of parameters(total)** |
| **CNN+RNN+**  **transfer learning with GRU** | **14** | **Training Accuracy :0.95**  **Validation Accuracy :0.96** | **Good performance in transfer learning models. Decent training and validation accuracy.** | **3,446,533** |

**Further steps/Recommendations:**

* **Using Transfer Learning**: Using a pre-trained *ResNet50/ResNet152/Inception V3* to identify the initial feature vectors and passing them further to a *RNN* for sequence information before finally passing it to a SoftMax layer for classification of gestures.
* **Using GRU:** A *GRU* model in place of *LSTM* appears to be a good choice. Trainable Parameters of a *GRU* are far less than that of a *LSTM*. However, its effect on the validation accuracies could be checked to determine if it is actually a good alternative over LSTM.
* **Deeper Understanding of Data:** The video clips were recorded in different backgrounds, lightings, persons and different cameras were used. Further exploration of available images could give some more information about them and bring more diversity in the dataset.
* **Tuning hyperparameters:** Experimenting with other combinations of hyperparameters like, activation functions (*ReLU, Leaky ReLU, tanh, sigmoid*), other optimizers like *Adagrad ()* and *Adadelta ()* can further help develop better and more accurate models. Experimenting with other combinations of hyperparameters like the *filter size, paddings, stride\_length, batch\_normalization, dropouts* etc. can further help improve performance.