Deep Learning Course Project- Gesture Recognition

- **Kuldeep Pawar – Group Facilitator**
- **Kuldeep Pawar**

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 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	: 'Jump' backwards 10 seconds.
Right swipe	: 'Jump' forward 10 seconds.
Stop	: Pause the movie.

Here's the data: https://drive.google.com/uc?id=1ehyrYBQ5rbQQe6yL4XbLWe3FMvuVUGiL

Understanding the Dataset

The training data consists of a few hundred videos categorised 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

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 Architectures: 3D Conv and CNN-RNN Stack

We will be using two types of architectures for analyzing videos using deep learning

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.

2. **3D Convolutional Neural Networks (Conv3D**
   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

Data preprocessing

- Deciding on number of images to be taken per video/sequence
- **Resizing and cropping the images** : This was mainly done to ensure that the Neural network only recognizes the gestures effectively rather than focusing on the other background noise present in the image. We used 120 by 120 image size after experimenting with different sizes.
- **Normalizing 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.

Data Generators

In this project, we have written our own batch data generator. After iterating over the chosen batch size, we have iterated over the remaining data points which were left after full batches.

Neural Network Architecture development and training

- We started by building a basic **CNN+RNN model** without using any dropouts. **GRU** was selected as it has fewer parameters than LSTM, as it lacks an output gate.
- We experimented with different models by using different model configurations and hyperparameters such as introducing dropouts, using L2 regularization in combination with different optimizers.
- We also experimented with *SGD()* and *Adam()* optimizers but finally used *Adam()* as it lead to improvement in model's accuracy. Due to the limited computational capacity we could not experiment with other optimizers such as *Adagrad* and *Adadelta* as these take a lot of time to run.

- We also played around with different learning rates and *ReduceLROnPlateau* was used to decrease the learning rate .
- To overcome the issue of overfitting we used *Batch Normalization*, *pooling* and *dropout layers*.
- Then we used Conv3D and experimented with different model configurations to arrive at the final model.

Observations

- The training time increased in proportion with the number of trainable parameters.
- A large batch size value was giving GPU Out of memory error. So we started with a batch size of 15 and gradually moved till batch size of 25. A small batch helped reducing training time but it had a trade off on model accuracy.
- *Transfer learning* **boosted** the overall accuracy of the model. We made use of the **Resnet50 ,VGGNET and** *MobileNet* Architecture . We took the model with MobileNet architecture as the final model 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.
- For detailed information on the Observations and Inference, please refer Table 1.

Experiment Number	Model	Accuracy	Result	Decision + Explanation
1	**Conv2D layers + GRU Layer+ Adam optimizer + No dropout**	**Training .99 Validation .74**	**Overfitting**	**Training accuracy was high. So, we introduced dropouts.**
2	**Conv2D layers + GRU Layer+ Adam optimiser + Dropout**	**Training .93 Validation .75**	**The overfitting slightly reduced but was still high**	**We thought of employing L2 regularization in Model 2 in next step**
3	**Conv2D layers + GRU Layer+ Adam optimiser + dropout + L2 regularization**	**Training .95 Validation .69**	**Val Accuracy Dropped**	**Still overfitting. We need to change the optimizer to SDG for the next steps**
4	**Conv2D layers + GRU Layer+ SDG optimiser + dropout + L2 regularization**	**Training .76 Validation .54**	**Overfitting problem resolved to some extent; however training / validation accuracy is quite low**	**We reverted the SDG optimizer as it reduced the accuracy on training / validation data and continued experimenting with Adam optimizer**
5	**Conv3D layers + Dropout + Batch**	**Training .59 Validation**	**The accuracy on train and validation data becomes almost**	**As the Accuracy got reduced, we experimented**

	Normalization + L2 Regularization	**.58**	**equal, but its seems to be underfitting**	**with Transfer Learning in next step.**
6	**Transfer Learning + Resnet50 + dropout + GRU Layer**	**Training .69 Validation .60**	**Number of trainable parameters became quite high (25,126,469)**	**Let Experiment with Another NN : VGGNET**
7	**Transfer Learning + VGGNET + dropout + GRU Layer**	**Training .53 Validation .59**	**Underfitting observed**	**Let Experiment with Another NN : mobilenet**
8	**Transfer Learning + mobilenet + dropout + GRU Layer**	**Training .97 Validation .98**	**High Accuracy both on train / validation data**	**Accuracy Improved with MobileNet Arch, as its has lightweight design & high speed performance as compared to other used Arch.**
9	**Transfer Learning + mobilenet + dropout + GRU Layer + Increased epoch (30)**	**Training .98 Validation .90**	**Increasing epoch does not increased validation accuracy.**	**Let experiment with increasing batch size**
10	**Transfer Learning + mobilenet + dropout + GRU Layer + Increased epoch + Increased Batch Size(20)**	**Training .98 Validation .97**	**High Accuracy on Training / validation Data**	**Selected Model**
11	**Transfer Learning + mobilenet + dropout + GRU Layer + Increased Batch Size(25)**	**Training .97 Validation .96**	**High Accuracy on Training / validation Data**	**Still Model 10, seems to be better.**

After doing all the experiments, we finalized **Model 10– Transfer Learning + mobilenet + dropout + GRU Layer + Increased epoch + Increased Batch Size(20)**, which performed well.

Reason:

- (Training Accuracy: 98%, Validation Accuracy: 96%)
- Number of Parameters (3,420,549) less according to other models' performance

Further suggestions for improvement:

- **Data augmentation :** Exploration by building model over augmented data(flipped,rotated) could have resulted into deeper insight of the model performance.
- **Tuning Hyperparamaters :** Experimentation with different hyperparameters such as different activation functions such as , *Leaky ReLU, tanh, sigmoid, optimizers like Adagrad and Adadelta,*

combination of different hyperparameters like dropouts, filter size, paddings, stride length etc. can further help in boosting performance.