

Problem Statement

To develop a 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

Each video is a sequence of 30 frames (or images).These frames would be used for training and recognizing the gestures.

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.

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). 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’ll do some pre-processing to standardize 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).

This model will further be used to train unseen test data to verify the model accuracy.

Model Architectures

**Approach1**: CNN - Conv3D

As we know, CNNs DO NOT have ability to capture Temporal Information or Sequential Information.

For that, we need RNNs that can Take Inputs in Sequence and then Resolve Time-dependent Delta between Images with the help of Recurrent Connections that Loop back into Same Layer from the Previous States.

So – for this assignment, one would normally use RNNs to start with the Modeling.

However, we will Attempt to see if CNNs can be used to some Acceptable Accuracy for this task at hand. For that, we cannot train the Models as Regular CNNs. Regular CNNs Treat each individual input as an Independent Input – that does not have any immediate Relationship with any of the other Inputs.

However, for our Project, to Recognize Gesture from a Sequence of Images, we Must Resolve the Movement of hands across some/all the different Frames of one particular Sequence, hence we will use a 3D model where those Frames of a Sequence will be Fed at a Single Instance.

In this 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 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). RNN, classically used to resolve Temporal Information, should give a Better Performance.

We will train LSTM, GRU and simple RNN models for comparisons and to get better results.

**GENERATOR FUNCTION**

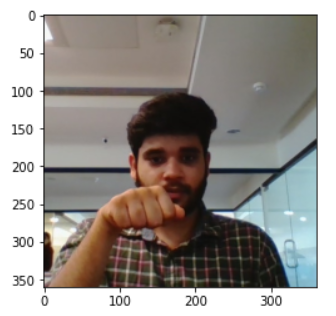
We have to set up the data ingestion pipeline. In most deep learning projects we need to feed data to the model in batches. This is done using the concept of generators.

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 our own generators from scratch. For example, in this problem, we need to feed batches of videos, not images. We’ll be building generator from the scratch.

For this assignment, we have used a Custom Generator which takes input as number of images as per batch size selected an performs below tasks.

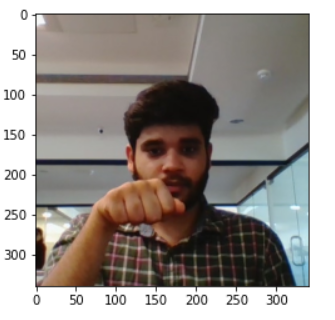
* Taking care of the Pre processing steps like cropping the images, Resizing and normalizing the images.
* Yielding images in Batches
* Handing the case where if Total Training data count is NOT perfectly divisible by a particular Batch Size.

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.



Above : **Image**

Down: **Cropped Image**



**Model Building and Training:**

A function is defined with some parameters and the value of these parameters can be passed while the function call. So, we can train multiple models using different values of parameters and can see the impact of these parameters on model accuracy.

Parameters are:

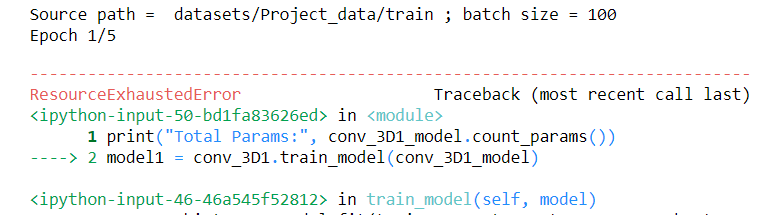
* The Model (dropout,filter,Padding,type,number of layers, Batch Normalization)
* Image Resolution (width, height)
* Batch Size
* Number of epochs
* Number of Image in a sequence to be considered for training

We have trained couple of models to arrive at the best accuracy for training and validation data. The details of each models have been described below in tabular format and then we’ll summarize the results for each type of models (Conv3D and CNN+RNN)

This Plot function is also Called after Training Each Model by passing as parameter the Return from the model.fit command – so both these Graphs are Drawn after Each Model Training and we can have a better idea on how the Training Performed.

Also, we have observed that it is very important to choose the correct Batch size as otherwise if Batch size is too large, it would result in Resource Exhaustion error.

**Model1**: Batch Size: 100: It has resulted in Resource Exhaustion as shown below.



\*\* X-Y, X=model choosen, Y= function call to that model with specific parameters

**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 | 75 | (160,160) | 5 | 15 | 64 | 0.25 | 1,714,437 | 0.1961 | 0.1700 |
| 1-2 | 75 | (120,120) | 5 | 15 | 64 | 0.25 | 997,637 | 0.8507 | 0.2000 |
| 1-3 | 75 | (100,100) | 5 | 15 | 64 | 0.25 | 665,861 | 0.1991 | 0.2100 |
| 1-4 | 75 | (120,120) | 5 | 20 | 64 | 0.25 | 1,919,237 | 0.8793 | 0.2000 |
| 1-5 | 50 | (120,120) | 5 | 20 | 64 | 0.25 | 1,919,237 | 0.8959 | 0.2600 |
| 1-6 | 50 | (120,120) | 5 | 15 | 64 | 0.25 | 997,637 | 0.8854 | 0.1600 |
| 1-7 | 40 | (120,120) | 5 | 20 | 64 | 0.25 | 1,919,237 | 0.9397 | 0.1100 |
| 1-8 | 40 | (120,120) | 5 | 18 | 64 | 0.25 | 1,919,237 | 0.8190 | 0.2200 |
| 1-9 | 40 | (120,120) | 5 | 22 | 64 | 0.25 | 1,919,237 | 0.8914 | 0.2000 |
| 1-10 | 40 | (120,120) | 5 | 15 | 64 | 0.25 | 997,637 | 0.7919 | 0.1600 |
| 2-1 | 50 | (120,120) | 5 | 15 | 128 | 0.25 | 1,932,485 | 0.8522 | 0.1700 |
| 2-2 | 50 | (100,100) | 5 | 15 | 128 | 0.25 | 1,268,933 | 0.9035 | 0.2200 |
| 2-3 | 50 | (100,100) | 5 | 20 | 128 | 0.25 | 2,448,581 | 0.9502 | 0.1600 |
| 2-4 | 50 | (120,120) | 5 | 20 | 128 | 0.25 | 3,775,685 | 0.9563 | 0.1600 |
| 3-1 (adding 1 more conv3D layer) | 40 | (160,160) | 5 | 20 | 64 | 0.25 | 1,112,645 | 0.9879 | 0.2300 |
| 3-2 | 40 | (120,120) | 5 | 20 | 64 | 0.25 | 694,853 | 0.9789 | 0.1900 |
| 3-3 | 30 | (120,120) | 5 | 20 | 64 | 0.25 | 694,853 | 0.9759 | 0.2800 |
| 4-1(adding 1 more dense layer) | 40 | (120,120) | 5 | 20 | 64 | 0.25 | 699,269 | 0.8100 | 0.2400 |
| 4-2 | 30 | (120,120) | 5 | 20 | 64 | 0.25 | 699,269 | 0.8281 | 0.3400 |
| 4-3 | 25 | (120,120) | 5 | 20 | 64 | 0.25 | 699,269 | 0.7964 | 0.2700 |
| 5-1 | 30 | (120,120) | 5 | 20 | 64 | 0.5 | 699,269 | 0.5460 | 0.1800 |
| 6-1 | 30 | (120,120) | 5 | 20 | 64 | 0.2 | 699269 | 0.8552 | 0.1300 |
| 6-2 | 40 | (120,120) | 5 | 20 | 64 | 0.2 | 699,269 | 0.8929 | 0.1600 |
| 7-1 | 40 | (120,120) | 5 | 20 | 128 | 0.2 | 1,113,925 | 0.9336 | 0.2000 |
| 7-2 | 50 | (120,120) | 5 | 20 | 128 | 0.2 | 1,113,925 | 0.8899 | 0.1600 |
| 8-1(filter Change) | 40 | (120,120) | 5 | 20 | 64 | 0.2 | 494,069 | 0.9397 | 0.1500 |
| 9-1 (optimiser=sgd) | 40 | (120,120) | 5 | 20 | 64 | 0.2 | 494,069 | 0.8115 | 0.1300 |
| 10-1 | 40 | (120,120) | 5 | 20 | 256 | 0.2 | 1,762,613 | 0.9774 | 0.2000 |
| 10-2 | 40 | (120,120) | 5 | 25 | 256 | 0.2 | 1,762,613 | 0.9593 | 0.1600 |
| 11-1 | 20 | (120,120) | 5 | 25 | 128 | 0.25 | 5,601,861 | 0.9819 | 0.3200 |
| 11-2 | 20 | (120,120) | 5 | 20 | 128 | 0.25 | 3,758,661 | 0.9713 | 0.2200 |
| 11-3 | 20 | (140,140) | 5 | 25 | 128 | 0.25 | 7,174,725 | 0.9698 | 0.2200 |
| 11-4 | 20 | (120,120) | 5 | 28 | 128 | 0.25 | 5,601,861 | 0.9683 | 0.3100 |
| 11-5 | 20 | (120,120) | 5 | 25 | 128 | 0.5 | 5,601,861 | 0.9095 | 0.2300 |
| 11-6 | 16 | (150,150) | 5 | 30 | 128 | 0.2 | 8,034,885 | 0.9638 | 0.1700 |
| 12-1(Removing BN layer) | 20 | (120,120) | 5 | 20 | 64 | 0.25 | 3,757,701 | 0.7587 | 0.7300 |
| 12-2 | 20 | (140,140) | 5 | 20 | 128 | 0.2 | 4,806,277 | 0.7451 | 0.6200 |
| 12-3 | 20 | (140,140) | 5 | 25 | 128 | 0.2 | 7,173,765 | 0.8854 | 0.7200 |
| 12-4 | 20 | (120,120) | 5 | 20 | 128 | 0.2 | 5,600,901 | 0.8130 | 0.7300 |
| 12-5 | 15 | (120,120) | 5 | 25 | 128 | 0.2 | 5,600,901 | 0.9050 | 0.8200 |
| 12-6 | 15 | (120,120) | 5 | 25 | 128 | 0.5 | 5600901 | 0.7195 | 0.7400 |
| 12-7 | 15 | (120,120) | 15 | 25 | 128 | 0.5 | 5,600,901 | 0.9849 | 0.8400 |
| 12-8 | 20 | (120,120) | 15 | 22 | 128 | 0.2 | 3757701 | 1.0000 | 0.8500 |
| 12-9 | 20 | (120,120) | 15 | 22 | 128 | 0.5 | 3757701 | 0.9668 | 0.7500 |
| 13-1(Removing padding) | 20 | (120,120) | 15 | 23 | 128 | 0.5 | 1,455,749 | 0.9698 | 0.8900 |
| 13-2 | 20 | (120,120) | 15 | 23 | 128 | 0.2 | 1,455,749 | 0.9517 | 0.8500 |
| 13-3 | 18 | (120,120) | 15 | 23 | 128 | 0.5 | 1,455,749 | 0.9623 | 0.8800 |
| 13-4 | 20 | (120,120) | 15 | 22 | 128 | 0.5 | 1,455,749 | 0.9668 | 0.8200 |
| 13-5 | 20 | (120,120) | 15 | 24 | 128 | 0.5 | 1455749 | 0.9457 | 0.8500 |

**CNN+RNN – Model Performances**

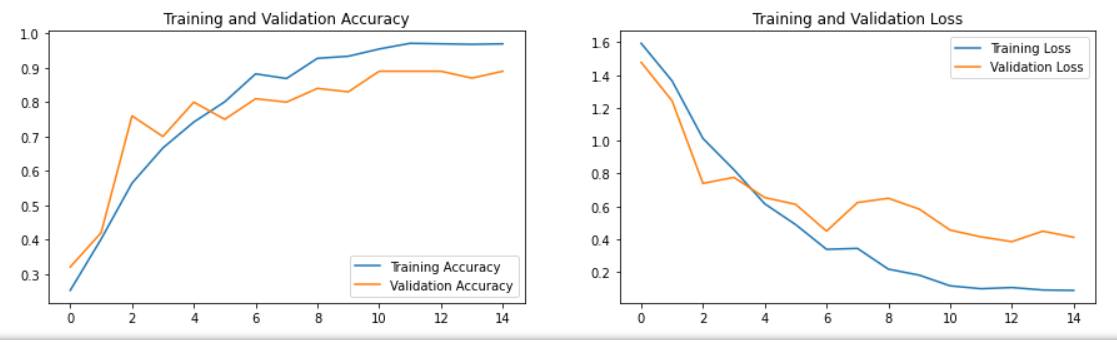
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model No. | Batch Size | Image Dimensions  (h,w) | Epoch | No. of Images | Dense Neurons | Dropout | Total  Params | Categorical Accuracy | Validation Accuracy |
| 1-1(LTSM) | 50 | (120,120) | 5 | 15 | 64 | 0.25 | 1,005,541 | 0.6546 | 0.2100 |
| 1-2 | 20 | (120,120) | 5 | 15 | 64 | 0.25 | 1,005,541 | 0.7391 | 0.2300 |
| 1-3 | 20 | (120,120) | 5 | 20 | 64 | 0.25 | 1,005,541 | 0.7195 | 0.2300 |
| 1-4 | 20 | (120,120) | 5 | 20 | 64 | 0.5 | 1,005,541 | 0.4465 | 0.2200 |
| 2-1(GRU) | 20 | (120,120) | 5 | 20 | 64 | 0.25 | 854,117 | 0.7979 | 0.2100 |
| 2-2 | 20 | (120,120) | 5 | 24 | 64 | 0.25 | 854,117 | 0.7557 | 0.1600 |
| 3-1(Simple RNN) | 20 | (120,120) | 5 | 20 | 64 | 0.25 | 550,693 | 0.7481 | 0.2400 |
| 4-1(RNN without BN) | 20 | (120,120) | 5 | 20 | 64 | 0.25 | 852,133 | 0.5385 | 0.5400 |
| 4-2 | 20 | (120,120) | 15 | 18 | 64 | 0.25 | 852,133 | 0.6244 | 0.5600 |

Hence, after training models with different Parameter values, the best mode is highlighted in yellow above. We can see each parameter has impact in model training.

**Best Model**

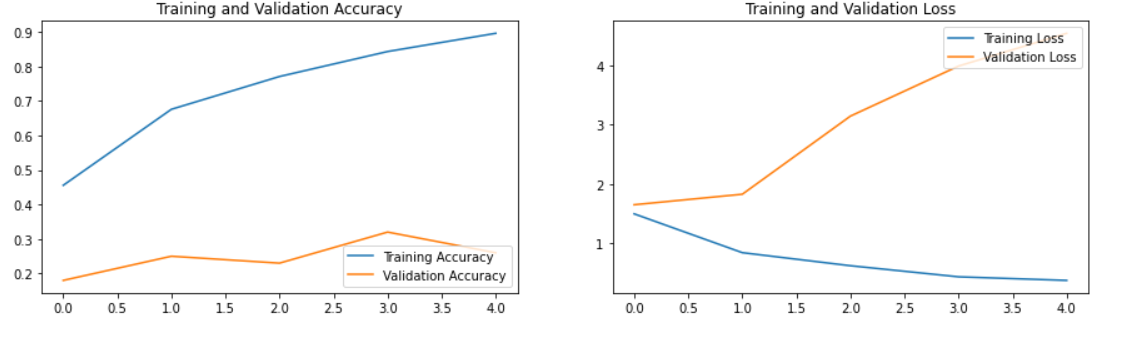
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 13-1 | 20 | (120,120) | 15 | 23 | 128 | 0.5 | 1,455,749 | 0.9698 | 0.8900 |

Below is the loss and accuracy graph for the best model.



 Observations:

1. **Activation-Relu , dense\_neurons-64, dropout-0.25 ,Batch size-50,number of images-20,Image size:120,120**



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.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1-6 | 50 | (120,120) | 5 | 15 | 64 | 0.25 | 997,637 | 0.8854 | 0.1600 |
| 2-1 | 50 | (120,120) | 5 | 15 | 128 | 0.25 | 1,932,485 | 0.8522 | 0.1700 |

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.

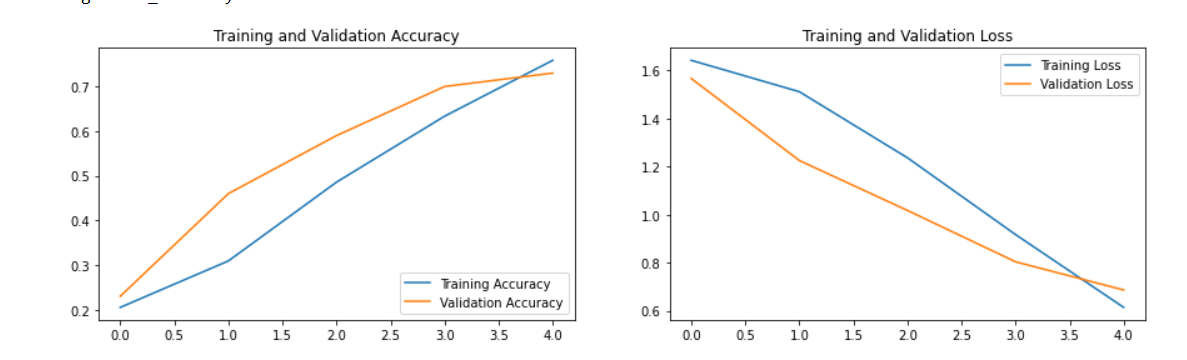
1. Adding one more conv3D layer: We have seen improvement in accuracy with adding one more conv3D layer.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1-7 | 40 | (120,120) | 5 | 20 | 64 | 0.25 | 1,919,237 | 0.9397 | 0.1100 |
| 3-2(with 1 more conv3D layer) | 40 | (120,120) | 5 | 20 | 64 | 0.25 | 694,853 | 0.9789 | 0.1900 |

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

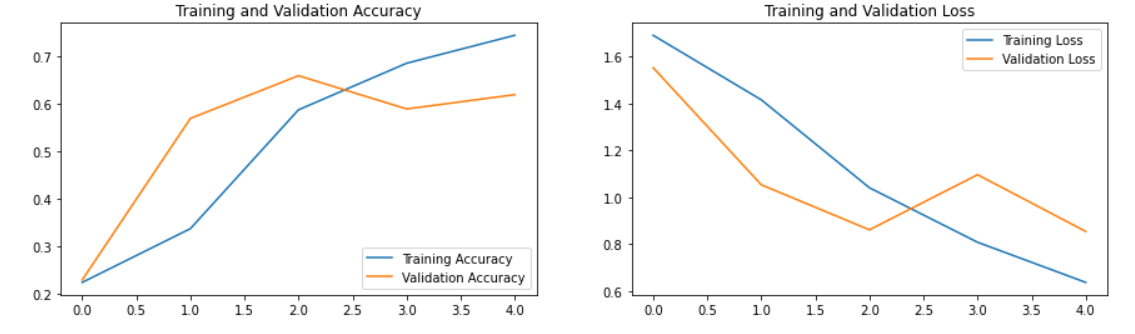
And the training and validation accuracy comes very close.

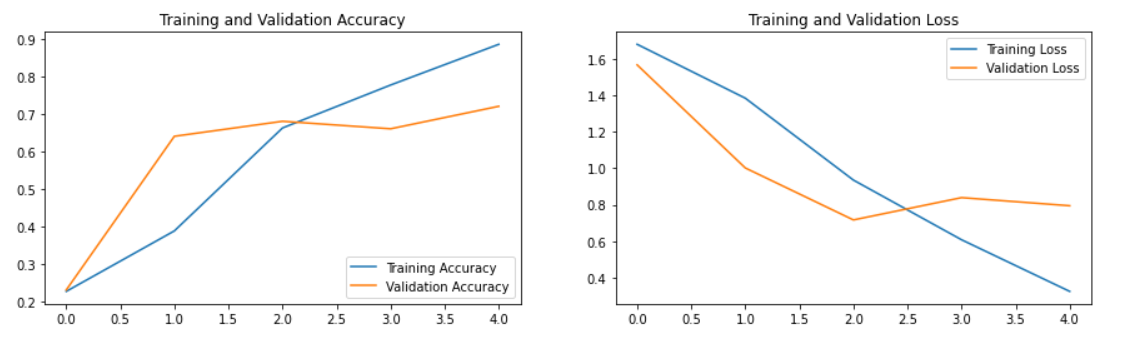
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 12-1(Removing BN layer) | 20 | (120,120) | 5 | 20 | 64 | 0.25 | 3,757,701 | 0.7587 | 0.7300 |



1. Increasing number of images: It has improved the validation accuracy to a great extend with increase in parameters. We have to find a trade off accordingly between increase in number of parameters and validation accuracy.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 12-2 | 20 | (140,140) | 5 | 20 | 128 | 0.2 | 4,806,277 | 0.7451 | 0.6200 |
| 12-3 | 20 | (140,140) | 5 | 25 | 128 | 0.2 | 7,173,765 | 0.8854 | 0.7200 |





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 more number of trainable parameters than GRU with almost same accuracy.