**Deep Learning Case Study- Gesture Recognition**

* Rajeshwar Rathi
* Saurabh Vyawahare

# **Problem Statement**

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 recognise five different gestures performed by the user which will help users control the TV without using a remote.

# **Objective**

Our objective in this case study is to train different models on the “train” folder, which is provided in the data set along with the “Val” folder. In this, we are going to predict the hand gestures performed by people; these actions are in the image and video format. Along with the “train” folder, we will also predict these actions in the “Val” folder. In the end, the model’s performance will be tested on the “test” set.

# **Architectures for analyzing videos**

There are two types of architectures that are suggested for analyzing videos.

1. CNN + RNN architecture
2. 3D Convolutional Neural Networks (Conv3D)

# **CNN + RNN architecture**

We feed a sequence of images (video) to the Conv2D network, and the network will extract a feature vector for each image. First, we will push this sequence of images to the CNN network, and this will extract features from each frame. Then we will give this to a sequence network like RNN. Lastly, the output of the RNN network will be a regular SoftMax which will provide us with the classification of gestures.

There are other ways to analyze videos that avoid using the RNN network because the RNN network will have a lot of parameters than the CNN network and hence will take a lot of effort to train.

Also, we have to be meticulous about hyperparameter tunning because the RNN network is a recurrent network. Due to their nature, it is hard to make them work in a suitable environment.

# **3D Convolutional Neural Networks (Conv3D)**

Instead of using Conv2D, we can use a 3D convolutional neural network (Conv3D). In this, we feed a network sequence of frames, and each frame is an image. Suppose the input we provide to a 3D Conv is a video (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.

One way to explain the advantages of Conv3D over Conv2D is to look at this through a moment of conv filter. The filter in Conv2D moves in horizontal and vertical dimensions that means it can only move in two directions. So, it does not matter whether we give a Grayscale or Color images sequence; the filter will only move in two dimensions. But in Conv3D, the additional 3rd dimension is depth; hence the convolutional filter can move in that dimension. Therefore, we get an advantage over Conv2D by using a 3D convolutional neural network.

**Data Generator**

The model building and generator are implemented within a single class to reuse the same class again for different model buildings. In the generator function, we have implemented three features:

1. Resizing the image
   * This was done to get the uniform image size to the neural network for performing apt convolutions.
2. Cropping the image
   * This was done to focus more on the gestures rather than the background, hence avoiding unnecessary noise in the images
3. Normalizing the image
   * Images were normalized to remove the possible distortions caused by shadows and lights in the image and provide uniform input to the neural network.

**Model Implementations**

The experimentation with the model building is done in the following ways with measures to counteract the deficiency of models before use.

| Experiment | Model | Accuracy | Decision + Explanation |
| --- | --- | --- | --- |
| 1 | OOM Error | - | Reduce the batch size and decrease the number of neurons |
| 2 | Conv3D | Training Acc. : 0.99  Validation Acc. : 0.81 | Model is overfitting, so to counter this, dropout layers are added to the new model |
| 3 | Conv3D | Training Acc. : 0.65  Validation Acc. : 0.52 | Validation accuracy didn’t change from 1.24219, so the early stopping criterion stopped the further training. So, now we need to decrease the learning rate. |
| 4 | Conv3D | Training Acc. : 0.76  Validation Acc. : 0.72 | By performing the above measures, overfitting is decreased but the accuracy didn’t improve. So, add more layers to the model. |
| 5 | Conv3D | Training Acc. : 0.83  Validation Acc. : 0.76 | No significant change in the accuracy of the model, therefore let’s add more dropouts to reduce the number of parameters. |
| 6 | Conv3D | Training Acc. : 0.84  Validation Acc. : 0.69 | Training accuracy is still the same though the validation accuracy is significantly dropped, let’s try to reduce the parameters to avoid overfitting |
| 7 | Conv3D | Training Acc. : 0.84  Validation Acc. : 0.74 | Reducing parameters increased the performance of the model slightly but not drastically, thus switching to CNN and LSTM architecture |
| Final Model | CNN+LSTM | Training Acc. : 0.93  Validation Acc. : 0.85 | The performance of the model is increased dramatically, this model gives the best validation accuracy with less number of parameters than previous models. |