Gesture Recognition Project Description

By- Divya Jayendrakumar Sodha

Problem Statement

As a data scientist at a home electronics company which manufactures state of the art smart televisions, we have 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.

Types of architectures suggested for analysing videos using deep learning:

1. 3D Convolutional Neural Networks (Conv3D)
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 parts 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 are performed.

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.

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.
- 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 $\propto$ GPU memory / available compute. A large batch size can throw GPU Out of memory error
- 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.
- CNN+LSTM based model with GRU cells along with Transfer Learning and data augmentation gave the best results.
- Transfer learning boosted the overall accuracy of the model. MobileNet Architecture was used 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.

SAMPLE MODEL

Note:-

1) We set the batch size as 32 on the basis of GPU utilization capacity. We kept increasing the batch size until the machine threw an error.
2) Experimented with different values of image dimensions and frame sizes to decide based on the trade-off between number of parameters and accuracy, for small number of epochs (5 epochs).
3) Here, we tried to keep the parameters in power of 2 since it is efficient for computation.

Experiment Number	Model	Result
1	**Conv3D** Image_height = 128 Image_width = 128 Frames_to_sample = 16 No of epochs = 5	No of parameters = 14816901 Test Accuracy = 0.5600000023841858
2	**Conv3D** Image_height = 128 Image_width = 128 Frames_to_sample = 20 No of epochs = 5	No of parameters = 22189701 Test Accuracy = 0.20999999344348907
3	**Conv3D** Image_height = 64 Image_width = 64 Frames_to_sample = 16 No of epochs = 5	No of parameters = 3282565 Test Accuracy = 0.5
4	**Conv3D** Image_height = 64 Image_width = 64 Frames_to_sample = 20 No of epochs = 5	No of parameters = 4888197 Test Accuracy = 0.20999999344348907

Inference : -

1) Batch Size = 32
2) Image_height = 64
3) Image_width = 64
4) Frames_to_sample = 16

Experiment Number	Model	Result	Decision + Explanation
1	**Model - 1** **Conv3D** (Without Augmentation) No of parameters = 708613 No of Epochs = 40	Test Accuracy : 0.7400000095367432 Test Loss: 0.8421393632888794	**Explanation:** Trained the model with above parameters. Model was overfitting, as based on plot. **Decision:** To try augmentation for overfitting (Image cropping + Resizing)
2	**Model - 1** **Conv3D** (With Augmentation) No of parameters = 708613 No of Epochs = 40	Test Accuracy : 0.6499999761581421 Test Loss: 0.8482721447944641	**Explanation:** Tried to overcome overfitting through augmentation techniques. No improvement found. **Decision:** Switch to different architecture
3	**Model - 2** **Conv2D + GRU** (Without Augmentation) No of parameters = 98693 No of Epochs = 40	Test Accuracy : 0.7300000190734863 Test Loss: 0.6915207505226135	**Explanation:** Used CNN+RNN architecture (GRU to be specific) The model accuracy was not up to the expectation. It stopped by early stopping **Decision:** To try LSTM instead of GRU
4	**Model - 3** **Conv2D + LSTM** (Without Augmentation) No of parameters = 675877 No of Epochs = 40	Test Accuracy : 0.5199999809265137 Test Loss: 1.073970079421997	**Explanation:** The accuracy on test set was too low and observed overfitting as well **Decision:** To try augmentation techniques (Image cropping + Resizing)
5	**Model - 3** **Conv2D + LSTM** (With Augmentation) No of parameters = 675877 No of Epochs = 40	Test Accuracy : 0.6299999952316284 Test Loss: 0.9589130878448486	**Explanation:** The model accuracy improved a bit. But still not up to the mark. **Decision:** To add transfer Learning.
6	**Model -4** **Transfer Learning + Conv2D + LSTM + Data Augmentation** No of parameters = 3840453 No of Epochs = 20	Test Accuracy : 0.9200000166893005 Test Loss: 0.2726801931858063	**Explanation:** Used MobileNet Architecture and fine-tuned using Conv2D+LSTM architecture along with data augmentation. Accuracy increased quite significantly. **Decision:** To also try GRU instead of LSTM and parameters will also reduce a bit.
7 **(Final Model)**	**Model -5** **Transfer Learning + Conv2D + GRU + Data Augmentation** No of parameters = 3693253 No of Epochs = 20	Test Accuracy : 0.9200000166893005 Test Loss: 0.23567628860473633	**Explanation:** Quite good accuracy on test set, similar to Above model. **Decision:** Considered this as the best model as parameters as well loss value are a bit less as compared to the previous model.

Conclusion

- CNN+LSTM based model with GRU cells along with Transfer Learning and data augmentation gave the good results considering parameter size, accuracy and loss functions trade-off.
- Transfer learning using MobileNet boosted the overall accuracy of the model.