Gesture Recognition

By Rishabh Vij

Problem Statement:

Design and Train a Gesture Recognition model to detect 5 different gestures from sequences(videos).

Solution and Strategy:

Given around 650 sequences (videos) as a part of training data, my aim was to design and train a deep neural network that could recognise gestures with decent accuracy, while keeping the total parameter number small so that the model could be used on edge.

I explored different model schemas and architecture to solve this problem. To be specific I tried the following models with different architectures and setups:

- 3D convolutions
- 2D convolutions with Conv LSTM
- 2D convolutions with GRU
- 2D convolutions with GRU (using pre-trained mobilenet CNN)

Initially I started using architectures involving 3D convolutions with the data of around 650 videos we were provided with, but quickly realised that the data provided wasn't enough for the task in hand and I'd need to increase the training data using some strategy. That strategy is data augmentation. Then I re-designed the generator in such a way that it could increase the data by x- fold (3 in most cases) by applying random augmentation techniques. The augmentation techniques used by me were:

- Adding random noise
- Adding random blur
- Adding random gamma
- Logarithmic correction

- Sigmoid correction
- Equalizing hist
- Equalising adaptive hist

I explored multiple augmentation techniques apart from the ones already mentioned, but ended up using the stated linear augmentations only, as they seemed to work best for our use case.

The generator was designed such that it'd increase training data x-fold, with x being a parameter it accepts and applies the augmentations using the following strategy:

- Of the upscaled images, ⅓ samples are left as is and no augmentation is applied to them
- ⅓ images are applied with a random amount of noise, blur or gamma
- ⅓ images go through the various correction techniques specified

No augmentations were applied on the validation data of 100 sequences

After upsampling the training records with Augmentation, the models showed significant improvements in terms of accuracy and loss.

But using augmentations introduced another side effect with it, the models would overfit easily. The following steps were performed to keep overfitting in check:

- Using shallower models
- Using dropouts
- Exploring regularization in optimisers

Data Selection

The data consisted of 30 frames/images per video. After studying the patterns in few if the randomly selected sequences, it was decided to use only 15 frames per sequence. The frames were selected with an alternating frame strategy.

The videos were of two sizes with the smallest frames in size 120, 120 so the decision was taken to resize all of the data to 128, 128 for all models so as to not distort frames of lesser dimensions while resizing. All models except for one which leveraged transfer

learning were trained on images resized to the aforementioned size. The model leveraging transfer learning was trained on image shape 224, 224.

Batch size z64 and 128 were used. It was kept low to control overfitting of the training data.

The following table gives a brief overview of some of the models I had tried. The table below is not exhaustive and does not include all the models and details around them that I tried. In all, I tried around 25 different models with different architectures and hyperparameters. The details can be found in the Nootebook:

Experiment Number	Augmentation Used	Model	Result	Summary
1	**No**	**Conv3D**	**Accuracy: train : 93% val: 21%**	**Model was capable of learning the patterns but was overfitting**
2	**No**	**Conv3D with dropout**	**Accuracy: train : 75% val: 21%**	**Reduce the size of the image/Reduce the number of layers**
3	**No**	**Conv3D with dropout (deeper model)**	**Accuracy: train : 83% val: 23%**	**The model was still overfitting**
4	**Yes**	**Conv3d with dropout (shallow model)**	**Accuracy: train : 25% val: 67%**	**The model performance was not good and was overfitting**
5	**Yes**	**Conv3d with dropout (deeper model)**	**Accuracy: train : 72% val: 73%**	**This model was decent and wasn't overfitting (best results we got using conv3d)**
6	**Yes**	**Conv2d + GRU (deeper model)**	**Accuracy: train : 41% val: 47%**	**The results weren't very encouraging**
7	**Yes**	**Conv2D + GRU (shallow model)**	**Accuracy: train : 81% val: 80%**	**This is the best model using GRUs**

8	**Yes**	**Conv2d + ConvLSTM**	**Accuracy: train : 40% val: 59%**	**This was the most light-weight model I tried with only ~13k params. But the accuracy wasn't very convincing**
9th	**Yes**	**Conv2D + GRU (using mobilenet pre-trained CNN)**	**Accuracy: train : 33% val: 23%**	**This model was very slow to train and the performance wasn't good.**
10th	**Yes**	**Conv2D + Bidirectional GRU (shallow model)**	**Accuracy: train : 75% val: 76%**	**Second best model with. here bidirectional GRU was used.**

Conclusion:

Although the data provided to us was pretty less to develop a deep model, I was able to increase the training data 3-fold with the help of 8 different linear augmentation strategies. Though that introduced increase in models' tendency to overfit the training data, it was controlled using smaller batch sizes and heavy dropouts.

Different models with different architectures were tried, ranging from simple models with less than 15,000 total parameters to complex models with just over a million parameters. Finally, we were able to get the best results with 2D convolution with GRUs which gave us around **80% accuracy** on validation data. Given more resources, this model can be further fine-tuned to improve its performance.