Hand gesture recognition using Neural Networks

Developers:

Narsingoju Naresh Chandra
Vollala Harish

Problem Statement:

Imagine you are working as a data scientist at a home electronics company which manufactures state of the art smart televisions. You 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:

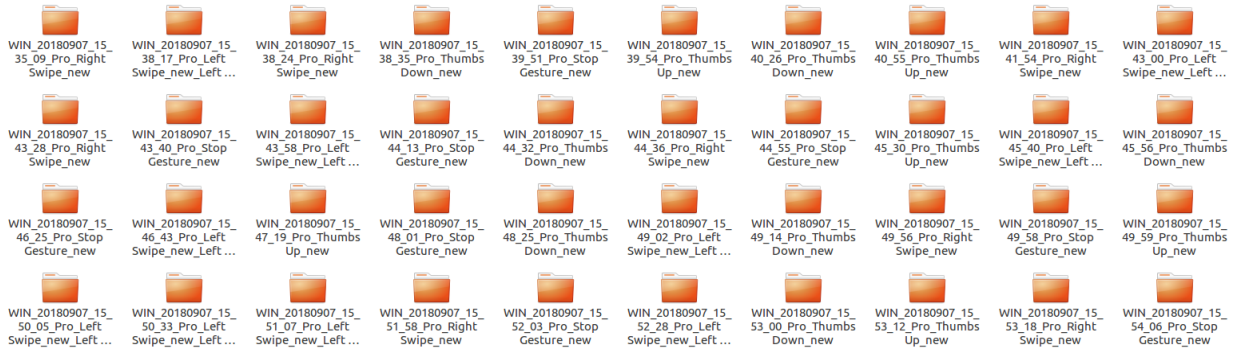
- Thumbs up: Increase the volume
- Thumbs down: Decrease the volume
- Left swipe: 'Jump' backwards 10 seconds
- Right swipe: 'Jump' forward 10 seconds
- Stop: Pause the movie

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

Understanding the Dataset:

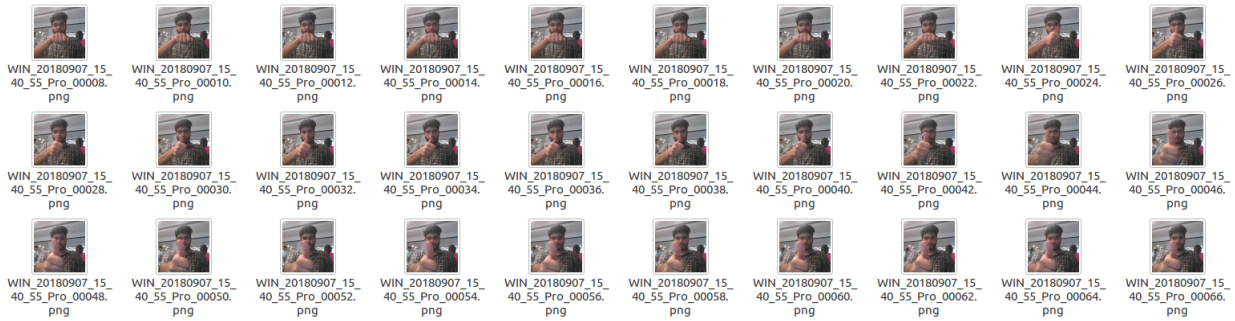
The training data consists of a few hundred videos categorised into one of the five classes. Each video (typically 2-3 seconds long) is divided into a sequence of 30 frames(images). These videos have been recorded by various people performing one of the five gestures in front of a webcam - similar to 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.

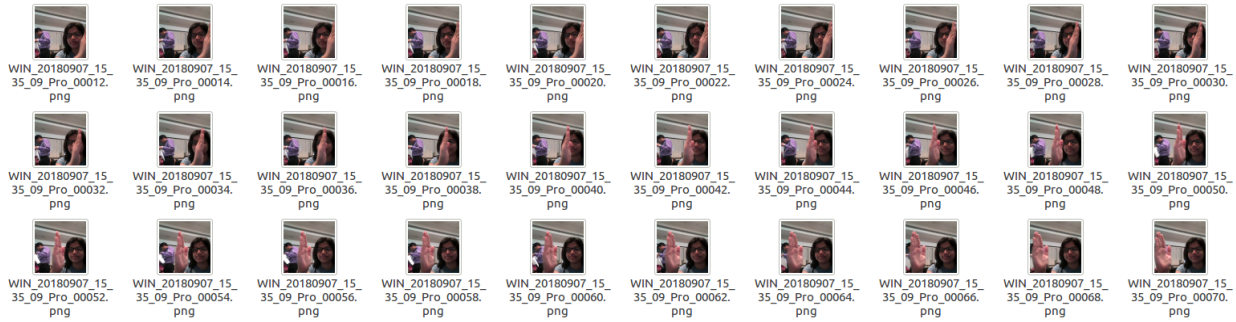


For Example, Each subfolder, i.e. a video, contains 30 frames (or images).

- Thumbs Up



- Right Swipe



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).

Two Architectures: 3D Convs and CNN-RNN Stack:

After understanding and acquiring the dataset, the next step is to try out different architectures to solve this problem.

For analysing 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.

Note:

- You can use transfer learning in the 2D CNN layer rather than training your own CNN
- GRU (Gated Recurrent Unit) or LSTM (Long Short Term Memory) can be used for the RNN

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

Google Colab Specifications:

RAM: 12.72GB
Disk: 68.40GB

Mounting Google Drive:

Mount Google drive running this two lines of code:

```
from google.colab import drive
drive.mount('/content/google_drive').
```

After running the above command, you need to enter the authorization code of your drive, click the link generated and give access to your data storage drive.

Reading and Understanding the Data

- Install the library ‘Pillow’, then library ‘scipy’ of version 1.2.1 for importing ‘imread()’ and ‘imresize()’ functions required for reading the image and video data.
- We use several libraries and modules like numpy, keras, tensorflow, datetime, os, random etc before reading the given ‘Project_data’.

Unpacking Zip File:

Then unzip ‘Project_data.zip’ using below code by importing and using shutil library.

```
# importing shutil module
import shutil
# Full path of the archive file
Project_data = "/content/google_drive/MyDrive/Project_data.zip"
# Unpack the archive file
shutil.unpack_archive(Project_data)
```

Generator Function

An Abstract Class named ModelBuilder is defined for generating data from Images in sets of batches. It contains below functions defined in it.

1. **Initialize_path:** initialises the path/location to the project folder and its subfolders.
2. **Initialize_image_properties:** Initialises image properties that are image resolution, number of channels of an image, number of classes labels/gestures, total number of frames folder contains.
3. **Initialize_hyperparams:** This function initializes hyperparameters for training models that include frames to sample, batch size, epochs.
4. **Generator :** This function yields the batch data & batch labels from the image data with selected number of frames in a sequence, augmenting data. The batch data is generated according to batch size parameter given, number of batches calculated from total training or validation sequences.
5. **One_batch_data:** This function is called from the above generator function for each batch to do cropping each image, image resizing or adjusting resolution of image according to hyperparameter set, normalizing the images in each batch data in the range of 0 to 1, augmenting or concatenating batch data from the different variations of an Image to increase the amount of variations of data which helps in handling overfitting of the model.
6. **Train_model:** This function is called for training the model we had designed which can save each epoch into the respective model folder with current date as reference name. This also sets number of training sequences, number of validation sequences, and fits the model on the generated batch data and return history of the model.
7. **Define_model:** abstract method inside abstract class.

Second abstract class is defined with more variations in augmentation which is very similar to the functions described above. It only differs from the above class in data augmentation functionality.

Tabulated Each different model parameters and Results for each experiment of model:

Experiment Number	Model	Total Parameters	Result	Decision + Explanation
1	**Type-1-Conv3D:** - Image Height-160, Image-width-160 - Frames to sample-30, - batch size-40 - epochs -1	1,736,389	loss: 1.9619 categorical_accuracy: **0.3195** val_loss: 2.5838 val_categorical_accuracy: **0.1600**	First, Experiment with this Image resolution, number of frames to use and batch_size., epochs. The model is overfitting too much with less accuracy. So reduce Frames to sample and batch size with more epochs.
2	**Type-1-Conv3D:** - Image Height-160, Image-width-160 - Frames to sample-16, - batch size-30 - epochs -3	1,736,389	loss: **0.9148** categorical_accuracy: 0.6578 val_loss: 7.6354 val_categorical_accuracy: **0.1600**	Second, Experiment with this Image resolution, number of frames to sample with reduced frames and batch_size, more epochs. The model is overfitting too much with less accuracy.

3	**Type-1-Conv3D:** - Image Height-100, Image-width-100 - Frames to sample-30, - batch size-30 - epochs -2	6,87,813	loss: 1.0943 categorical_accuracy: **0.5762** val_loss: 2.6685 val_categorical_accuracy: **0.18**	Third, Experiment with this Image resolution, number of frames to use and batch_size, with reduced image resolution and epochs and using full 30 frames in a sequence accuracy is Worst.
4	**Type-1-Conv3D:** - Image Height-100, Image-width-100 - Frames to sample-30, - batch size-60 - epochs -2	6,87,813	loss: 1.3048 categorical_accuracy: **0.4911** val_loss: 3.8390 val_categorical_accuracy: **0.24**	Fourth, Experiment with Image resolution, number of frames to use and batch_size increased to 60. The model is overfitting too much with less accuracy.
5	**Type-1-Conv3D:** - Image Height-100, Image-width-100 - Frames to sample-16, - batch size-60 - epochs -2	6,87,813	loss: 1.0997 categorical_accuracy: **0.5742** val_loss: 2.0025 val_categorical_accuracy: **0.13**	Fifth, Experiment with Image resolution, number of frames to use and more batch_size=60 reducing number of frames to sample to 16. The model is overfitting too much with less accuracy.

6	**Type-2-Conv3D:** - No Data Augmentation, - (3,3,3) filter - Image Height-160, Image-width-160 - Frames to sample-20, - batch size-40 - epochs -15	1,117,061	loss: 0.1948 categorical_accuracy: **0.9684** val_loss: 7.5372 val_categorical_accuracy: **0.22**	Sixth, Experiment with Changing filter size to (3,3,3) with no data augmentation. This clearly shows overfitting on the data. So, Data augmentation is required.
7	**Type-2-Conv3D:** - with Data Augmentation, - (3,3,3) filter - 160 X 160 image resolution - Frames to sample-20, - batch size-20 - epochs -25 - Dense neurons-256 - dropout-0.5	3,638,981	loss: 0.5393 categorical_accuracy: **0.8073** val_loss: 0.5476 val_categorical_accuracy: **0.84**	Seventh, Experiment with Changing filter size to (3,3,3), adding layers of Dense nuerons and dropout with data augmentation. This clearly shows data augmentation solves the overfitting problem.
8	**Type-3-Conv2D-LSTM:** - 120 X 120image resolution - Frames to sample-18, - batch size-20 - epochs -20 - Dense neurons-128 - dropout-0.25 - lstm cells-128	1,657,445	loss: 0.3385 categorical_accuracy: **0.8896** val_loss: 0.6991 val_categorical_accuracy: **0.68**	Eighth, Experiment using new architecture of conv2D-LSTM. This model also overfits as there is no data augmentation. So, more augmentation is required to handle overfitting problems.

9	**Type-4-Conv3D** - Data Augmentation - (3,3,3) filter size - 160X160 image resolution - frames to sample-20 - batch size-20 - epochs-30 - dense neurons-256 - dropout-0.50	3,638,981	loss: 0.7063 categorical_accuracy: **0.7518** val_loss: 0.7470 val_categorical_accuracy: **0.7400**	Ninth, Experiment using new architecture of conv2D-LSTM. Adding more augmentation gives us a robust model without overfitting. But accuracy percentages are not enough and should improve.
10	**Type-5-Conv3D:** - Data Augmentation - (2,2,2) filter size - 120X120 image resolution - frames to sample-16 - batch size-30 - epochs-30 - dense neurons-256 - dropout-0.50	1,762,613	loss: 0.9054 categorical_accuracy: **0.6749** val_loss: 0.7770 val_categorical_accuracy: **0.7400**	Conv3D model reducing filter size to (2,2,2). This also is not giving much high accuracy and underfits on the data.
11	**Type-6-Conv3D: Adding more layers** - Data Augmentation - (3,3,3) filter size - 120X120 image resolution - frames to sample-16 - batch size-20 - epochs-30 - dense neurons-256 - dropout-0.50	2,556,533	loss: 0.7845 categorical_accuracy: **0.7147** val_loss: 0.6220 val_categorical_accuracy: **0.8000**	Adding more layers to the conv3D model gives an underfitting model in 30 epochs. But training in more epochs or early stopping the model can solve this problem.

12	**Type-7-Conv3D: Adding Dropouts:** - Data Augmentation - (3,3,3) filter size - 120X120 image resolution - frames to sample-16 - batch size-20 - epochs-25 - dense neurons-256 - dropout-0.25	2,556,533	loss: 0.9164 categorical_accuracy: **0.6491** val_loss: 3.5468 val_categorical_accuracy: **0.2700**	Adding more Dropouts is making the model poor in accuracy and robustness. So, let's try other architectures with the Conv2D-LSTM model next and check for better performance.
13	**Type-8-Conv2D-LSTM with GRU:** - Data Augmentation - 120X120 image resolution - frames to sample-18 - batch size-20 - epochs-20 - dense neurons-128 - lstm-cells-128 - dropout-0.25	2,573,925	loss: 0.2342 categorical_accuracy: **0.9457** val_loss: 0.8364 val_categorical_accuracy: **0.7300**	This architecture CNN-LSTM is showing some training accuracy more than 90% but still overfits having categorical more than validation accuracy percentages Then we shall try some transfer learning on the data with mobile net architecture.
14	**Type-9-Transfer Learning using Mobilenet:** - Data Augmentation - 120X120 image resolution	3,840,453	loss: 0.0373 categorical_accuracy: **0.9871** val_loss: 0.6280	Transfer learning has given some good results but it still overfits. Let's use all

	- frames to sample-16 - batch size-5 - epochs-20 - dense neurons-128 - lstm-cells-128 - dropout-0.25		val_categorical_accuracy: **0.8300**	the weights to train the model in the next model.
15 **(Final Model)**	**Type-10-Transfer Learning with GRU using all weights:** - Data Augmentation - 120X120 image resolution - frames to sample-16 - batch size-5 - epochs-20 - dense neurons-128 - gru-cells-128 - dropout-0.25	3,693,253	loss: 0.0100 categorical_accuracy: **0.9970** val_loss: 0.1059 val_categorical_accuracy: **0.9800**	Best model possible without overfitting and least error.

Plots of final model with transfer learning with file size of 42.4MB to fit in the memory of a webcam.