Gesture Recognition

Model Design and Prediction

Sanjeev S

2/14/23

Master of Science in Machine Learning & Artificial Intelligence

Table of Contents

1. Project Overview 2
1.1 Data Set Brief Information 2
1.2 Objective and Solution 2
1.3 Data Analysis 3
1.3.1 Memory Calculation 3
2. Base Model Designing 4
2.1 Model Creation and Initialization 4
2.2 Data Generator 4
2.3 Training 4
3. Model Generation 6
4. Model Summary 8
4.1 Conclusion 10

List of Figures

Figure 1: Sample Dataset from Train Folder 2
Figure 2: Original and Augmented image 4
Figure 3: 3D CNN 6
Figure 4: CNN + RNN 6

List of Tables

Table 1: Model Summary 8

1. Project Overview

As a data scientist at a home electronics company which manufactures state of the art smart televisions, 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:

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

1.1 Data Set Brief Information

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 – like 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). 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).

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.

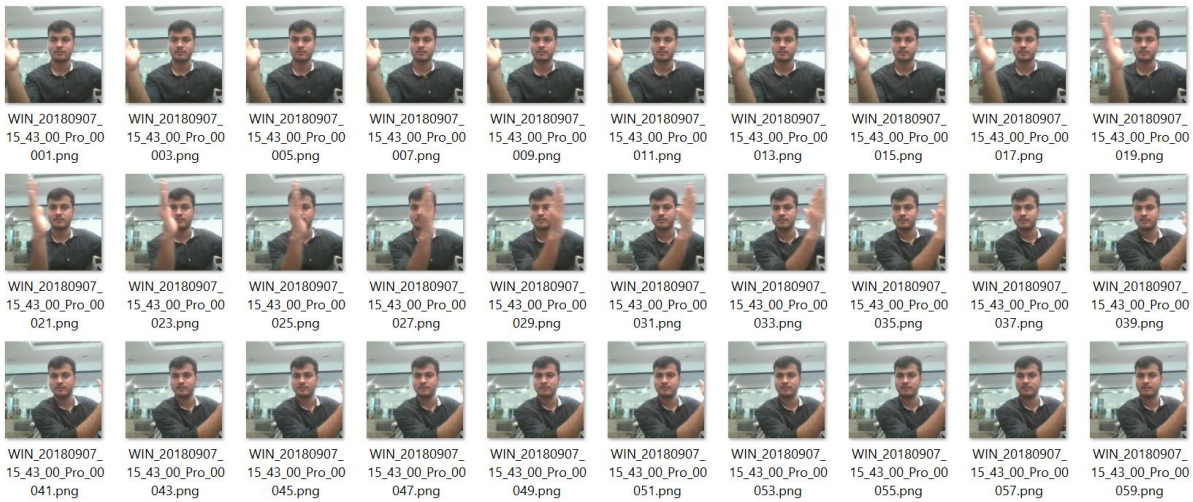

Figure 1: Sample Dataset from Train Folder

1.2 Objective and Solution

Use different architectures to build a DL model to classify 5 gestures correctly by training the model on the 'train' folder which performs well on the validation 'val' folder as well.

The trained model should be fit enough to be used in small web camera and identify the 5 gestures in real-time.

1.3 Data Analysis

- Each video contains 30 frames.
- Frame is either 360x360 or 120x160 resolution.
- RGB colour image - 3 channels.
- Training datasets have 663 images.

1.3.1 Memory Calculation

- If we go for high resolution (360x360) and 30 frames per video, for a batch size of 32, it will be around:
  = (Image Height x Image Width x Channels x Images per Video x Total Dataset) / (1 GB in Bytes)
  = (360 x 360 x 3 x 30 x 663) / (1024 x 1024 x 1024)
  = ~7 GB
- This will eat up memory if we have greater than 1 million params in designed model. And, also it requires more time to train.
- If we go for lower resolution (120x160) and 30 frames per video, for a batch size of 32, it will be around ~1 GB. This we could manage further down by adjusting images per video and resolution.
- Batch size doesn't matter in terms of memory, since we will train for entire 663 dataset! The only thing it affects is the speed of training per epoch. If batch size is more, then number of batch data per epoch will be reduced.
- We have huge dataset, so we need to adjust memory and speed to get good amount of accuracy and the model should fit in web camera for real-time classification.

2. Base Model Designing

A base class was designed to instantiate different models and adjust parameters for training. The base class will have abstract method to define required model for training!

2.1 Model Creation and Initialization

A folder path is passed as argument to initialize the base class. Train and Validation folders are identified from given path and as well as the respective .csv files. The CSV file contains list of sub-folders and its corresponding gesture label.

User can adjust the height and width of image, number of epochs, batch size and images to process per video.

2.2 Data Generator

This function is the heart of the base model class and is responsible for supplying batches of sequence of images from dataset. Using random permutation with initial hardcoded seed value, sub-folders are shuffled and required number of images are read.

- **Resizing** - The images are then pre-processed by adjusting the resolution.
- **Normalization** - The images are normalized by dividing each RGB channel data by 255.
- **Cropping** - If user wants augmented data, each image is duplicated, and slight cropping is done to make a new image. This is to ensure that the model will predict the result accurately even if the end user places their hands outside web-camera's boundary frame.

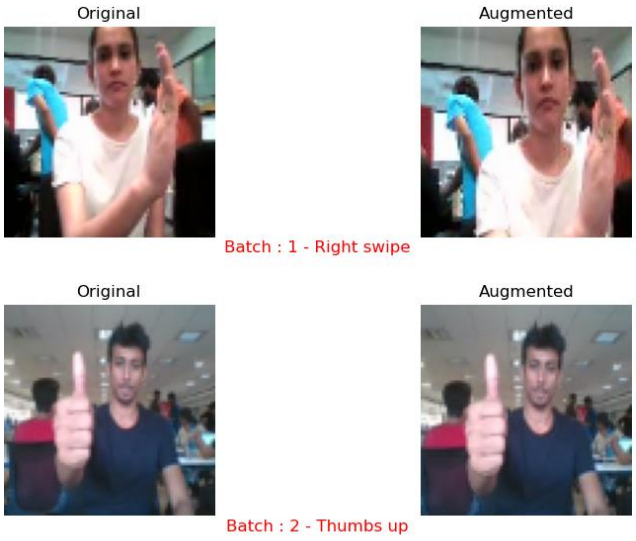

Figure 2: Original and Augmented image

2.3 Training

User can pass the required model to be fitted for training and the function validates the accuracy of validation dataset.

Callback functions are introduced to save time and model.

- **ModelCheckpoint** – callback is used in conjunction with training using model.fit() to save a model or weights (in a checkpoint file) at some interval, so the model or weights can be loaded later to continue the training from the state saved. With *save_best_only* feature, we will save only the best model.

- **ReduceLROnPlateau** – callback is used to reduce learning rate when a metric has stopped improving.
- **EarlyStopping** – callback is used to Stop training when a monitored metric has stopped improving.

3. Model Generation

For this case study, 3 types of models were developed and analysed its performance.

- **3D Convolutional Neural Network (CNN)**

  A 3D CNN remains regardless of what we say a CNN that is very much like 2D CNN. Except that it differs in 3D matrix multiplication.

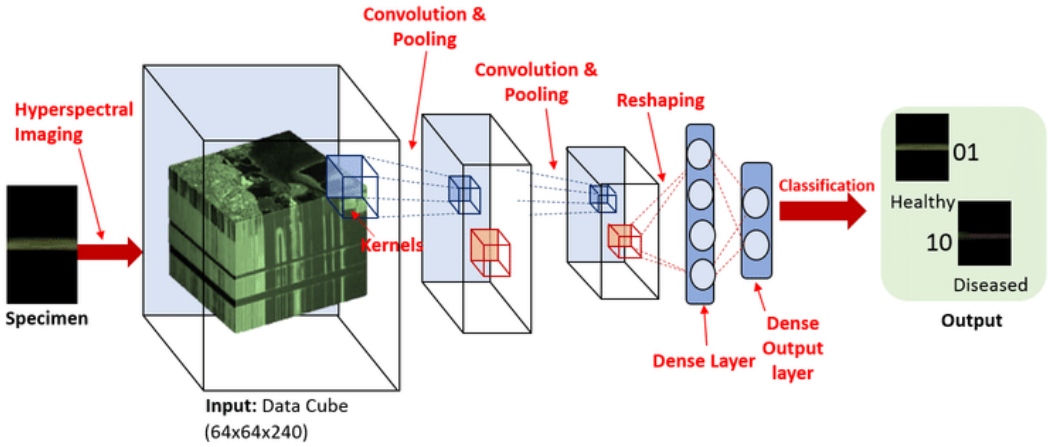

Figure 3: 3D CNN

- **CNN + Recurrent Neural Network (RNN)**

  In this architecture, both Time distributed 2D CNN and RNN are stacked together. We will 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. The output of the RNN is a regular softmax (for a classification problem such as this one).

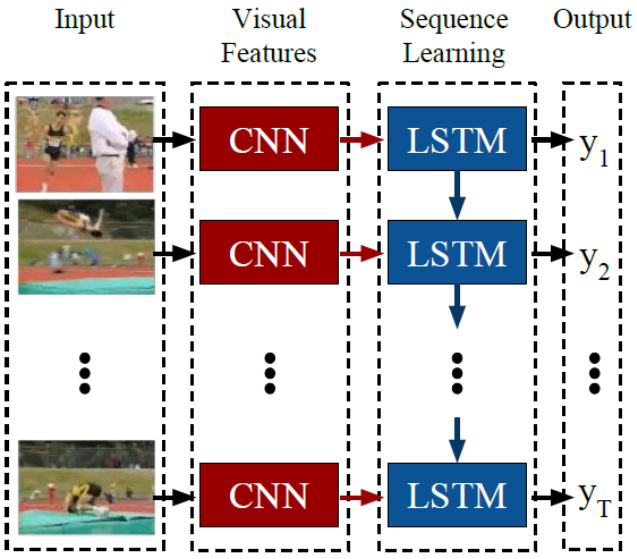

Figure 4: CNN + RNN

- Pre-trained Network with RNN

  A pre-trained neural network like VGG, Reset etc is used to extract feature from video images and then pass the information to RNN. This is similar to second point and the only difference is that we are using pre-trained CNN layers instead of our own CNN design.

4. Model Summary

We have designed various models to classify the 5 gestures. Let's look at quick summary and understand the model explanation.

- Experimented with different model configurations and hyper-parameters. Tried various combinations of batch sizes, image resolution and filter sizes to reduce memory and increase processing speed.
- categorical_accuracy is used to predict one-hot labels.
- categorical_crossentropy loss function was used for multi-class classification model when there are two or more output labels.
- Different learning rates and ReduceLROnPlateau was used to decrease the learning rate if the monitored metrics (val_loss) remains unchanged in between epochs.
- We chose Adam optimizer since they are generally better than every other optimization algorithm, have faster computation time, and require fewer parameters for tuning.
- Dropout layers are used to avoid overfitting and control regularization.
- Early stopping was used to put a halt at the training process when the val_loss would start to saturate / model's performance would stop improving.
- As the Number of trainable parameters increase, the model takes much more time for training.
- 2DCNN-RNN structure didn't improve the accuracy over 3DCNN model.
- Transfer learning boosted the overall accuracy of the model.

Table 1: Model Summary

Model Number	Model	Result	Decision and Explanation
1	3D CNN	Training Accuracy - 85.82 % Training Loss - 0.35 Validation Accuracy - 85 % Validation Loss - 0.42	Images was resized to 120x160 for reducing memory footprint. Designed several convolution layers to learn many features as possible. Added dropout layer to regularize and to avoid overfitting. This model is little bit heavy but still manageable and metrics is acceptable for testing in real-time predictions.
2	3D CNN	Training Accuracy - 76.62 % Training Loss - 0.57 Validation Accuracy - 69 % Validation Loss - 0.71	Structure is same as model 1. We reduced image count for fast processing. Since images to use per video is reduced, we added more neurons in the dense layer and increased dropout rate to unlearn. Model performance went down and is not acceptable. It might be due to dropout rate.
3	3D CNN	Training Accuracy - 84.62 %	Structure is same as model 2.

		Training Loss - 0.40 Validation Accuracy - 74 % Validation Loss - 0.67	We reduced kernel size and increased the learning rate for fast processing. Performance improved with learning rate.
4	3D CNN	Training Accuracy - 76.47 % Training Loss - 0.59 Validation Accuracy - 74 % Validation Loss - 0.69	Structure is same as model 1. We reduced image resolution to square image 120x120 for memory reduction. Since dimensionality is reduced, we increased the images to use per video. Increased the dense layer neurons and learning rate. Reducing the dimensionality slightly affected performance.
5	3D CNN	Training Accuracy - 83.26 % Training Loss - 0.42 Validation Accuracy - 85 % Validation Loss - 0.40	Structure is same as model 1. We further reduced image resolution to square image 100x100 for memory reduction. Since dimensionality is reduced, we further increased the images to use per video. Increased the learning rate. More images to use per video increased the performance to acceptable level.
6	3D CNN	Training Accuracy - 18.10 % Training Loss - 1.60 Validation Accuracy - 26 % Validation Loss - 1.47	Added two convolution layer stack to extract more features. Increased images to use per video and dense neurons. We reduced image resolution to square image 120x120. Model was **underfitting** after adding more convolution layer and performance reduced drastically!
7	2D CNN + LSTM	Training Accuracy - 86.73 % Training Loss - 0.35 Validation Accuracy - 57 % Validation Loss - 1.38	To learn temporal data, we designed 2D CNN + LSTM stack. We reduced image resolution to square image 120x120 for memory reduction. Since dimensionality is reduced, we increased the images to use per video. Chose same number of LSTM cells and dense neurons.

			Clearly the model is **overfitted** and didn't improve validation loss!
8	2D CNN + GRU	Training Accuracy - 95.48 % Training Loss - 0.14 Validation Accuracy - 69 % Validation Loss - 1.09	Structure is same as model 7. We used GRU to reduce trainable parameters. Overfitting was observed, but accuracy of validation dataset improved from model 7.
9	2D CNN + GRU + Augmentation	Training Accuracy - 78.05 % Training Loss - 0.57 Validation Accuracy - 56 % Validation Loss - 1.19	Structure is same as model 8. We added augmentation. Slight cropping of images can make the model unlearn some features. This reduced the overfitting, but validation accuracy didn't improve.
10	MobileNetV2 + GRU + Augmentation	Training Accuracy - 82.2 % Training Loss - 0.5 Validation Accuracy - 83 % Validation Loss - 0.42	A pre-trained network was used to avoid our own 2D CNN design. Since MobileNetV2 have less trainable parameters, we chose this pre-trained network. Since GRU gave better results and less trainable parameters, we chose RNN part from model 9. Acceptable results and accuracy-loss metrics looks good.
11	MobileNetV2 Trainable + GRU + Augmentation	Training Accuracy - 98.94 % Training Loss - 0.03 Validation Accuracy - 99 % Validation Loss - 0.07	Since pre-trained network gave good results, we chose to train the network parameters to see the effect. Training layers of pre-trained network gave 100% accurate results. Hence, we chose this model as final one.

4.1 Conclusion

We have chosen the top 4 models based on performance and analysed the prediction metrics.

- Model 11 gave superb results in prediction! But it has 2,816,581 parameters and model size is 33 MB.
- Model 10 gives decent prediction result! It has 2,816,581 parameters and model size is 15 MB.
- Model 1 and model 5 gives average prediction and model 5 ran faster than model 1.
- Both models 1 and 5 have very less trainable parameters when compared to model 10 and 11.
- Both models 1 and 5 have less accuracy and close to 50 % in prediction.
- Model 1 have 867,589 parameters and model size is 10 MB.
- Model 5 have 589,061 parameters and model size is 6 MB.

- Even though models 1 and 5 have less parameters, it won't be able to identify gestures correctly in real-time.

We will use **model 11** for prediction because it can predict gestures accurately in real-time.
Final chosen model - model-00017-0.03484-0.98944-0.07473-0.99000.h5