Deep Learning Course Project- Gesture Recognition

- **Hamza Tanveer**
- **Harikrushn Raiyani**

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 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.

Objective

Our task is to train different models on the 'train' data to predict the action performed in each sequence or video and which performs well on the 'validation' data as well. The final test folder for evaluation is withheld - final model's performance will be tested on the 'test' data.

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. 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*. 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.

2. **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(20 degree range)*** 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 were experimented with. We also played around with *ReduceLROnPlateau* was used to decrease the learning rate if the monitored metrics (*val_loss*) remains unchanged in continuous **3 epochs**.
- We 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 *RMSProp* 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 in 6 epochs.

Observations

- It was observed that as the Number of trainable parameters increase, the model takes much more time for training.
- 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.
- 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+GRU* based model 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*** 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. As we have less number of train data, we have to train weights of MobileNet and it has resulted to increase validation accuracy drastically.

- For detailed information on the Observations and Inference, please refer Table 1.

MODEL	**EXPERIMENT**	**RESULT**	**DECISION + EXPLANATION**
Conv3D	1	OOM Error	• Reduce the batch size and Reduce the number of neurons in Dense layer
Conv3D	2 • Batch size 50 ; Epochs 25 • Conv layer with 16,32,64 neurons • Dense Layer with 64,32 neurons • Dropout 0.25 at Both layers	• Parameters : 1,792,981 • Training Accuracy : 0.54 • Validation Accuracy : 0.16	• Early stopping of model training as val_loss did not improved; • Reduce number of neurons in Dense layer and increase drop out to 0.4
Conv3D	3 • Batch 30 ; Epoch 25; • Conv layer with16,32,64 neurons; • Dense layer with 32,16 neurons ; Dropout of 0.4 at both layers	• Parameters : 906,581 • Training Accuracy : 0.21 • Validation Accuracy : 0.18	• Early stopping of model training as val_loss did not improved • Let's switch to CNN + RNN
CNN+LSTM	• Batch size 10, epochs 25 • Conv2d with 8 & 16 units, ConvLSTM2D layer with 8 units & Dense Layer with 64 neurones	• Parameters : 13,781 • Training Accuracy : 0.70 • Validation Accuracy : 0.58	• Early stopping of model training • Let's add more neurons/ units in layers as training accuracy is lower
CNN RNN with GRU	Batch size 15, epochs 25 - Conv2d with 32,32,64 & 128 units, GRU layer with 128 units & Dense Layer with 128 neurons - Drop out in all layer 0.4	• Parameters : 7,247,077 • Training Accuracy : 0.97 • Validation Accuracy : 0.50	• Early stopping of model training as val_loss did not improved • Clear sign of overfitting. • Let's reduce neurones in Dense layer and include BatchNormalization in Conv layer
CNN RNN with GRU	Batch size 15, epochs 25 - Conv2d with 32,32,64 & 128 units, GRU layer with 128 units & Dense Layer with 64 neurons(reduced) -Drop out for conv layer is 0.3 & for Dense Layer 0.4 (increased) - BatchNormalization in Conv layers	• Parameters : 13,781 • Training Accuracy : 0.23 • Validation Accuracy : 0.20	• Early stopping of model training as val_loss did not improved
CNN RNN with GRU	• Batch size 15, epochs 25	• Parameters : 1,810,021 • Training Accuracy : 0.98	• Clear sign of overfitting.

	• Conv2d with 16,32,64units, GRU layer with 64 units & Dense Layer with 64 neurons	• Validation Accuracy : 0.58	• As we have overfitting let's introduce MobileNet with LSTM
MobileNet + LSTM	• No augmentation • Batch size 20, epochs 25 • MobileNet, LSTM layer with 128 units & Dense Layer with 128 neurons • Dropout 0.25 at Conv Layers & Dense Layers	• Parameters : 3,840,453 • Training Accuracy : 0.95 • Validation Accuracy : 0.58	• Clear sign of overfitting. • As number of parameters are high, let's try GRU with reduced units in GRU layer
MobileNet + GRU	• Batch size 10, epochs 25 • MobileNet, GRU layer with 64 units & Dense Layer with 64 neurons • Dropout 0.25 at Conv Layers& Dense Layers	• Parameters : 3,446,533 • Training Accuracy : 0.94 • Validation Accuracy : 0.68	• Overfitting is reduced by some fraction but still it is to be addressed • We are going to train MobileNet weights as we have less train data
Transfer Learning with GRU (With training MobileNet)	• Batch size 10, epochs 30 • MobileNet, GRU layer with 32 units & Dense Layer with 32 neurons • Dropout 0.3 at Conv Layers & 0.4 at Dense Layers	• Parameters : 3,692,869 • Training Accuracy : 0.95 Validation Accuracy : 0.87	• Overfitting has reduced • Awesome result!!

Further suggestions for improvement:

- **Tuning hyperparameters:** Experimenting with other combinations of hyperparameters like, activation functions (*Leaky ReLU, mish, tanh*), other optimizers like *Adagrad* 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 ****