Neural Network Project: Gesture Recognition

Kamran Shaikh

Kamil Shaikh

**Problem Statement**

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
* Each video is a sequence of 30 frames (or images).

**Understanding the Data**

The data is in a zip file. The zip file contains a 'train' and a 'val' folder with two CSV files for the two folders. These folders are in turn divided into subfolders where each subfolder represents a video of a particular gesture. Each subfolder, i.e. a video, contains 30 frames (or images). Note that 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, you will need to do some pre-processing to standardise the videos.

Each row of the CSV file represents one video and contains three main pieces of information - the name of the subfolder containing the 30 images of the video, the name of the gesture and the numeric label (between 0-4) of the video.

**Objective**

The task is to train a model on the 'train' folder which performs well on the 'val' folder as well (as usually done in ML projects). We have withheld the test folder for evaluation purposes - your final model's performance will be tested on the 'test' set.

**Models**

**Approach1: CNN (Conv3D)**

Since we are classifying the images from the videos and CNNs are incapable to process the sequential data and time dependency between the images.

Hence RNNs are ideal for such a situation.

But We will use CNN to get some Acceptable accuracy but as we know CNNs incapability ie Regular CNNs takes spatial data as input and does not have any immediate relationship with any other inputs.

However, for our Project we must resolve the movement of hands across some/all the different Frames of one frame of one sequence and therefore we will use CNN 3D model where those frames of sequence will be fed at a single instance.

In 3D Matrix the 3rd dimension will be the temporal dimension. This 3rd dimension will contain the Delta between each Frame and CNNs will be able to capture this.

**Approach2: CNN + RNN**

The Simple CNN will be fed with sequence of images, and it will extract feature vector of each image and then the sequence of these feature vectors are passed to RNN. The output of RNN is Softmax.

LSTM, GRU and Simple RNN would be used for comparison.

**Generator Function**

We have to set up the data ingestion pipeline. In most deep learning projects, we need to feed data to the model in batches. This is done using the concept of generators.

Creating data generators is probably the most important part of building a training pipeline. Although libraries such as Keras provide built in generator functionalities, they are often restricted in scope, and we have to write your own generators from scratch. For example, in this problem, we need to feed batches of videos, not images.

For this assignment we will use custom Generator which will work with yield and this will help in creating a batch of any kind of data

This Generator takes number of images as per batch size selected and perform the experiment with below parameters

number of images to be taken per video/sequence

* cropping the images
* resizing the images
* normalizing the images

The Generator has an infinite while loop. So it’s always ready to yield a batch once next() is called. The First call to Yield is made during Initialization. We have selected any random image to see the cropping is working fine or not.

A person with his hand on his chin

Description automatically generated with low confidence

A person with his hand on his chin

Description automatically generated with low confidence

**Model Building and Training:**

We have a function which accepts some parameters when the function is called. Then we have build multiple models by giving different values to these Parameters and find the best fit model with highest Accuracy

* The Model (dropout, filter, Padding, type, number of layers, Batch Normalization)
* Image width and height
* Batch Size
* Number of epochs
* Number of images in a sequence to be considered for training

We have trained couple of models on the basis of the specified parameters to arrive at best Accuracy.

We have noticed to choose correct Batch size as whenever the Batch size is high the Kernel dies automatically taking up most of the memory.

Graphical user interface, text, application

Description automatically generated

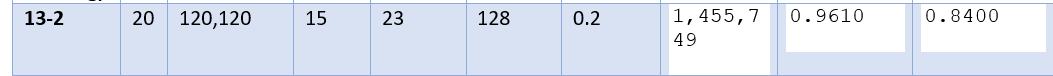
**CNN: Conv3D: Model Performances**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model No. | Batch Size | Image Dimensions  (h,w) | Epoch | No. of Images | Dense Neurons | Dropout | Total  Params | Categorical Accuracy | Validation Accuracy |
| 1-1 | 40 | 160,160 | 5 | 15 | 64 | 0.25 | 1,714,437 | 0.8091 | 0.1400 |
| 1-2 | 40 | 160,160 | 5 | 20 | 64 | 0.25 | 3,352,837 | 0.7743 | 0.3200 |
| 1-3 | 40 | 120,120 | 5 | 15 | 64 | 0.25 | 997,637 | 0.8735 | 0.2000 |
| 1-4 | 40 | 120,120 | 5 | 20 | 64 | 0.25 | 1,919,237 | 0.8813 | 0.1400 |
| 1-5 | 32 | 120,120 | 5 | 20 | 64 | 0.25 | 1,919,237 | 0.8633 | 0.1500 |
| 1-6 | 32 | 120,120 | 5 | 15 | 64 | 0.25 | 997,637 | 0.8216 | 0.1600 |
| 2-1 | 40 | 120,120 | 5 | 20 | 128 | 0.25 | 3,775,685 | 0.8671 | 0.2500 |
| 2-2 | 40 | 100,100 | 5 | 20 | 128 | 0.25 | 2,448,581 | 0.8557 | 0.2400 |
| 3-1(adding 1 more conv3D layer) | 40 | 160,160 | 5 | 20 | 64 | 0.25 | 1,112,645 | 0.9232 | 0.1900 |
| 3-2 | 30 | 120,120 | 5 | 30 | 64 | 0.25 | 694853 | 0.9213 | 0.2700 |
| 4-1(adding 1 more dense layer) | 40 | 120,120 | 5 | 20 | 64 | 0.25 | 699,269 | 0.7893 | 0.2400 |
| 4-2 | 25 | 120,120 | 5 | 20 | 64 | 0.25 | 699,269 | 0.7633 | 0.1600 |
| 5-1 | 30 | 120,120 | 5 | 20 | 64 | 0.5 | 699,269 | 0.4845 | 0.1500 |
| 6-1 | 30 | 120,120 | 5 | 20 | 64 | 0.2 | 699,269 | 0.7659 | 0.2000 |
| 7-1 | 30 | 120,120 | 5 | 20 | 128 | 0.2 | 1,113,925 | 0.7740 | 0.2400 |
| 8-1(changing filters to 2,2,2) | 30 | 120,120 | 5 | 20 | 64 | 0.2 | 494,069 | 0.8232 | 0.1600 |
| 9-1(changing the optimizer) | 40 | 120,120 | 5 | 20 | 64 | 0.2 | 494,069 | 0.8201 | 0.2200 |
| 10-1 | 40 | 120,120 | 5 | 20 | 256 | 0.2 | 1,762,613 | 0.9627 | 0.2400 |
| 11-1 | 20 | 120,120 | 5 | 25 | 128 | 0.25 | 5,601,861 | 0.8339 | 0.2900 |
| 12-1(Removing Batch Normalization) | 20 | 120,120 | 5 | 20 | 128 | 0.2 | 3,757,701 | 0.7074 | 0.7000 |
| 12-2 | 20 | 140,140 | 5 | 20 | 128 | 0.2 | 4,806,277 | 0.7305 | 0.6500 |
| 12-3 | 20 | 120-120 | 5 | 25 | 128 | 0.2 | 5600901 | 0.6655 | 0.5800 |
| 12-4 | 15 | 120-120 | 5 | 25 | 128 | 0.2 | 5,600,901 | 0.6917 | 0.6300 |
| 13-1(Remove Padding) | 20 | 120-120 | 15 | 23 | 128 | 0.5 | 1,455,749 | 0.9313 | 0.7200 |
| 13-2 | 20 | 120,120 | 15 | 23 | 128 | 0.2 | 1,455,749 | 0.9610 | 0.8400 |
| 13-3 | 18 | 120,120 | 15 | 23 | 128 | 0.5 | 1,455,749 | 0.9242 | 0.7200 |
| 13-4 | 20 | 120,120 | 20 | 22 | 128 | 0.5 | 1,455,749 | 0.9232 | 0.6600 |

**CNN + RNN – Model Performance**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model No. | Batch Size | Image Dimensions  (h,w) | Epoch | No. of Images | Dense Neurons | Dropout | Total  Params | Categorical Accuracy | Validation Accuracy |
| 1-1 LSTM | 50 | 120,120 | 5 | 15 | 64 | 0.25 | 1,005,541 | 0.6239 | 0.2100 |
| 1-2 | 20 | 120,120 | 5 | 15 | 64 | 0.25 | 1,005,541 | 0.5890 | 0.1800 |
| 1-3 | 20 | 120-120 | 5 | 20 | 64 | 0.25 | 1,005,541 | 0.5903 | 0.2300 |
| 1-4 | 20 | 120,120 | 5 | 20 | 64 | 0.5 | 1,005,541 | 0.4475 | 0.2200 |
| 2-1 GRU | 20 | 120,120 | 5 | 20 | 64 | 0.25 | 854,117 | 0.6386 | 0.2100 |
| 2-2 | 20 | 120,120 | 5 | 24 | 64 | 0.25 | 854,117 | 0.7225 | 0.2100 |
| 3-1 GRU-Removing BN | 20 | 120,120 | 5 | 20 | 64 | 0.25 | 852,133 | 0.4382 | 0.4300 |
| 3-2 | 20 | 120,120 | 15 | 18 | 64 | 0.25 | 852,133 | 0.7254 | 0.6300 |
|  |  |  |  |  |  |  |  |  |  |

**Best Model**

****

Graphical user interface, chart, line chart

Description automatically generated

Observations

1. **Activation-Relu , dense\_neurons-64, dropout-0.25 ,Batch size-40,number of images-20,Image size:120,120**

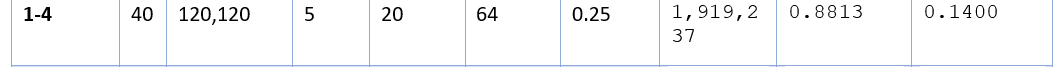
**Chart, line chart

Description automatically generated**

Clearly this model is overfitting as validation accuracy is much less than training Accuracy

The Parameters have to be tuned to see the effects and to reduce the overfitting

1. Seeing the impact of increasing the number of dense neurons.





Sill the model is overfitting. It can be seen increasing the number of dense neurons has not much impact on validation accuracy with huge increase in number of trainable parameters. The results are checked after 5 epoches. Hence, still overfitting is present even after increasing dense neurons. It might be possible that for higher epochs, accuracy starts to improve. But still, no satisfactory results.

1. Adding one more conv3D layer: We have seen improvement in accuracy with adding one more conv3D layer.



Table

Description automatically generated

1. We have observed that by removing the Batch Normalization, we have got better validation accuracy.

And the training and validation accuracy comes very close.

Graphical user interface, text, application

Description automatically generated

Chart, line chart

Description automatically generated

1. Decreasing the Dense Neuron to 0.2 increases both Accuracy and Val Accuracy.

Table, calendar

Description automatically generated

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

1. It is important to choose wisely the image resolution as it can be noticed that by increasing the image size after a certain value, number of parameters increase drastically with very less increase in validation accuracy.
2. LSTM has a greater number of trainable parameters than GRU with almost same accuracy.