Deep Learning Course Project- Gesture Recognition

- **Rakesh Yadav**
- **Aditya Mhamunkar**

Cohort: DS 26

Problem Statement

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.

Download Data Here - Project data - Google Drive

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.

Objective

We must make models using different functionalities that Kera's provides. Remember to use Conv3D and MaxPooling3D and not Conv2D and Maxpooling2D for a 3D convolution model. You would want to use Time Distributed while building a Conv2D + RNN model. Also remember that the last layer is the SoftMax. Design the network in such a way that the model can give good accuracy on the least number of parameters so that it can fit in the memory of the webcam.

Types Of architecture Used

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 *100 x 100 x 3*, for example, the video becomes a 4D tensor of shape *100 x 100 x 3 x 30* which can be written as *(100 x 100 x 30) x 3* where *3* is the number of channels. Hence, deriving the analogy from 2D convolutions where a 2D kernel/filter (a square filter) is represented as *(f x f) x c* where *f* is filter size and *c* is the number of channels, a 3D kernel/filter (a *'cubic'* filter) is represented as *(f x f x f) x c* (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 *(100 x 100 x 30)* tensor

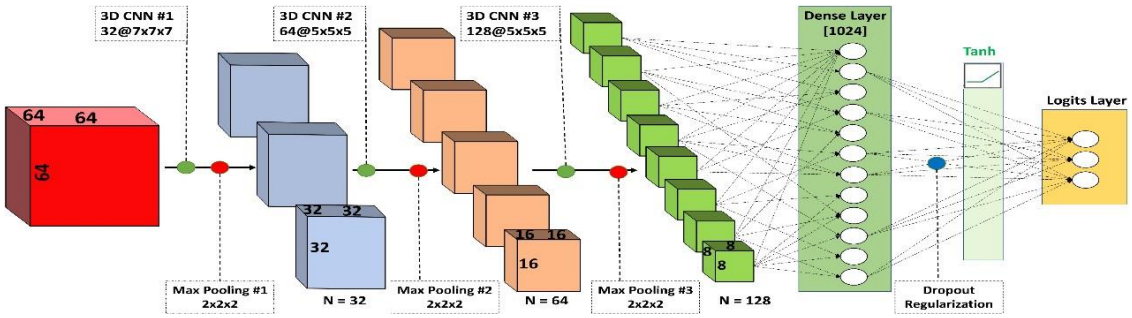

Figure 1: A simple example of working of a 3D-CNN

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).

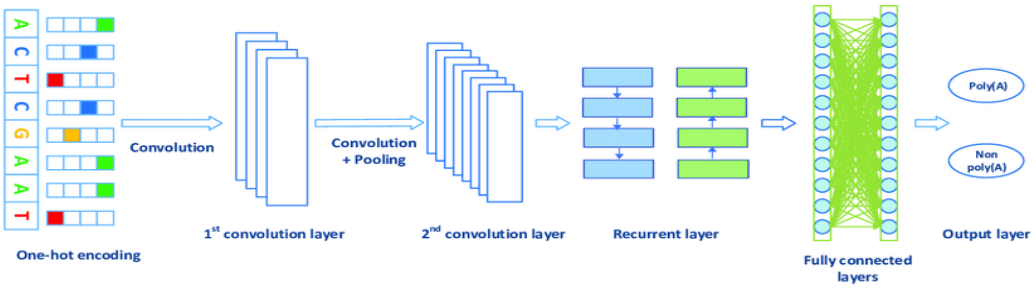

Figure 2: A simple representation of an ensembled CNN+LSTM Architecture

Transfer Learning

In transfer learning, the knowledge of an already trained machine learning model is applied to a different but related problem. For example, if you trained a simple classifier to predict whether an image contains a backpack, you could use the knowledge that the model gained during its training to recognize other objects like sunglasses.
With transfer learning, we basically try to exploit what has been learned in one task to improve generalization in another. We transfer the weights that a network has learned at "task A" to a new "task B."

Few Popular Transfer Learning architectures

- AlexNet
- VGGNet
- GoogleNet
- ResNet
- ENet

Data Generator

This is one of the most important part 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 should be performed

Write Up

Experiment No.	Model Name	Result	Decision + Explanation
1	**Conv3D(Model-1)**	TRN-Accuracy:80% VAL-Accuracy:76%	Tested with batch **size-20**, on **18 frames** using total **10 epochs**, input **image size was 100*100,** found that model is slightly overfit with some decent accuracy not so bad but not good as well. Let's try to increase the batch size and run it on 10+epoch more
2	**Conv3D(Model-2)**	TRN-Accuracy:88% VAL-Accuracy:85%	**Awesome!!! now both the accuracy (Trg+Val) is optimal, model is performing good** now but if we try to **add more epoch** runs and increase the batch size and parameters do, we get more accurate results let's try

3	**Conv3D(Model-3)**	TRN-Accuracy:48% VAL-Accuracy:56%	This time we have tried something new, and we have reduced the **kernel size to (2,2,2)** and used activation **"Relu"** with optimizer **"Adam"** but it seems Not a good idea performance reduce drastically also it took computationally more time with poor results lets stick up with the previous model and **switch to CNN+RNN**
4	**Cnv2d+RNN-(Model 1)**	TRN-Accuracy:97% VAL-Accuracy:54%	First let's start with **15 frames** image size **120*120, batch size** let's take **32 and will run it on 30 epochs**, it seems model is **overfitting** train accuracy is high but accuracy on validation data is not that effective let's try inc. learning parameters
5	**Cnv2d+RNN-(Model 2)**	TRN-Accuracy:96% VAL-Accuracy:62%	Let's see what we got, this time we can see that **overfitting** is still there also we have not used dropouts in previous **run to reduce overfitting lets add dropouts** now
6	**Cnv2d+RNN-(Model 3)**	TRN-Accuracy:20% VAL-Accuracy:28%	As we can see this idea also didn't work, we are end up with **not good results** its combination of CNN+RNN requires **more computational power and storage capacity** which we are **currently lacking** not to worry let's **switch to another architectures** this time will take help of **"Transfer Learning"** let's see what we got
7	**TL_VGG16_GRU**	TRN-Accuracy:31% VAL-Accuracy:43%	This time we have used the popular **CNN ImageNet architecture "VGG16"** and at the **dense layer we are using RNN** its seems combination both the architecture are showing good results but it seems it **need more parameters** to run on and also it will take **long run time which will take more time** and write now we can do this as we are **running out of balance and GPU** runs and more computational power :(so we decided to **stop our experiments** here only :)
8	**TL_EfficientNetB0_GRU**	**TRN-Accuracy:99%** **VAL-Accuracy:98%**	Hurray! we are end up with some great results you can see how choosing a correct architecture will lead you to some great results, **EfficientNetB0 is doing a great job overall. Time Consumption** is also **low**, **low** run of **epochs**, less **GPU consumption** and awesome results 😊

After doing all the experiments, we finalized **Experiment no. 8– with Combination of Transfer Learning Architecture (EfficientNetB0) & GRU,** which performed well.

Reason:

- (**Training Accuracy: 99%, Validation Accuracy: 98%)**
- Number of Trainable Parameters (4,266,305) and non -trainable Params (42,023) in just 20 epochs according to other models'

Further suggestions for improvement:

- **Deeper Understanding of Data:** The video clips were recorded in different backgrounds, lightings, persons and different cameras where used. Further exploration on the available images could give some more information about them and bring more diversity in the dataset. This added information can be exploited in favour inside the generator function adding more stability and accuracy to model.

- **Tuning hyperparameters:** Experimenting with other combinations of hyperparameters like, activation functions (*ReLU, Leaky ReLU, mish, tanh, sigmoid*), other optimizers like *Adagrad()* and *Adadelta()* can further help develop better and more accurate models. Experimenting with other combinations of hyperparameters like the *filter size, paddings, stride_length, batch_normalization, dropouts* etc. can further help improve performance.

- **Using Transfer Learning**: Using a pre-trained *ResNet50/ResNet152/Inception V3* to identify the initial feature vectors and passing them further to a *RNN* for sequence information before finally passing it to a SoftMax layer for classification of gestures. (This was attempted but other pre-trained models couldn't be tested due to lack of time and disk space in the nimblebox.ai platform.)