Gesture Recognition with Deep Networks

Iron Man demonstrating futuristic holographic technology controlled by gestures

Group Members

Saurav Mishra
Debasis Garabadu
Sandhita Agarwal
Senjuti Kar

Contents

List of Figures

Introduction

Gesture recognition is a topic in computer science and language technology with the goal of interpreting human gestures via mathematical algorithms. Gestures can originate from any bodily motion or state but commonly originate from the face or hand. Current focuses in the field include emotion recognition from face and hand gesture recognition. Users can use simple gestures to control or interact with devices without physically touching them. Many approaches have been made using cameras and computer vision algorithms to interpret sign language. However, the identification and recognition of posture, gait, proxemics, and human behaviors is also the subject of gesture recognition techniques. Gesture recognition can be seen as a way for computers to begin to understand human body language, thus building a richer bridge between machines and humans than primitive text user interfaces or even GUIs (graphical user interfaces), which still limit the majority of input to keyboard and mouse and interact naturally without any mechanical devices. Using the concept of gesture recognition, it is possible to point a finger at this point will move accordingly. **(Ref - https://en.wikipedia.org/wiki/Gesture_recognition)**

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 recognize five different gestures performed by the user which will help users control the TV without using a remote.

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

Data

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 - like what the smart TV will use.

1. Training
2. Validation

Each training and validation set consists of several sub folders. Each subfolder consists of frames of one video (30 frames for a gesture). Also, two csv files which consist of the video metadata (i.e. path of the video and its label).

All images/frames of a video are of same dimension, there are two variants of the video image dimensions: 360x360 and 120x160

Let's see how the sequence of 30 frames end up forming a gesture.

Figure 1 - Sequence for frames leading to a gesture

Solution Approach

The following steps were followed in-order to classify the gestures correctly -

1. Data Understanding.
2. Data preprocessing and cleaning.
3. Define the model architectures
4. Train the models.
5. Model performance comparison.
6. Final model selection.

Let's understand each phase one by one.

Data Understanding

Data provided consists of the training and validation sets. Each set consists of several sub folders, each sub folder having 30 frames.

Input video frame dimensions:

- 360x360x3
- 120x160x3

Preprocessing

Following two steps are performed in preprocessing:

1. **Resize**: In the preprocessing step we have converted two different size of frames into one dimension of size 120x120. We have images are of 2 different shape and the conv3D will throw error if the inputs in a batch have different shapes. We will crop the image to (120, 120, 3), if the shape is (120, 160, 3)and we will resize to (120, 120, 3), if the shape is (360, 360, 3)

2. **Normalization**: Rescale pixel values from the range of 0-255 to the range 0-1 preferred for neural network models. Image data is normalized by dividing pixel values by 255.

Along with the data normalization and resizing, we need to implement generator function. which will feed data to the model for each epoch. We need to implement generator function which can feed equal size batches of sequences and take care of remaining sequences.

For example, with training size of 23 sequences and batch size of 5 leads to 5 batches as below-

Batch-1: 5 sequences
Batch-2: 5 sequences
Batch-3: 5 sequences
Batch-4: 5 sequences
Batch-5: 3 sequences

Model Architecture

There are 3 different deep learning architectures which can be used to build model for gesture recognition. Let's see those architectures below.

CNN + RNN

This is the standard architecture for processing videos. In this architecture, video frames are passed through a CNN layer which extracts features from the images and then these feature vectors are fed to an RNN network to simulate sequence behavior of the video. Output of RNN is regular SoftMax function/

1. We can use transfer learning in 2D CNN layer instead of training own network.
2. LSTM or GRU can be used in RNN

3D Convolution Network or Conv3D

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, the filter is moved 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 100x100x3, for example, the video becomes a 4-D tensor of shape 100x100x3x30 which can be written as (100x100x30)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 (100x100x30) tensor.

Transfer Learning

We will use some pre-trained models and try to use their knowledge to classify our cases correctly. These pre-trained models which are trained on millions of images may prove vital in solving our problem efficiently. For transfer learning -

- We have done our experiments with the following pre-trained models -
    1. MobileNet
    2. InceptionResNetV2
    3. EfficientNetB0
- We experimented several such models in the **2D CNN** layer.
- We experimented with GRU units as well as LSTM units.
- We also experimented transfer learning by setting the layers of the pre-trained models set to –
    1. Trainable - False
    2. Trainable – True

Many models were tested based on the above architectures. These models were created taking into consideration the following model parameters while performing a series of experiments throughout the course of the project –

- Depth of the model - The number of layers a model could contain.
- Batch Normalization for the hidden layers.

- Dropout layers to de-activate some of the neurons randomly.
- Dense Layer architecture.
- The Optimizer functions.
- Adding the L2 regularization to some models.
- Time Distributed Layers – LSTM's and GRU's.
- Transfer Learning.

Model Training History

One of the default callbacks that is registered when training all deep learning models is the **History** callback. It records training metrics for each epoch. This includes the loss and the accuracy (for classification problems) as well as the loss and accuracy for the validation dataset, if one is set.

The history object is returned from calls to the fit() or the fit_generator() function used to train the model. Metrics are stored in a dictionary in the history member of the object returned. For a classification problem the history callback returns - [**'acc', 'loss', 'val_acc', 'val_loss'**]

We have made use of the history callback to access the training history for our models and plot the **loss**, **val_loss** and **accuracy**, **val_accuracy** against the number of epochs.

The plots give us insights to -

- It's speed of convergence over epochs (slope).
- Whether the model may have already converged (plateau of the line).
- Whether the mode may be over-learning the training data (inflection for validation line).

(Ref - https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/)s

Model Comparison

A brief and tabular comparison for the various models created for this task. The models are being evaluated and compared based on the metrics provided the model history callback.

- Training accuracy
- Validation accuracy
- Loss incurred during the training phase
- Loss incurred during the validation phase

**** NOTE – Our best models in the below table are in bold and font green.**

Experiment Number	Model Architecture	Result	Decision + Explanation
Model 1	**Conv2D + RNN** Time Distributed Model 1 with LSTM - 5 Convolutional Layers	**loss:** 0.2615 **categorical_accuracy:** 0.9254 **val_loss:** 0.7647 **val_categorical_accuracy:** 0.7000	We got the best validation accuracy as 70% when training accuracy is 92.54%. The result is not satisfactory. Thus, Model 1 is not able to learn much from the training data.
Model 2	**Conv2D + RNN** Time Distributed Model 2 with LSTM - 6 Convolutional Layers	**loss:** 0.0253 **categorical_accuracy:** 0.9955 **val_loss:** 0.5846 **val_categorical_accuracy:** 0.8300	We got validation accuracy as 83% when training accuracy is 99.55%, which is a good model in terms of CNN (CONV2D) + RNN. Thus, Model 2 can learn better than the model 1 in terms of capturing information from the training data
Model 3	**Conv2D + RNN** Time Distributed Model with GRU units	**loss:** 0.0264 **categorical_accuracy:** 0.9940 **val_loss:** 0.4969 **val_categorical_accuracy:** 0.8400	**We got the validation accuracy as 84% when training accuracy is 99.4%. This is a slight improvement from model 2 and is a very good score in terms of CNN (CONV2D) + RNN.** **Thus, Model 3 can learn & capture information better that the previous model**
Model 4	**Conv3D** MaxPooling3D model with 4 Convolutional Layer	**loss:** 0.4572 **categorical_accuracy:** 0.8518 **val_loss:** 0.4768 **val_categorical_accuracy:** 0.8200	Validation Accuracy is 82% when Training Accuracy is 84.77%. As compared to the models 2 & 3 there is a slight drop in the learning capability of this model. This is not as good when compared to the current benchmark scores.
Model 5	**Conv3D** MaxPooling3D Model with 5 Convolutional Layers	**loss:** 0.8853 **categorical_accuracy:** 0.6462 **val_loss:** 0.6357 **val_categorical_accuracy:** 0.7600	With a validation accuracy of 76%, this is not a good model when compared with previous models. This is not acceptable.

Model 6	**Conv3D** Adding more similar convolution layers to make the model deeper – 8 convolution layers.	**loss:** 1.2465 **categorical_accuracy:** 0.4871 **val_loss:** 1.0427 **val_categorical_accuracy:** 0.5700	The model returns Validation Accuracy of just 57% which is very low compared to the benchmark scores. This model is not acceptable.
Model 7	**Conv3D** Adding dropout layer after each convolution layer.	**loss:** 2.3495 **categorical_accuracy:** 0.2393 **val_loss:** 2.8366 **val_categorical_accuracy:** 0.2700	We got a Validation Accuracy as a mere 27%. Also, as the loss did not decrease for 10 consecutive epochs, the model stopped training after the 11-epochs due to the Early Stopping callback implemented
Model 8	**Conv3D** Adding L2 regularizer	**loss:** 1.4558 **categorical_accuracy:** 0.6408 **val_loss:** 1.2641 **val_categorical_accuracy:** 0.7500	We got a Validation Accuracy of 75% with the val_loss at 1.26411. There is too much loss of information and the accuracy is not up to the expectation
Model 9	**Conv3D** L2 regularizer with add an extra convolution layer added to model 8	**loss:** 1.3442 **categorical_accuracy:** 0.6886 **val_loss:** 1.2432 **val_categorical_accuracy:** 0.7300	We got a Validation Accuracy of 73% with a corresponding val_loss at 1.24322. There is too much loss of information and the accuracy is not up to the expectation
Model 10	**Conv3D** **Adding the 256 and 128 units to the Dense layers**	**loss: 0.7510** **categorical_accuracy: 0.9398** **val_loss: 1.0364** **val_categorical_accuracy: 0.8400**	**We got a Validation Accuracy of 84% & a training accuracy of 93.98%. The numbers look good when compared to the previous conv3D models & can infer that this model has learnt a lot more.** **This turns out to be a good model and now the bench mark score for conv3D model is set as 84%.**
Model 11	**Conv3D** Adding more convolution layers to model 10	**loss:** 2.1615 **categorical_accuracy:** 0.7583 **val_loss:** 2.1542 **val_categorical_accuracy:** 0.8000	This model does not perform as good as the previous one. We get the validation accuracy of 80%.

Model 12	**Transfer Learning** MobileNet with LSTM Layers	**loss:** 0.5453 **categorical_accuracy:** 0.7950 **val_loss:** 1.1403 **val_categorical_accuracy:** 0.5700	We got a Validation Accuracy of 57%. This turns out to be a bad model and unacceptable performance.
Model 13	**Transfer Learning** MobileNet with GRU Layers	**loss:** 0.0257 **categorical_accuracy:** 0.9955 **val_loss:** 0.0840 **val_categorical_accuracy:** 0.9600	The Validation Accuracy is 96% when Training Accuracy is 99.7%. This is a major improvement considering the Transfer Learning Model.
Model 14	**Transfer Learning** InceptionResnetV2 with trainable GRU Layers	**loss:** 0.0118 **categorical_accuracy:** 0.9970 **val_loss:** 0.14465 **val_categorical_accuracy:** 0.9500	We got a Validation Accuracy as a mere 95% when Training accuracy is 99.7%. This turns out to be a very good classifier model
Model 15	**Transfer Learning** **EfficientNetB0 with trainable GRU Layers**	**loss: 0.0048** **categorical_accuracy: 0.9985** **val_loss: 0.0440** **val_categorical_accuracy: 0.9800**	**We got a Validation Accuracy of 98% while the Training Accuracy is 99.85% with Validation loss at 0.044. This is the best model we got using Transfer Learning.**

Additional Experiments

In addition to the above architectures and models, each of the model was experimented with the following parameters also –

Epochs – We tried training with 20 epochs, 50 epochs and 100 epochs. It was observed that there is no improvement of validation loss after epoch 40. Thus, we decided to keep 40 epochs as the optimum one.

Batch Size – We tried with batch size 5, 10, 15 and 20. It was observed that our GPU was not able to handle a batch size of 20 and 15 that well. We experimented with batch size 10 and our machine was able to handle it properly. So, we kept batch size 10 as the optimum one.

Trainable Layers – Incase of the transfer learning models we experimented with the layers of the pre-trained models set to **trainable - true** and **trainable - false**. The results observed were –

- Trainable - False: The results obtained were not satisfactory and the models performed badly.
- Trainable - True: The models performed exemplary and the accuracy obtained were on the higher side of 90%.

Early Stopping - Some of the models were also tested with the Early Stopping call back mechanism with two different patience levels of 5 and 10.

Model Storage Location

Each of the models have their corresponding .h5 files. Each .h5 file is stored in google drive which can be found at –

https://drive.google.com/open?id=1Nbv3V30b3qzrubp79HSibzCWpWnmrJHu

Summary

In this case study, we experimented on 3 different types of model architectures for gesture recognition:

1. CNN (Conv2D) + RNN
2. CNN (Conv3D)
3. CNN (Transfer Learning) + RNN

Based on the model performance we selected our top three models out of the 15 models we tested. A high-level summary of these 3 models are written out below.

CNN (CONV2D) + RNN

Model 3 is the best model in terms of CNN(CONV2D) + RNN. It achieved a validation accuracy of 84% consistently without overfitting the training data and has a training accuracy between 90-94%.

Model 3 is designed using 5 time distributed - convolution2D layers having the Relu activation function. Each layer has a Batch Normalization layer implemented following a MaxPooling2D layer with a pool size (2, 2,) followed by a flatten layer. It has a GRU unit with a dropout rate of 0.5. The last layer in the Dense layer with 5 units having softmax activation.

Model-3 History Plot

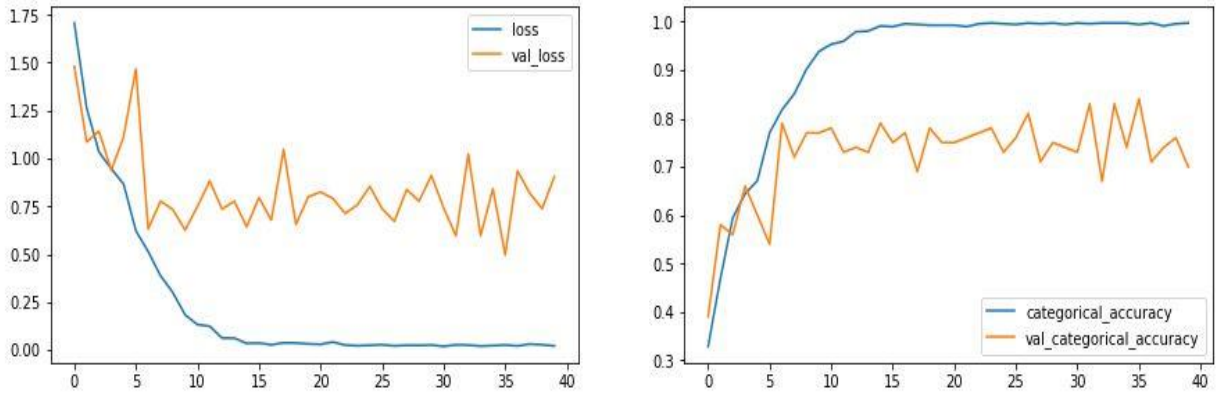

Figure 2 - Model - 03: Performance History vs Epochs

Model-3 Link to h5 file

https://drive.google.com/open?id=1C-tdDLhXJG2b6Kc9RL3zHWkaGReMil4Y

Conv3D

Model 10 is the best model in terms of CNN - Conv3D models. It got a validation accuracy of 84% consistently without overfitting the training data and has a training accuracy between 90-95%.

Model 10 is designed using 4 convolution3D layers having the Relu activation function. Each layer has a Batch Normalization layer implemented following a MaxPooling3D layer with a pool size (2, 2, 2). It is followed by a flatten and a dense layer each with 256 and 128 units respectively. These two layers have a dropout rate of 0.5 to control overfitting. Finally, it has the softmax layer with 5 units.

Model-10 History Plot

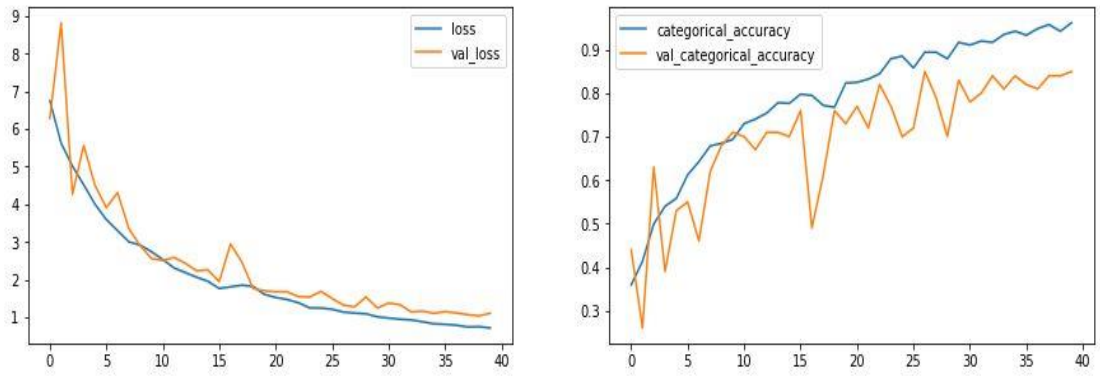

Figure 3 - Model - 10: Performance History vs Epochs

Model-10 Link to h5 file

https://drive.google.com/open?id=1Lz51tUaxvPV3m9dZUPt9Z9mmtShrVZB_

CNN (Transfer Learning) + RNN

Model 15 is the best model in terms of CNN (Transfer Learning) + RNN. It got a validation accuracy of 98% consistently without overfitting the training data and has a training accuracy between 98-99%.

Model 15 is designed using the EfficientNetB0 pre-trained model. The base model is fed into a time distributed layer and then into a flatten layer and finally a GRU unit is added with Relu activation having 256 units and a dropout of 0.5 to control overfitting.

**Note: The keras_efficientnets module required to import and run EfficientNet have been included in the folder - keras_efficientnets. Import keras_efficientnets and call either the model builder EfficientNet or the pre-built versions EfficientNetBX where X ranges from 0 to 7.
The modules contain code to build the EfficientNets B0-B7 and includes weights for configurations B0-B3.

Model-15 History Plot

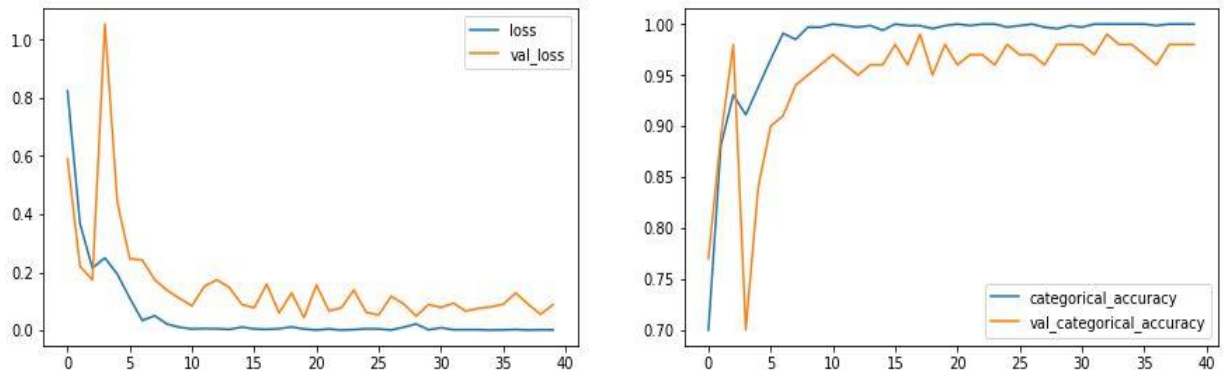

Figure 4 - Model - 15: Performance History vs Epochs

Model-15 Link to h5 file

https://drive.google.com/open?id=192mnQR2p-H-KZpEfpKbSq6VPP0Koevvh

Final Verdict

From the summary above we see that our transfer learning model built using **EfficientNetB0** pre-trained model achieves the best validation accuracy of 98% without overfitting on the training data. The training accuracy stands at 99.85% with a very minimal loss of information.

Thus, we choose Model 15 as the best model for the task of gesture recognition.