## **Neural Networks Project- Gesture Recognition**

* **Mehul Kumawat – Group Facilitator**
* **Nasaraiah Vemula**

**Cohort: ML C29**

# 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.

**Here’s the data:**

# Understanding the Dataset

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.

**A picture containing photo, many, various, sitting

Description automatically generated**

# 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.

# 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

.

**A close up of a box

Description automatically generated**

**30 frames….**

 **Depth**

**Error**

**A picture containing person, woman, holding, sitting

Description automatically generated**

**Conv3D**

**Back**

**Propagation**

**RGB**

***e****.g****.*** *(100 x 100 x 3 x 30)*

**Update**

**Figure 1: A simple representation of working of a 3D-CNN**

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

**Figure 2: A simple representation of an ensembled CNN+LSTM 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 *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*** using the technique ‘***warpAffine’***, where pre-processed images of the gestures 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 P100 GPU with High RAM memory provided by *‘Google Colab pro+’* 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** |
| **-** | **-** | **OOM Error** | **- Reduce the batch size**  **- Reduce the image dimensions**  **- Reduce the number of frames** | **-** |
| **Conv 3D (Base Model)**  **No Data Augmentation**    **Batch Size: 40 Epoch: 15** | **1** | **Training Accuracy : 0.97**  **Validation Accuracy : 0.24** | **Overfitting:**  **1.Add some Dropout Layers**  **2.Regularization should be implemented to**  **overcome the problem of Overfitting**  **3.Need to do Data Augmentation** | **1,117,061** |
| **Conv 3D (Model - 2)**  **Data Augmentation**    **Batch Size: 20 Epoch: 25** | **2** | **Training Accuracy : 0.76**  **Validation Accuracy : 0.75**  ***(Best weight Accuracy, Epoch: 24/25)*** | **Model is not overfitting, and we get a best validation accuracy of 76% and training accuracy of 75%.  Next, we will try to reduce the filter size and image resolution and see if we get better results. Moreover, since we see minor oscillations in loss, let's try lowering the learning rate to 0.0002.** | **3,638,981** |
| **Conv 3D (Model - 3)**  **Reduce filter size to (2,2,2)**    **Batch Size: 30 Epoch: 30** | **3** | **Training Accuracy : 0.76**  **Validation Accuracy : 0.75** | **Model has a best validation accuracy of 76% and training accuracy of 75%. Also, we were able to reduce the parameter size by half the earlier model. Let's trying to add more layers.** | **1,762,613** |
| **Conv 3D (Model - 4)**  **Adding more layers**    **Batch Size: 20 Epoch: 30** | **4** | **Training Accuracy : 0.78**  **Validation Accuracy : 0.75** | **Don’t see much performance improvement. Let's try adding dropouts.** | **2,556,533** |
| **Conv 3D (Model - 5)**  **Adding dropouts**    **Batch Size: 20 Epoch: 22** | **5** | **Training Accuracy : 0.83**  **Validation Accuracy : 0.52** | **Overfitting Increase, adding dropouts has further reduced validation accuracy. Let's try to reduce the parameters** | **2,556,533** |
| **Conv 3D (Model - 6)**  **Reducing no. of parameters**    **Batch Size: 20 Epoch: 30** | **6** | **Training Accuracy : 0.79**  **Validation Accuracy : 0.74**  ***(Best weight Accuracy, Epoch: 27/30)*** | **Overfitting decrease, decent validation accuracy of 74% with significant drop in the number of parameters.** | **696,645** |
| **Conv 3D (Model - 7)**  **Reducing no. of parameters**    **Batch Size: 20 Epoch: 25** | **7** | **Training Accuracy : 0.76**  **Validation Accuracy : 0.84** | **Further decrease in number of parameters, observed a increase in the validation score to 84%.** | **504,709** |
| **Conv 3D (Model - 8)**  **Reducing no. of parameters**    **Batch Size: 20 Epoch: 30** | **8** | **Training Accuracy : 0.68**  **Validation Accuracy : 0.58**  ***(Best weight Accuracy, Epoch: 28/30)*** | **Observed that both the training and validation accuracy had been reduced significantly by reducing the number of parameters. Let’s try with CNN+LSTM Model.** | **230,949** |
| **CNN+LSTM**  **(Model - 9)**  **Batch Size: 20 Epoch: 20** | **9** | **Training Accuracy : 0.86**  **Validation Accuracy : 0.73** | **CNN + LSTM model - we get a best validation accuracy of 73%.** | **1,657,445** |
| **CNN+ GRU**  **(Model - 10)**  **Batch Size: 20 Epoch: 20** | **10** | **Training Accuracy : 0.93**  **Validation Accuracy : 0.77** | **CNN + GRU:**  **Observed that overfitting is considerably less when we do more augmentation. However, there is not much improvement on accuracy** | **2,573,925** |
| **Transfer Learning**  **(Model - 11)**  **Batch Size: 5 Epoch: 20** | **11** | **Training Accuracy : 0.99**  **Validation Accuracy : 0.72** | **Transfer Learning:**  **We are not training the ‘Mobilenet’ weights, and we see validation accuracy is very poor. Let's train them as well and observe if there is performance improvement** | **3,840,453** |
| **Transfer Learning + GRU**  **(Model - 12)**  **Batch Size: 5 Epoch: 20** | **12** | **Training Accuracy : 0.98**  **Validation Accuracy : 0.97**  ***(Best weight Accuracy, Epoch: 18/20)*** | **Transfer Learning+GRU:**  **When trained all the weights with transfer learning + GRU, we are getting the best validation score of 97%.** | **3,693,253** |

**Table 1: Observations and Results for numerous tested NN architectures**

After doing all the experiments, we finalized **Model 12 – Transfer Learning+ GRU**, which performed well.

**Reason:**

* (Training Accuracy: 98%, Validation Accuracy: 97%)
* Number of Parameters (3,693,253) less according to other models’ performance
* Learning rate gradually decreasing after some Epochs

# Further suggestions for improvement:

* **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. (This was attempted but other pre-trained models couldn’t be tested due to lack of time and disk space in the nimblebox.ai platform.)
* **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*. Therefore would have resulted in faster computations. 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 where recorded in different backgrounds, lightings, persons and different cameras where used. Further exploration on the available images could give some more information about them and bring more diversity in the dataset. This added information can be exploited in favour inside the generator function adding more stability and accuracy to model.
* **Tuning hyperparameters:** Experimenting with other combinations of hyperparameters like, activation functions (*ReLU, Leaky ReLU, mish, 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.

**\*\* END OF THE DOCUMENT \*\***