Gesture Case Study

Phuc Thanh Nguyen Email: phucnguyenthanh.stu@gmail.com

Kuldeep Deol Email: Email: kuldeepdeol2011@gmail.com

1. Problem Statement

Imagine you are working as a data scientist at a home electronics company which manufactures state of the art smart televisions. You 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.

The gestures are continuously monitored by the webcam mounted on the TV. Each gesture corresponds to a specific command:

1. Thumbs up: Increase the volume
2. Thumbs down: Decrease the volume
3. Left swipe: 'Jump' backwards 10 seconds
4. Right swipe: 'Jump' forward 10 seconds
5. Stop: Pause the movie

Each video is a sequence of 30 frames (or images)

2. 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 - like what the smart TV will use.

3. 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.

4. Architectures used:

- Convolutions + RNN

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).

1. Base model without Transfer Learning
2. CNN-LSTM model with Transfer learning

- 3D Convolutional Network, or 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 15 RGB images). If we assume that the shape of each image is 128x128x3, for example, the video becomes a 4-D tensor of shape 128x128x3x15 which can be written as (128x128x15) x3 where 3 is the number of channels. Hence, deriving the analogy from 2-D convolutions where a 2-D kernel/filter (a square filter) is represented as (fxf)xc where f is filter size and c is the number of channels, a 3-D kernel/filter (a 'cubic' filter) is represented as (fxfxf)xc (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 (128x128x15) tensor.

- Training process: Use **Adam** for better accuracy then **SGD**. Use **ReduceonPlateau** as learning rate scheduler.

5. Data preprocessing:

• Deciding on number of images to be taken per video/sequence: 15 images from 30 frames

• 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 128 by 128 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.

6. 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 8 for all models.

• Transfer learning boosted the overall accuracy of the model. We made use of the Resnet50 , VGGNET and Mobile Net Architecture. We took the model with Mobile Net architecture as the final model due to its lightweight design and high-speed performance coupled with low maintenance as compared to other well-known architectures like VGG16, Alex Net, Google Net etc.

• For detailed information on the Observations and Inference, please refer Table below.

Experiment Number	Model	Total parameter	Trainable parameter	Train Accuracy	Validation Accuracy	Decision and Result
1	C3D liked	3M	3M	20.51%	28.00%	Try a very famous structure C3D like, but model underfit
2	3D CNN VGG liked	3.7M	3.7M	20.66%	25.00%	Changing with layer look like VGG, but model underfit
3	Custom 3D CNN	1M	1M	97.59%	86.00%	Custom 3DCNN with lots of BatchNorm and Dropout. Model is quite good
4	2DCNN_1 + LSTM	6.8M	6.8M	20.97%	29.00%	Custom 2DCNN with LSTM Model underfit

5	2DCNN_2 + GRU	1.2M	1.2M	20.51%	25.00%	Another model above but change from LSTM to GRU Cannot out of underfit problem
6	2DCNN_3 + GRU	0.2M	0.2M	99.10%	87.00%	Another custom CNN with GRU, use more dropout. Model performs better.
7	VGG16 + GRU	15M	0.4M	99.10%	86.00%	Same architect of classifier and input, batch_size. Only use different pretrained models (also freeze them). VGG16 and MobileNetv2 are better than the others. However, MobileNetv2 is the best.
8	Resnet50 + GRU	15M	0.4M	99.10%	83.00%	
9	EfficientNetv2B0 + GRU	6.9M	1M	19.61%	23.00%	
10	EfficientNetv2B1 + GRU	7.9M	1M	20.21%	26.00%	
11	MobileNetv2 + GRU	3.2M	1M	96.38%	92.00%	
12	XceptionNet + GRU	22.4M	1.5M	92.31%	70.00%	
13	MobileNetv2 + LSTM	3.6M	1.3M	96.98%	90.00%	Try to change from GRU to LSTM for MobileNetv2 pretrained.

7. Conclusion:

- Best model for val_acc is pretrained MobileNetv2 + GRU, about 92% for val_acc
- If running on edge device, we can chooise model CNN_03, because it only has total 200.000 para, very light weight but still good enough (87% val acc)
- For further improvement, we may need more data, data augmentation, larger batch_size, etc