**Deep Learning Course Project – Gesture Recognition**

*By:*

* *Anuraag Godika*
* *Nitesh Maan*
* *Amar Nandan*
* *Akshay Rohankar*

**Problem Statement**

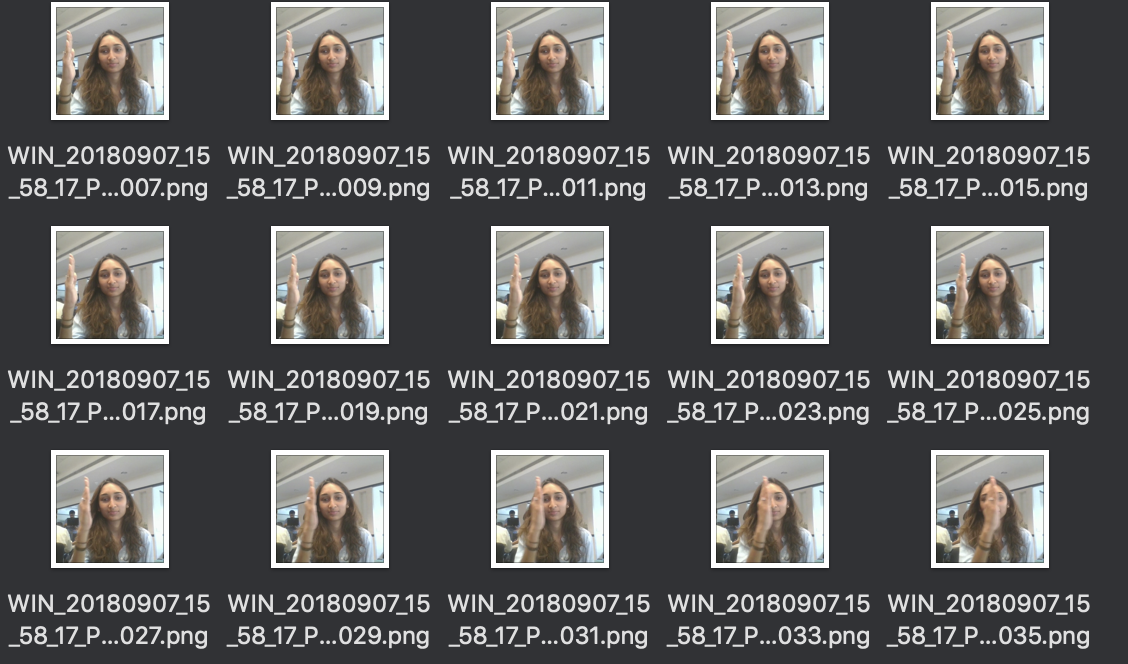
As a data scientist at a home electronics company which manufactures state of the art smart televisions. We want to develop a cool feature in the smart-TV that can recognise five different gestures performed by the user which will help users control the TV without using a remote.

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 - similar to what the smart TV will use.



**Objective**

Our task is to train different model 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.

**Model Architectures**

For analyzing videos using neural networks, **two types of architectures**are used commonly. One is the standard **CNN + RNN architecture** in which you pass the images of a video through a CNN which extracts a feature vector for each image, and then pass the sequence of these feature vectors through an RNN. This is something you are already familiar with (in theory).

The other popular architecture used to process videos is a natural extension of CNNs - a **3D convolutional network**. In this project, you will try both these architectures.

1. **3D Convolutional Network, or 3D Conv**

3D convolutions are a natural extension to the 2D convolutions. Just like in 2D conv, we move the filter in two directions (x and y), in 3D conv, we 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 120x120x3, for example, the video becomes a 4-D tensor of shape 120x120x3x30 which can be written as (120x120x30)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 (120x120x30) tensor.

3DConv

Input

Dense

Output

1. **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).

Dense

Output

RNN

2DConv

Input

**Data Generator**

This is one of the most important part of the code. In the generator, We are going to preprocess the images as we have images of 2 different dimensions as well as create a batch of video frames. We did experiment with img\_idx, y,z and normalization such that we get a high accuracy.

Experimented with variety of permutations and combination of the different augmentations techniques and parameters.

The generator, ***yields*** the batch output and labels along with other augmentations as listed below.

Augmentations:

* Flipping

Flip array in the left/right direction. Flip the entries in each row in the left/right direction. Columns are preserved, but appear in a different order than before.

* Translation

AffineTransform with

**Shifting Left :** transaltion\_transforms = tf.AffineTransform(translation=(rn.choice(range(15,40)), 0))

**Shifting Right :** transaltion\_transforms = tf.AffineTransform(translation=(rn.choice(range(-40,-15)), 0))

**Shifting Up :** transaltion\_transforms = tf.AffineTransform(translation=(0, rn.choice(range(-40,-15))))

**Shifting Down :** transaltion\_transforms = tf.AffineTransform(translation=(0, rn.choice(range(15, 40))))

**Zoom in and Zoom Out :** scale\_transforms = tf.AffineTransform(scale=(zoom, zoom))

* Noise

Adding Random Noise may help with lighting distortions and make the model more robust in general.

* Contrast and Brightness

Rescaling the intensities of the contrast and Brightness of the Images for adding lighting variety to the images.

* Jitters

Adding Jitters further to make the model more robust in general.

* Rotations

Rotate : rot(image, rn.choice(range(5,20)))

anti-rotate images : rot(image, rn.choice(range(-20,-5))),

**#Models**

**Model – 1: Conv3D**

**Training Parameters:**

num\_classes = 5

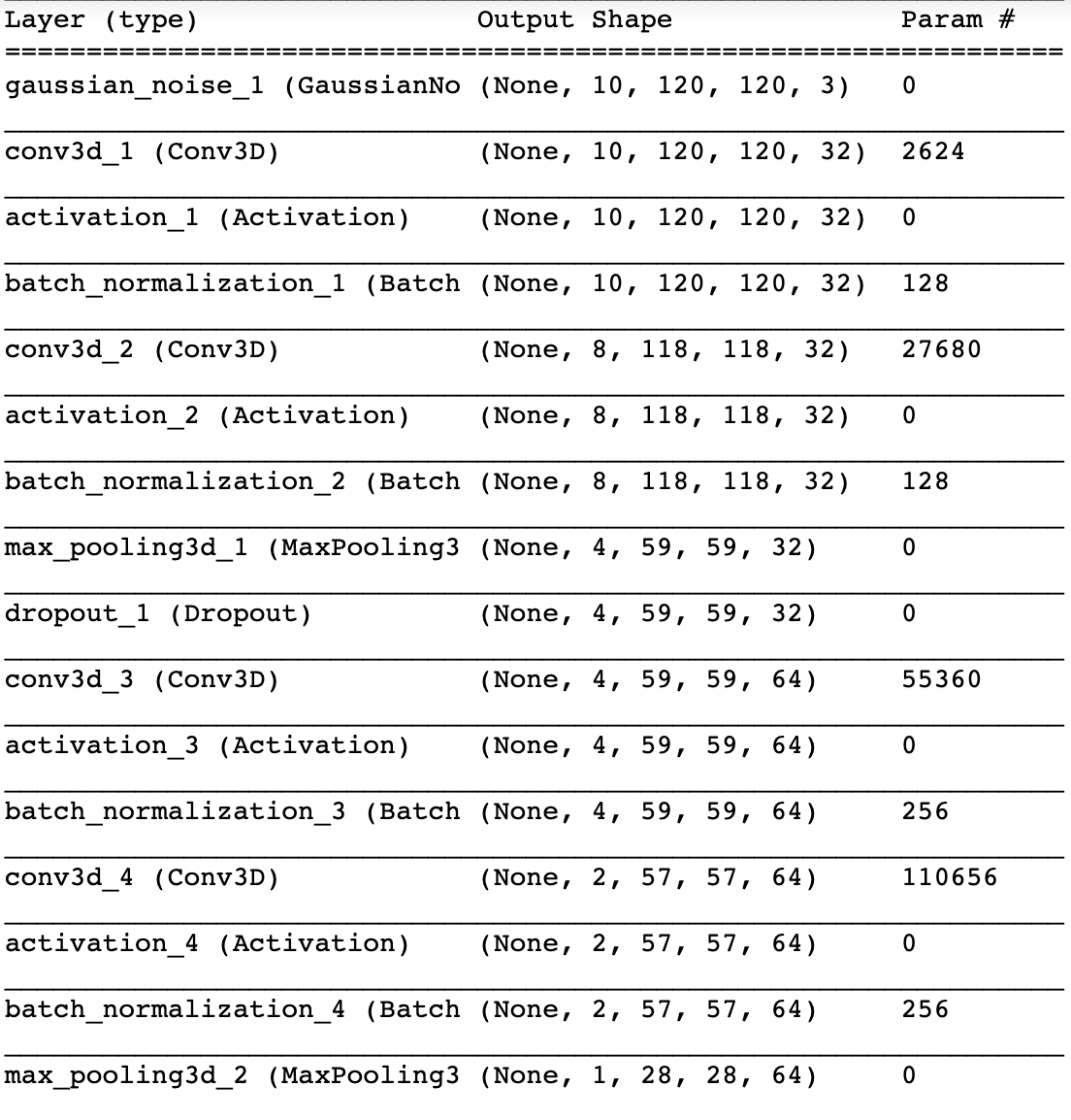
num\_epochs = 30

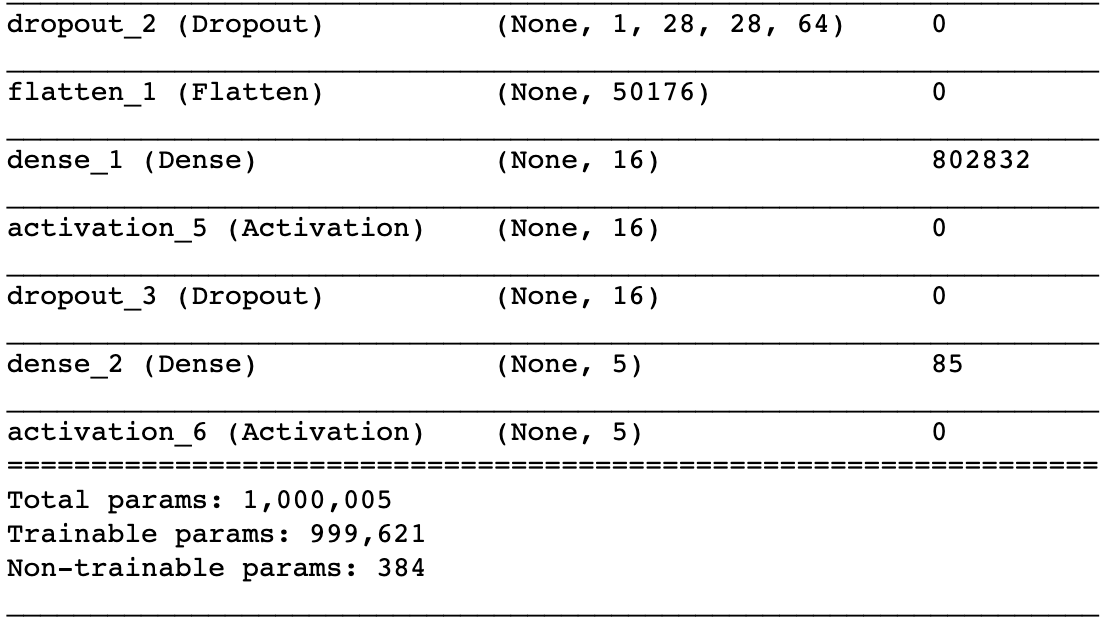
loss='categorical\_crossentropy'

optimizer = SGD (lr=0.01, clipnorm=1.)

steps\_per\_epoch = steps\_per\_epoch \* 6 🡪 #Because we are augmenting each image

LR = ReduceLROnPlateau(monitor='val\_loss', factor=0.2,patience=3, min\_lr=0.001)

****



Total params: 1,000,005

Trainable params: 999,621

Best Model Accuracy:

model\_init\_2019-06-2320\_14\_42.038750/model-00030-1.57303-0.37557-1.29646-0.52000.h5

- categorical\_accuracy: 0.3744

- val\_loss: 1.2965

- val\_categorical\_accuracy: 0.5200

**Model – 2: model\_LSTM**

**Training Parameters:**

num\_classes = 5

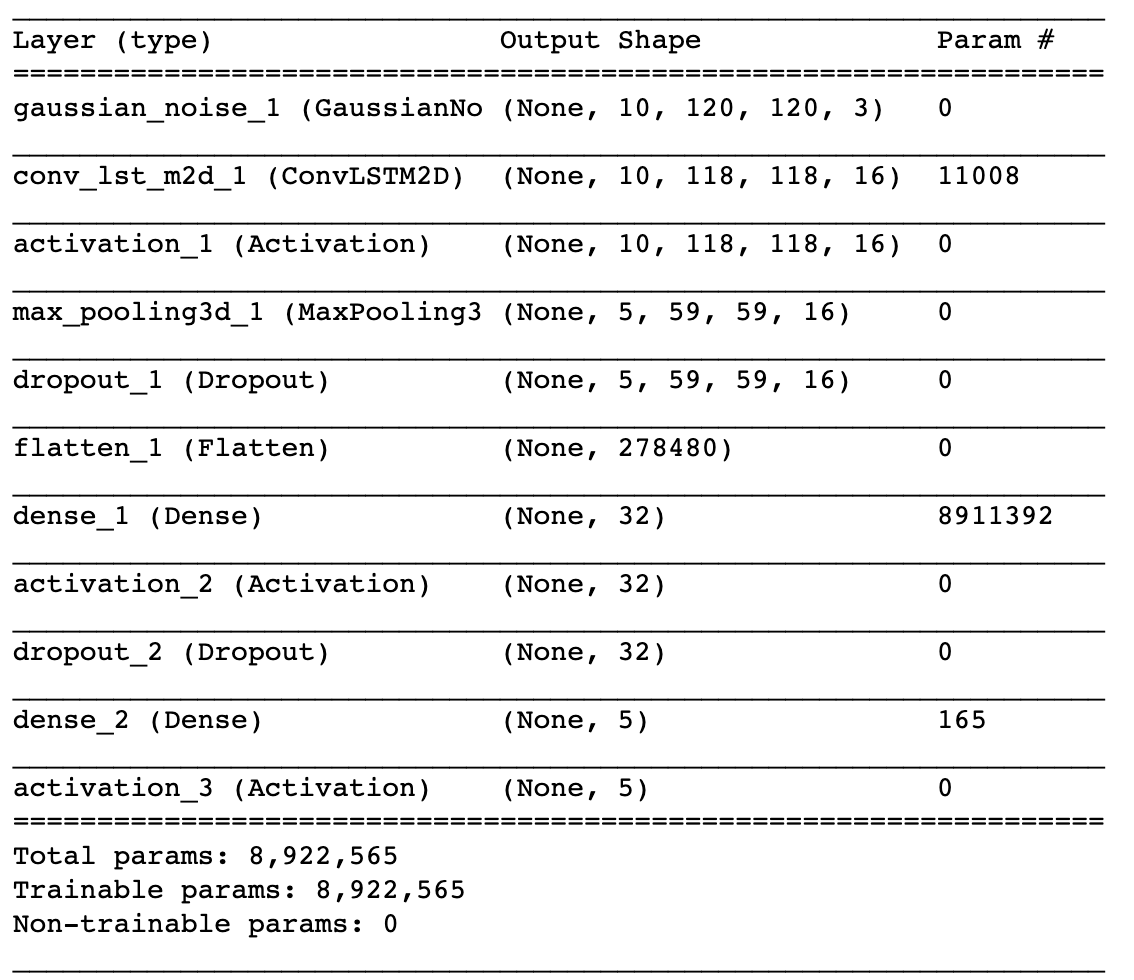
num\_epochs = 30

loss='categorical\_crossentropy'

optimizer = SGD (lr=0.01, clipnorm=1.)

steps\_per\_epoch = steps\_per\_epoch \* 6 🡪 #Because we are augmenting each image

LR = ReduceLROnPlateau(monitor='val\_loss',factor=0.2,patience=3, min\_lr=0.001)



Total params: 8,922,565

Trainable params: 8,922,565

Best Model Accuracy:

model\_init\_2019-06-2310\_06\_50.311870/model-00030-1.61525-0.55153-1.46911-0.60000.h5

- categorical\_accuracy: 0.5502

- val\_loss: 1.4691

- val\_categorical\_accuracy: 0.6000

**Model – 3: model\_LSTM**

**Training Parameters:**

num\_classes = 5

num\_epochs = 30

loss='categorical\_crossentropy'

optimizer = SGD (lr=0.01, clipnorm=1.)

steps\_per\_epoch = steps\_per\_epoch \* 6 🡪 #Because we are augmenting each image

LR = ReduceLROnPlateau(monitor='val\_loss',factor=0.2,patience=3, min\_lr=0.001)

Best Model Accuracy:

This is a sample write-up. The write-up need not be in tabular form.

It doesn’t state that ConvLSTM will give you better results than Conv3D. The explanation should be as detailed as possible so that the logic behind the decision is conveyed. Also, there are a lot of things you can experiment with in the generator function and elsewhere. Please do not forget to specify the exact metric values, here Accuracy which drives your decision.

You can draw inspiration from the concepts taught in the Industry demo in CNNs to experiment with the data and different architectures.

|  |  |  |  |
| --- | --- | --- | --- |
| **Experiment Number** | **Model** | **Result** | **Decision + Explanation** |
| **1** | **Conv3D** | **Throws Generator error** | **Crop the images correctly, try to overfit on less amount of data** |
| **2** | **Conv3D** | **Model not trainable as a lot of parameters** | **Reduce the size of the image/Reduce the number of layers** |
| **3** | **Conv3D** | **Accuracy: 0.21** | **Increase the amount of trainable data/ reduce the filter size** |
|  |  |  |  |
|  |  |  |  |
| **2** | **Conv3D** | **Accuracy: 0.32** | **Reduce Cropping** |
| **3** | **Conv3D** | **Accuracy : 0.38** | **………………** |
|  |  |  |  |
| **l-1th** | **Conv3D** | **Accuracy: 0.45** | **Try ConvLSTM as Conv3D not giving desired accuracy** |
| **lth** | **ConvLSTM** | **Accuracy: …….** | **…………..** |
|  |  |  |  |
| **Final Model** | **……………….** | **………….** | **…………………** |