Neural Networks Project - Gesture Recognition

- Rahul Prashar (ML C39)

Problem Statement:

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:

- 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 - like what the smart TV will use.

Suggested Solution Methods:

1. **3D Convolutional Neural Networks (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 30 RGB images). If we assume that the shape of each image is 100x100x3, for example, the video becomes a 4-D tensor of shape 100x100x3x30 which can be written as (100x100x30)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 (100x100x30) tensor.
2. **2D CNN + 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).

Preprocessing:

Creating Generators:

- In most deep learning projects you need to feed data to the model in batches to save memory, its done by using generators
- While we have Generators readily available in libraries such as Keras, we will build the generator from the scratch as we should be able to use this generator with other cases as well
- Enabled by python Yield statement called repeatedly, the generator will create batches and pass the last saved state of the parameters without wasting the space/memory
- Generator **yields a batch of data** and 'pauses' until the 'fit_generator' calls __next__()

Next step is resizing and normalizing the image so that all the images in a frame are of similar size and shape

Model Build:

Multiple models have been built using both architecture one having base Con3d and second having CNN+RNN combinations, I tried multiple permutation and combination by changing model parameters such as Batch Size, #Epochs, Frame Size, LR, Adding more convolution layers to arrive at the best model, please see below different models used and why I chose the CNN+RNN one

Name	Model	Accuracy	Details	# of trainable parameters
Model 1	Conv 3D	Epochs = 15 Batch Size = 30 LR = 0.01 Frame Size = 120*120 Train: 0.25 Test: 0.225 Optimizer = Adam	Base Architecture: 3 Conv3D layers, , Flatten, 3 Dense Layers (last with SoftMax 5 class), with batch normalization and max-pooling Model Underfit	87,183,749
Model 2	Conv3D	Epochs = 15 Batch Size = 30 LR = 0.001 Frame Size = 120*120 Train: 0.6 Test: 0.25 Optimizer = Adam	Base Architecture: 5 Conv3D layers, Flatten, 2 Dense Layers (last with SoftMax 5 class), with batch normalization and max-pooling Model Overfit	591,605
Model 3	Conv3D	Epochs = 20 Batch Size = 10 LR = 0.001 Frame Size = 160*160 Train: 0.9 Test: 0.85 Optimizer = Adam	Base Architecture: 5 Conv3D layers, Flatten, 2 Dense Layers (last with SoftMax 5 class), with batch normalization and max-pooling Model Fitting	27,269,253
Model 4	Conv3D	Epochs = 15 Batch Size = 15 LR = 0.001 Frame Size = 160*160 Train: 0.85 Test: 0.84 Optimizer = Adam	Base Architecture: 5 Conv3D layers, Flatten, 2 Dense Layers (last with SoftMax 5 class), with batch normalization and max-pooling Model Fitting	27,269,253

Model 5	Conv3D	Epochs = 15 Batch Size = 15 LR = 0.005 Frame Size = 100*100 Train: 0.45 Test: 0.52 Optimizer = Adam	Base Architecture: 5 Conv3D layers, Flatten, 2 Dense Layers (last with SoftMax 5 class), with batch normalization and max-pooling Model Not Fitting	10,885,253
Model 6	Conv3D	Epochs = 20 Batch Size = 12 LR = 0.001 Frame Size = 120*120 Train: 0.84 Test: 0.85 Optimizer = Adam	Base Architecture: 5 Conv3D layers, Flatten, 2 Dense Layers (last with SoftMax 5 class), with batch normalization and max-pooling Model Fitting	14,214,213
Model 7	CNN + RNN	Epochs = 20 Batch Size = 15 LR = 0.001 Frame Size = 160*160 **Train: 0.98** **Test: 0.88** Optimizer = Adam	Base Architecture: Transfer Learning + RNN: Time distributed layer taking mobilenet input with imagenet weights followed by time dist flatten layer, GRU layer, 2 Dense layers (last with SoftMax 5 class) Best Fit	8,152,773

Conclusion

- Model 7 is by far the best model when we used CNN & RNN combination with frame size of 160 and #Epochs = 20
- Model 4 & 5 can be also used if we want to use Convolution 3D as both had training and validation accuracy near to 90%