**Considerations:**

**Our model should be able to identify gestures which correspond to specific commands in a smart TV. There are mentioned below:**

* 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

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.

Each video is a sequence of 30 frames (or images). All images in a particular video subfolder have the same dimensions but different videos may have different dimensions. Specifically, videos have two types of dimensions - either 360x360 or 120x160 (depending on the webcam used to record the videos).

Hence, we need to do some pre-processing to standardise the videos. We have done resizing of the frames to a shape of 120 X 120.

**Conv3D Architecture**

**There were 2 different sized images Input\_shape** (30, 120, 120, 3) has 4 dimensions. A 3D image is a 4-dimensional data where the fourth dimension represents the number of **colour channels.**

***Keras Code***

# Input

input\_shape=(img\_frames,img\_height,img\_width,img\_channel)

# Input Layer

model = Sequential()

model.add(Conv3D(img\_filter[0], kernel\_size=(5, 5, 5), padding='same', input\_shape=input\_shape))

model.add(Activation('relu'))

model.add(BatchNormalization())

model.add(MaxPooling3D(pool\_size=(2, 2, 2)))

# Layer 2

model.add(Conv3D(img\_filter[1], kernel\_size=(3, 3, 3), padding='same'))

model.add(Activation('relu'))

model.add(BatchNormalization())

model.add(MaxPooling3D(pool\_size=(2, 2, 2)))

# Layer 3

model.add(Conv3D(img\_filter[2], kernel\_size=(1, 3, 3), padding='same'))

model.add(Activation('relu'))

model.add(BatchNormalization())

model.add(MaxPooling3D(pool\_size=(2, 2, 2)))

# Layer 4

model.add(Conv3D(img\_filter[3], kernel\_size=(1, 3, 3), padding='same'))

model.add(Activation('relu'))

model.add(BatchNormalization())

model.add(MaxPooling3D(pool\_size=(2, 2, 2)))

#Flatten the layers

model.add(Flatten())

#Fully Connected Layer

model.add(Dense(img\_dense[0],activation='relu'))

model.add(BatchNormalization())

model.add(Dropout(dropout[0]))

model.add(Dense(img\_dense[1],activation='relu'))

model.add(BatchNormalization())

model.add(Dropout(dropout[1]))

#softmax layer

model.add(Dense(nb\_classes,activation='softmax'))

Argument **kernel\_size** (5,5,5) represents (height, width, depth) of the kernel, and 4th dimension of the kernel will be the same as the colour channel.

The details of each experiment is presented in the table below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Experiment Number** | **Model Architecture** | **Result** | **Parameters Used** | **Decision and Explanation** |
| **1** | **Conv2D + Time Distributed** | **Throws error** | **Specify the Input Shape of Time Distributed layers** |  |
| **2** | **Conv2D + Time Distributed** | **Accuracy: 45%** | **With 3\*3 Filter Size** | **The number of epochs were limited to 10 and batch size of 10 to check on accuracy. Accuracy was 45%, not too good.** |
| **Batch Size = 10** |
| **Epoch = 10** |
| **Optimizer = Adam** |
| **Learning Rate = 0.0005000000237487257** |
| **3** | **Conv2D + Time Distributed** | **Accuracy: 69%** | **With 3\*3 Filter Size** | **Increased the number of epochs to check on accuracy. Accuracy improved.** |
| **Batch Size = 10** |
| **Epoch = 30** |
| **Optimizer = Adam** |
| **Learning Rate = 4.882812731921149e-07** |
| **4** | **Conv3D + RNN (Without Data Augmentation)** | **Accuracy: 63%** | **With 1st Filter = 5\*5 and second Filter = 3\*3** | **Used Conv3D by some image pre-processing: 1. Resizing - 160 X 160 2. Normalization  The number of epochs were limited to 10 and batch size of 24 to check on accuracy. Accuracy was 63%, not too good.** |
| **Batch Size = 24** |
| **Epoch = 10** |
| **Optimizer = Adam** |
| **Image Size = 160 X 160** |
| **Trainable Parameters = 1,702,821** |
| **Learning Rate = .001** |
| **5** | **Conv3D + RNN (Without Data Augmentation)** | **Accuracy: 69%** | **With 1st Filter = 5\*5 and second Filter = 3\*3** | **Used Conv3D by some image pre-processing: 1. Resizing - 120 X 120 2. Normalization  The number of epochs were limited to 10 and batch size of 24 to check on accuracy. Accuracy increased to 69% on reducing the image size.** |
| **Batch Size = 24** |
| **Epoch = 10** |
| **Optimizer = Adam** |
| **Image Size = 120 X 120** |
| **Trainable Parameters = 1,752,901** |
| **Learning Rate = 0.005** |
| **6** | **Conv3D + RNN (Without Data Augmentation)** | **Accuracy: 82%** | **With 1st Filter = 5\*5 and second Filter = 3\*3** | **Run the model wfor 30 epcohs and accuracy imprved to 82%. But there was a difference of 14% with the training accuracy. Model seemed to be overfitting** |
| **Batch Size = 24** |
| **Epoch = 30** |
| **Optimizer = Adam** |
| **Image Size = 120 X 120** |
| **Trainable Parameters = 1,752,901** |
| **Learning Rate = 0.0006** |
| **7** | **Conv3D + RNN (Without Data Augmentation)** | **Accuracy: 87%** | **With 1st Filter = 5\*5 and second Filter = 3\*3** | **Changed the batch size to 32 and accuracy increased to 87%.** |
| **Batch Size = 32** |
| **Epoch = 30** |
| **Optimizer = Adam** |
| **Image Size = 120 X 120** |
| **Trainable Parameters = 1,752,901** |
| **Learning Rate = 1.220703097715159e-06** |
| **8** | **Conv3D + RNN (Without Data Augmentation)** | **Accuracy: 74%** | **With 1st Filter = 3\*3 and second Filter = 3\*3** | **Wanted to check with a different filter size. But accuracy decreased.** |
| **Batch Size = 32** |
| **Epoch = 30** |
| **Optimizer = Adam** |
| **Image Size = 120 X 120** |
| **Trainable Parameters = 1,932,517** |
| **Learning Rate = 0.005** |
| **9** | **Conv3D + RNN (Without Data Augmentation)** | **Accuracy: 61%** | **With 1st Filter = 3\*3 and second Filter = 3\*3** | **Changed the strides but accuracy further reduced** |
| **Strides = 2\*2** |
| **Batch Size = 32** |
| **Epoch = 10** |
| **Optimizer = Adam** |
| **Image Size = 120 X 120** |
| **Trainable Parameters = 1,752,901** |
| **Learning Rate = 0.005** |
| **10** | **Conv3D + RNN (Without Data Augmentation)** | **Accuracy: 68%** | **With 1st Filter = 5\*5 and second Filter = 3\*3** | **Tried with a different Optimizer called Adadelta but accuracy decreased.** |
| **Batch Size = 32** |
| **Epoch = 30** |
| **Optimizer = Adadelta** |
| **Image Size = 120 X 120** |
| **Trainable Parameters = 1,752,901** |
| **Learning Rate = 0.0005000000237487257** |
| **11** | **Conv3D + RNN (Without Data Augmentation)** | **Accuracy: 82%** | **With 1st Filter = 5\*5 and second Filter = 3\*3** | **Accuracy increased to 82% when the batch size was reduced to 24 with Addelta optimizer. But it was still less than the one achieved using Adam. Hece decided to continue using Adam** |
| **Batch Size = 24** |
| **Epoch = 30** |
| **Optimizer = Adadelta** |
| **Image Size = 120 X 120** |
| **Trainable Parameters = 1,752,901** |
| **Learning Rate = 0.008** |
| **12** | **Conv3D + RNN (With Data Augmentation)** | **Accuracy: 68%** | **With 1st Filter = 5\*5 and second Filter = 3\*3** | **Tried some data augmentation by rotating the images on the fly with an angle of 30 degrees. Limited the epoch to 10 to check on accuracy and overall fit.** |
| **Batch Size = 32** |
| **Epoch = 10** |
| **Optimizer = Adam** |
| **Image Size = 120 X 120** |
| **Trainable Parameters = 1,752,901** |
| **Learning Rate = 0.001** |
| **13** | **Conv3D + RNN (With Data Augmentation)** | **Accuracy: 59%** | **With 1st Filter = 5\*5 and second Filter = 3\*3** | **Reduced the image size to check whether we can further improve on accuracy but accuracy decresed** |
| **Batch Size = 32** |
| **Epoch = 10** |
| **Optimizer = Adam** |
| **Image Size = 100 X 100** |
| **Trainable Parameters = 1,326,917** |
| **Learning Rate = 0.0005** |
| **14** | **Conv3D + RNN (With Data Augmentation)** | **Accuracy: 82%** | **With 1st Filter = 5\*5 and second Filter = 3\*3** | **Run with image size 120 X 120 for 30 epochs to see the overall improvement with data augmentation. The accuracy achieved was 82% which was less than the accuracy of 87% achieved without data augmentation. So decided not to use any data augmentation technique.** |
| **Batch Size = 32** |
| **Epoch = 30** |
| **Optimizer = Adam** |
| **Image Size = 120 X 120** |
| **Trainable Parameters = 1,752,901** |
| **Learning Rate = 3.125000148429535e-05** |
| **15** | **Conv3D + RNN (Without Data Augmentation)** | **Accuracy: 72%** | **With 1st Filter = 5\*5 and second Filter = 3\*3** | **Increased the number of network layers but it seems to have overfitted the model. So decided to continue with the original model itself.** |
| **Batch Size = 32** |
| **Epoch = 20** |
| **Optimizer = Adam** |
| **Image Size = 120 X 120** |
| **Trainable Parameters = 1,997,589** |
| **Learning Rate = 0.0005** |
| **Number of Network Layers = Doubled** |
| **16** | **Conv3D + RNN (Without Data Augmentation)** | **Accuracy: 80 %** | **With 1st Filter = 5\*5 and second Filter = 3\*3** | **Reduced the number of trainable parameters from dense layer and accuracy increased but it was still overfitting. So reverted back to the original design of the dense layers** |
| **Batch Size = 32** |
| **Epoch = 30** |
| **Optimizer = Adam** |
| **Learning Rate = 1.5625000742147677e-05** |
| **Trainable Parameters = 933,317** |
| **Image Size = 120 X 120** |
| **Number of Network Layers = Single** |
| **Number of Parameters reduced** |
| **17** | **CNN + LSTM (Without Data Augmentation)** | **Accuracy: 73 %** | **With 1st Filter = 5\*5 and second Filter = 3\*3** | **Tried to use LSTM but accuracy reduced and the overfitting problem persisted** |
| **Batch Size = 32** |
| **Epoch = 30** |
| **Optimizer = Adam** |
| **Learning Rate = 0.000125** |
| **Trainable Parameters = 789,509** |
| **Image Size = 120 X 120** |
| **Number of Network Layers = Single** |
| **18** | **Transfer Learning with Mobilenet** | **Accuracy: 73 %** | **With 1st Filter = 5\*5 and second Filter = 3\*3** | **Tried using a transfer learning mechanism using mobilenet but it did not benefit us.** |
| **Batch Size = 32** |
| **Epoch = 30** |
| **Optimizer = Adam** |
| **Learning Rate = 0.000125** |
| **Trainable Parameters = 285,317** |
| **Image Size = 120 X 120** |
| **Number of Network Layers = Single** |

References:

<https://towardsdatascience.com/complete-image-augmentation-in-opencv-31a6b02694f5>

<https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/>