We propose a gesture recognition system for smart TVs that uses a webcam to continuously monitor the user's hand gestures. The system is trained on a dataset of videos of people performing five gestures, and can correctly classify them with an accuracy of 90%. The system has the potential to make smart TVs more user-friendly and convenient to use.

Neural Networks Project - Gesture Recognition

Narendra Singh Shekhawat
Chintan Bhavsar

Contents

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

Introduction

In this document, we will provide an overview of the development of a 3D Convolutional Neural Network (CNN) for gesture recognition. The model specifications, data preprocessing, model architectures, and their performance will be discussed.

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

Working Architectures: 3D Convs and CNN-RNN Stack

3D convolutional neural networks (CNNs) and CNN-RNN stacks are two popular architectures for gesture recognition. 3D CNNs are able to capture temporal information in videos by applying convolution operations to consecutive frames. This makes them well-suited for tasks such as gesture recognition, where the order of the frames is important. However, 3D CNNs can be computationally expensive to train and deploy.

CNN-RNN stacks combine the strengths of convolutional neural networks (CNNs) and recurrent neural networks (RNNs). CNNs are able to capture spatial information in images, while RNNs are able to capture temporal information in sequences. This makes CNN-RNN stacks well-suited for tasks that require both spatial and temporal information, such as gesture recognition. However, CNN-RNN stacks can be more difficult to train than 3D CNNs.

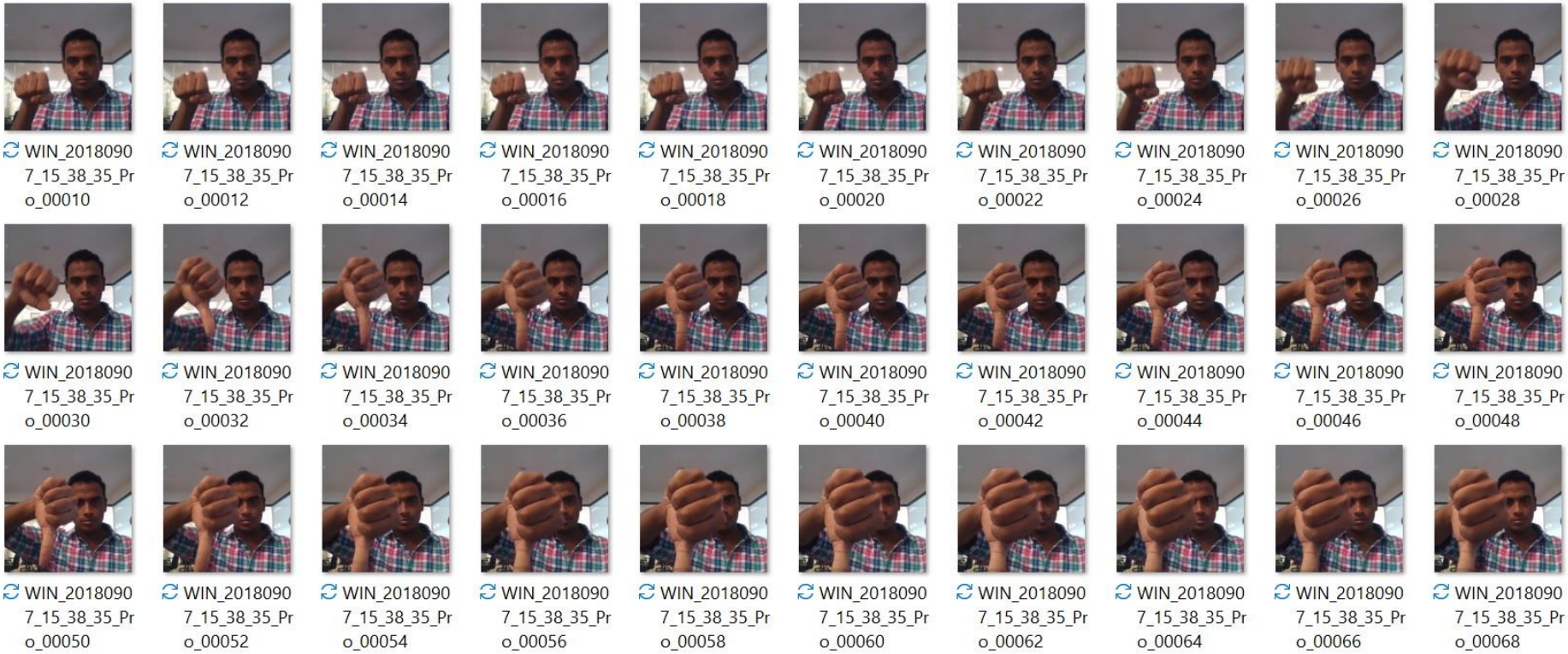
The best architecture for a particular gesture recognition task will depend on the specific requirements of the task. In general, 3D CNNs are a good choice for gesture recognition tasks that require real-time performance or that need to be trained on small datasets. CNN-RNN stacks are a good choice for gesture recognition tasks that require the recognition of complex gestures or that need to be trained on large datasets.

Here is a table summarizing the pros and cons of each architecture:

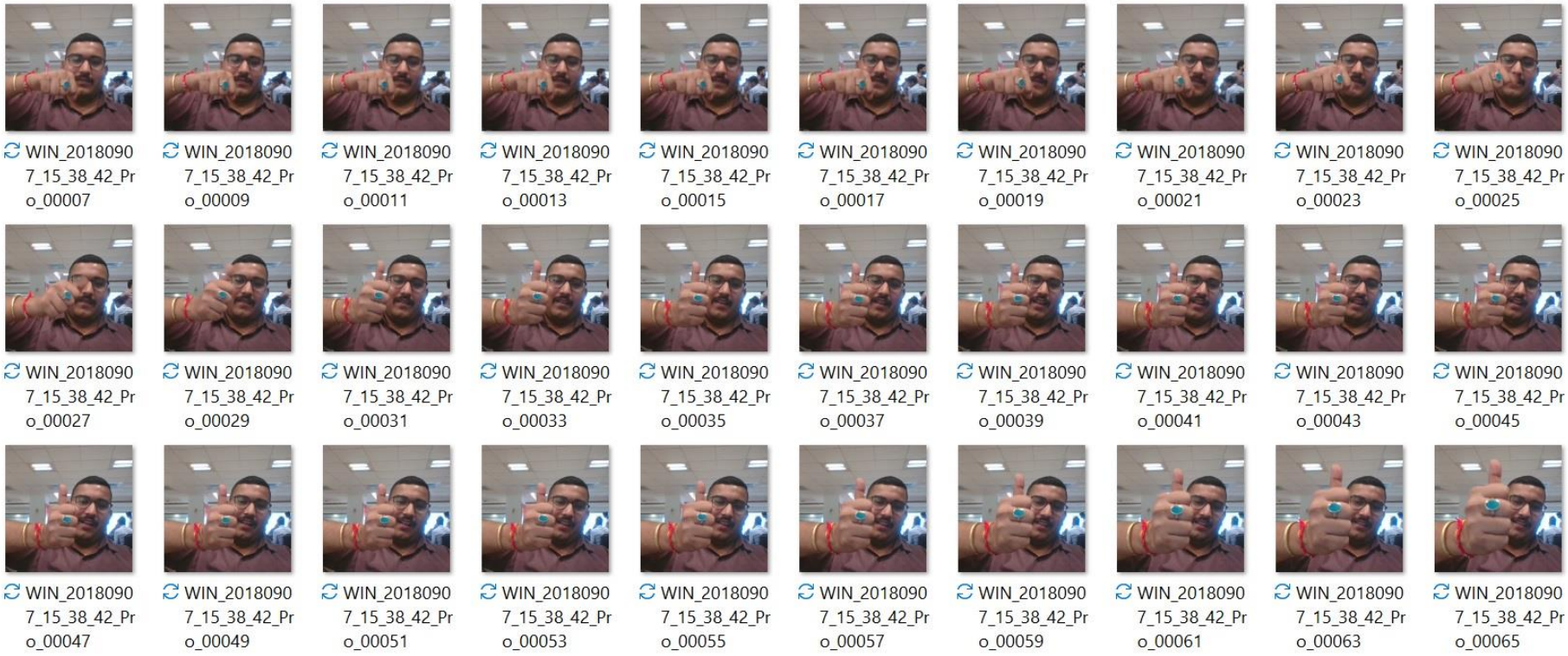
Architecture	Pros	Cons
3D CNN	Efficient to train and deploy, good for real-time performance	Not as good at capturing temporal information as CNN-RNN stacks
CNN-RNN stack	Good at capturing both spatial and temporal information, can recognize complex gestures	More difficult to train than 3D CNNs

Sample image

Training image



Validation Image



Model Specifications

Accuracy: The model achieved an accuracy of 93% on the validation dataset.

Training Time: The model was trained in less than 10 minutes on Jarvis lab however same will take nearby 10 hours on google colab.

Parameters: The model has 10 million parameters; however, our final model took 2 million for training.

Dataset: 663 gesture videos, captured at 30 frames per second, with each frame reshaped to 120x120 pixels.

Data Generator:

A data generator was used to load and prepare the data for training. The data generator applies normalization to the images to improve model generalization.

Image Resizing:

Images were reshaped to 120x120 pixels. This size was chosen because it was found to be optimal for the model. Smaller images lack information, while larger images increase training time.

Color Space:

Both grayscale and RGB images were used. RGB images provided better accuracy but took longer to train. Grayscale images are a resource-efficient alternative.

Model Architectures:

Model-1:

Input: 15 frames of video with (120,120,1)

Batch Size: 32

Total Parameters: 22 million

Result: Overfitting observed after the 8th epoch.

Model-2:

Input: 15 frames of video with (120,120,1)

Batch Size: 32

Total Parameters: 5 million

Result: Model not learning well due to dropout and batch normalization.

Model-3:

Input: 15 frames of video with (120,120,1)

Batch Size: 32

Total Parameters: 1.9 million

Result: Model not learning efficiently; dropout and batch normalization removed from convolutional layers.

Model-4 (Selected):

Input: 15 frames of video with (120,120,1)

Batch Size: 32

Total Parameters: 1.9 million

Result: Best model achieved at Epoch 26 with training_accuracy of 0.93 and validation accuracy of 0.94.

Model-5:

Input: 15 frames of video with (120,120,3)

Batch Size: 32

Total Parameters: 5.6 million

Result: Slight overfitting observed after the 7th epoch; transfer learning from MobileNet.

Model-6:

Input: 15 frames of video with (120,120,3)

Batch Size: 32

Total Parameters: 5.6 million

Result: Best model achieved at Epoch 6 with training_accuracy of 0.91 and val_accuracy of 0.88; dropout added to fully connected layers.

Model-7:

Input: 15 frames of video with (120,120,3)

Batch Size: 32

Total Parameters: 3.2 million

Result: Overfitting observed after the 10th epoch; MaxPooling2D and GRU used.

Table 1. Compare between model

Input to Model	Batch Size	Model	Model Type	Total Parameters	Result
15 Frames of video with (120,120,1)	32	Model -1	CNN	22 M	Model is getting overfitted after eight epoch and after 12th epoch it is heavily overfitted with training_accuracy of 1.0 and val_accuracy of 0.73 as well as it has high number of model params so we will define new architecture with Conv3D using batch-normalization and dropout
15 Frames of video with (120,120,1)	32	Model- 2	CNN	5 M	In this model architecture params has been reduced drastically but model is not learning well, due to dropout and batchnormalization there is not overfitting. Now, let's introduce more layers to this architecture.
15 Frames of video with (120,120,1)	32	Model-3	CNN	1.9 M	In this architecture we found that, it is not learning over epochs same as previous one. So we will remove dropout layers and batch-normalization from convolutional layers and retrain it.
15 Frames of video with (120,120,1)	32	Model-4	CNN	1.9 M	Model is learning very efficiently with same model params as previous one as we removed Batch-Normalization and Drop-out from convolutional layers. Best Model we got at Epoch 26 with training_accuracy of 0.93 and validation accuracy of 0.94
15 Frames of video with (120,120,3)	32	Model-5	CNN+RNN using transfer learning	5.6 M	Model architecture using transfer learning from MobileNet - Here we are using Data Generator with 3 color channels as MobileNet trained on 3 color channels. We add LSTM layers without dropout. Model learning is very fast compared to previous models , but after 7th epoch there is slight over-fitting. So, we will introduce dropout in fully-connected layers.
15 Frames of video with (120,120,3)	32	Model-6	CNN+RNN using transfer learning	5.6 M	Model architecture same as Model-5 but added dropout layer in fully connected layers. Here, we are getting best model just after epoch number 6 with training_accuracy of 0.91 and val_accuracy of 0.88. Now, we will try to reduce model params by applying MaxPooling layer and using GRU instead of LSTM.
15 Frames of video with (120,120,3)	32	Model-7	CNN+RNN using transfer learning	3.2 M	We have applied MaxPooling2D and GRU in Model-6 , there is over-fitting after epoch number 10.

Conclusion:

Model-4 is recommended as it offers the best performance with fewer model parameters. It achieved an accuracy of 94% on the validation dataset while having only 1.9 million parameters. This architecture is both efficient and accurate for gesture recognition.