GESTURE RECOGNITION USING NEURAL NETWORKS

**Submitted by:**

* Harsh Agrawal
* Chandana Deeksha Pulavarthi
* Caroline Abraham

**Contents**

[**Introduction 2**](#_gjdgxs)

[**Understanding the dataset 2**](#_30j0zll)

[**Architectures for Video Analysis 3**](#_lfvi7pezc887)

[CNN + RNN architecture 3](#_511cfi6brdqk)

[3D Convolutional Neural Networks (Conv3D) 3](#_qjn9m4udw5gr)

[Transfer Learning + RNN Architecture 4](#_1fob9te)

[**Data Generator 4**](#_bn80mmb2hhy7)

[**Neural Network Architecture Selection and Training 5**](#_3znysh7)

[Procedure 5](#_j3g3f0pnks4q)

[Model Hyperparameters 5](#_2et92p0)

[Observation Summary 6](#_rlglwnqucjww)

[**Final Model Architecture 9**](#_r273u5shovb)

# Introduction

The objective of this case study is to develop an advanced feature for a state-of-the-art smart TV that enhances user interaction through gesture recognition. Using the TV's built-in webcam, this feature continuously monitors for five distinct gestures, each of which corresponds to a specific command. This functionality enables users to control the TV effortlessly, eliminating the need for a remote and providing a hands-free, intuitive experience.

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

# Understanding the dataset

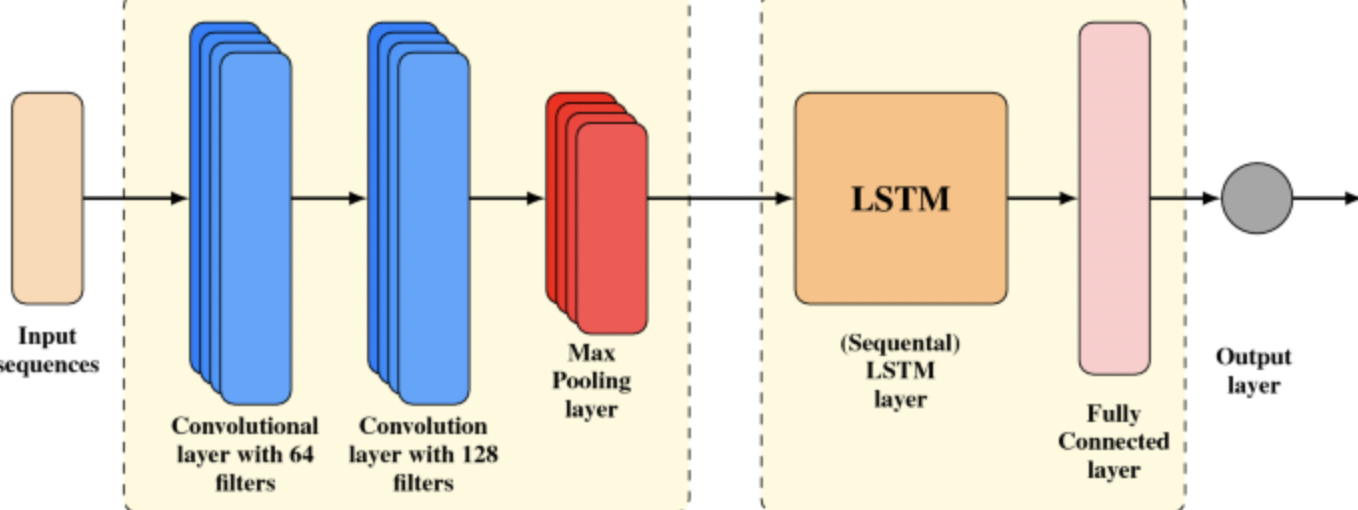
The training data consists of a few hundred videos categorized into one of the five classes. Each video (typically 2-3 seconds long) is divided into a sequence of 30 frames(images). These videos have been recorded by various people performing one of the five gestures in front of a webcam - similar to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

The task involves training various models on the 'train' folder to accurately predict the action performed in each sequence or video, while also ensuring strong performance on the 'val' folder. The final model's performance will be evaluated on the 'val' set, which is withheld for the final assessment.

# Architectures for Video Analysis

## CNN + RNN architecture

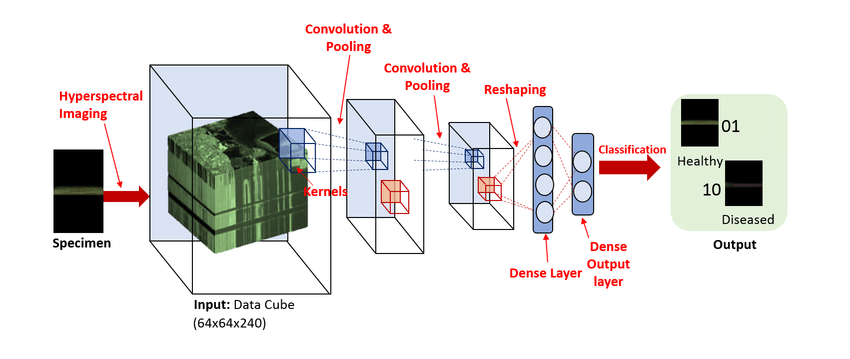
The Conv2D network processes each image in the input sequence independently, extracting a feature vector for each image. These feature vectors form a sequence, which is then fed into an RNN-based network. The RNN processes the sequence of feature vectors and generates an output. In a classification problem, such as the one described, the output of the RNN is passed through a regular softmax layer to produce the final classification probabilities.



**Fig 1: CNN + RNN (LSTM) Architecture**

## 3D Convolutional Neural Networks (Conv3D)

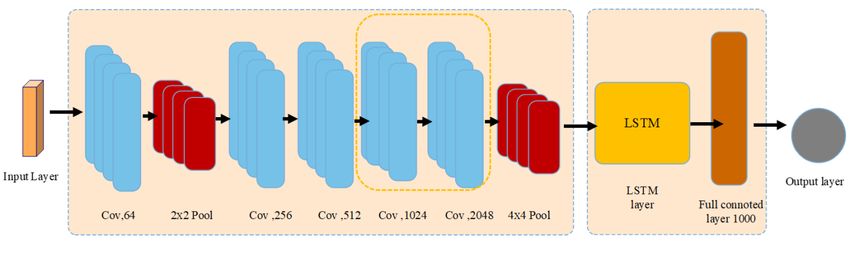
3D convolutions expand the concept of 2D convolutions by introducing a third dimension, enabling filters to move across three axes: x, y, and z. This approach is particularly useful in video analysis, where sequences of frames are treated as 4D tensors. A 3D convolutional filter is represented as (f x f x f) x c, where fff is the filter size and ccc is the number of channels, allowing the filter to capture spatial and temporal information across multiple frames.

****

**Fig 2: Conv 3D Architecture**

## Transfer Learning + RNN Architecture

The idea behind using transfer learning is to leverage learning from models which were trained on vast amounts of data since the computational resources and data availability are constraints for Neural Network training. Pre-trained CNN architectures such as ResNet can be used to extract spatial features from images which are then passed through an RNN to capture temporal dependencies across the frames.



**Fig 3: ResNet50 + RNN (LSTM) Architecture**

# Data Generator

Generator function is used to import batches of data as the training proceeds since having the entire data in memory is impractical while dealing with large datasets such as the current one. Here,

* The sequence of image indices to be considered will be used to determine which frames will finally get used in model training and evaluation.
* Image dimensions are chosen to be $80\times80$ since we have images of two different dimensions in our dataset. Appropriate cropping and resizing is done within this code.
* Normalization of images is done to ensure data is consistent and on the same scale. This helps in the training process for gradient descent.
* Batch labels are converted to one-hot encoding vectors to help with the classification tasks.
* The code is made robust enough to handle the case when the number of training data points isn't divisible by the chosen batch size. This edge case is handled by adding one additional batch containing the remaining data points.
* The generator function finally yields the batch data and the corresponding one-hot encoded label vector which the model can use for training and validation.
* This custom generator function is written since the keras generator function doesn't allow customization for our specific use case.

# Neural Network Architecture Selection and Training

## Procedure

1. **Model Configuration and Hyperparameter Tuning:**

* Various model architectures were configured with different layer counts, incorporating batch normalization and dropout layers in multiple combinations.
* A range of hyperparameters was tested, including different numbers of epochs, batch sizes and learning rates.
* The ReduceLROnPlateau technique was applied to dynamically adjust the learning rate whenever validation loss showed no improvement across epochs.

1. **Optimizer Selection:**

* Models were trained using the Adam optimizer, with a focus on the AMSGrad variant, which demonstrated the best performance.
* Adagrad and Adadelta optimizers were excluded to avoid the increased computational cost and training time required for their dynamic learning rate adjustments.

1. **Overfitting Prevention:**

* The model was monitored for signs of overfitting, such as a noticeable gap between training and validation accuracy.
* Batch Normalization, pooling layers, and dropout layers were incorporated as needed to reduce overfitting.

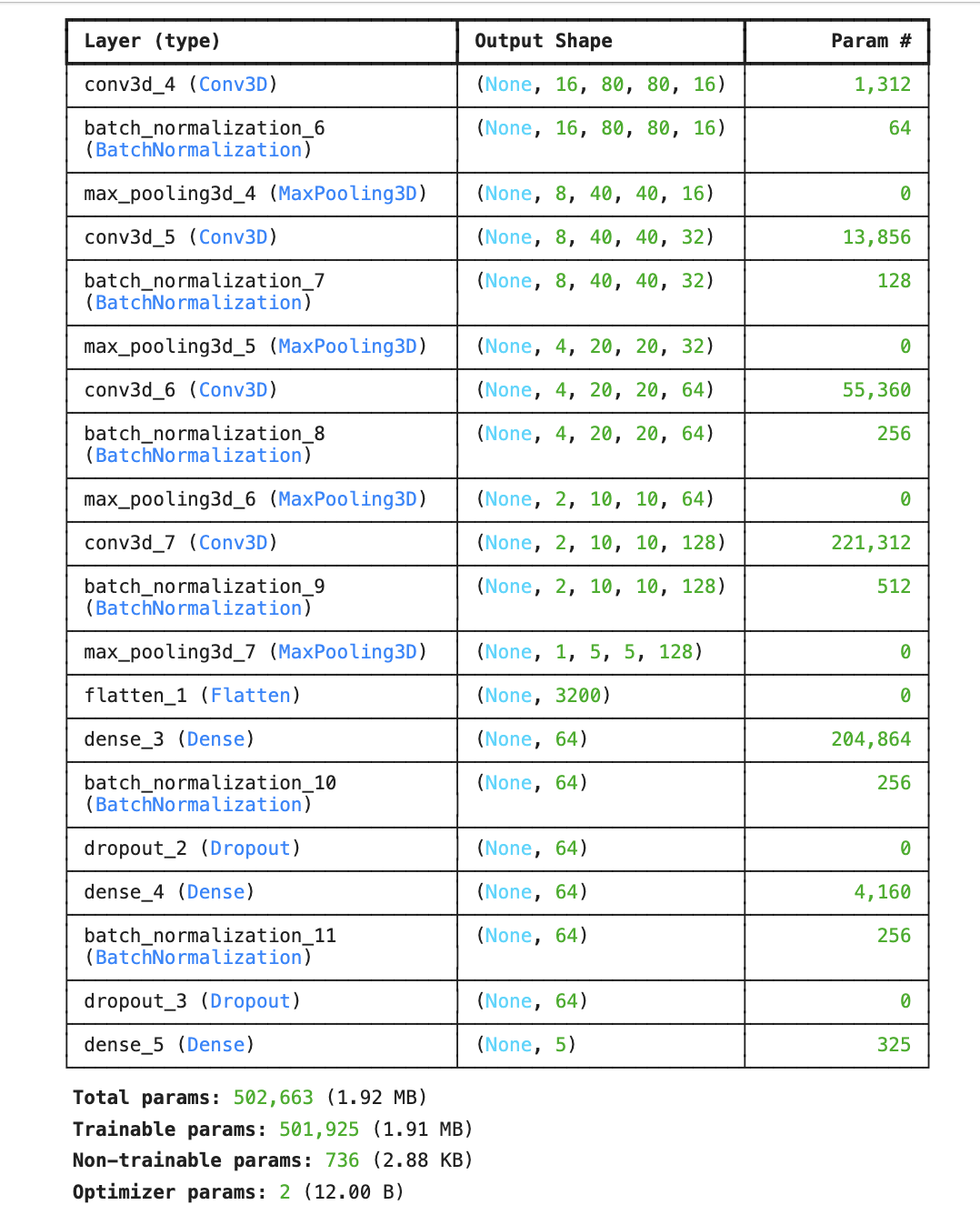
## Model Hyperparameters

* **Trainable Parameters:** It was noted that as the number of trainable parameters increased, the training time of the model also increased. It also leads to overfitting on certain instances.
* **Batch Normalization and Dropout:** Performing Batch Normalization and adding dropout layers to the model improved its generalization, leading to better performance on unseen data.
* **Batch Size and GPU Memory:** The batch size was directly proportional to the available GPU memory and computational resources. Using a large batch size sometimes resulted in GPU Out of Memory errors, necessitating adjustments to determine an optimal batch size that the GPU could support.
* **Trade-off Between Batch Size and Accuracy:** Increasing the batch size significantly reduced training time, but it also had a detrimental effect on the model's accuracy. This trade-off necessitated a decision between shorter training times and higher accuracy by selecting an appropriate batch size.
* **Mitigation of Overfitting:** To mitigate overfitting call-backs were used. These include adaptive learning rate, early stopping, and periodically saving model parameters.
* **Optimal Model Selection:** The experimentation identified a Conv3D model with the Adam optimizer using the AMSGrad algorithm as the best-performing model among those trained. The performance of these architectures depended on various factors, including dataset characteristics, architectural design, and the chosen hyperparameters.
* The loss function used was **categorical cross entropy** and the model evaluation metric used is **Accuracy**.

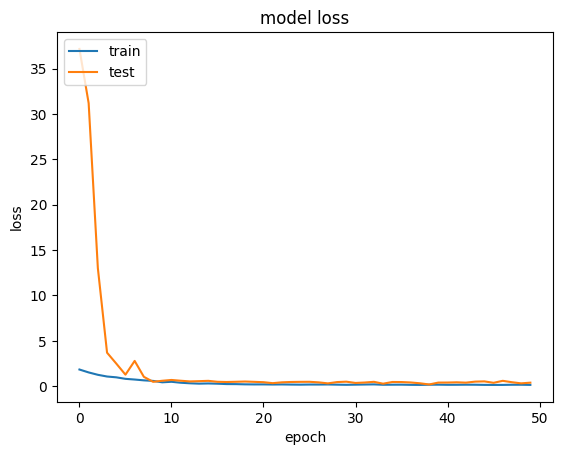
## Observation Summary

| **Exp No** | **Model** | **Evaluation Metric** | **Decision + Explanation** |
| --- | --- | --- | --- |
| **Model: 1** | **Conv3D** | Training Accuracy : 0.19  Validation Accuracy : 0.18 | * Trained the model without Batch Normalization and Dropouts. * Trained only for 5 epochs. Very low training and validation accuracy. |
| **Model: 2** | **Conv3D** | Training Accuracy : 0.23  Validation Accuracy : 0.18 | * Trained the model using Batch Normalization and Dropouts. * However, it showed very low training and validation accuracy. |
| **Model: 3** | **Conv3D** | Training Accuracy : 0.39  Validation Accuracy : 0.36 | * Trained a deeper model with more parameters. Increased the number of epoch to 15 * Training loss decreased from 3.2803 to 1.2796, but validation accuracy improved only slightly from 19% to 36%, indicating potential overfitting. * The learning rate was reduced during training, suggesting a need for better tuning for improved performance. |
| **Model: 4** | **Conv3D** | Training Accuracy : 0.90  Validation Accuracy : 0.77 | * In this model, the Learning rate has been reduced to (learning rate=0.0005). * The Optimization method used is Adam(amsgrad). * Increased the number of Epochs to 20 in this model. * This model has performed better in terms of Training and validation scores. |
| **Model: 5** | **CNN + RNN** | Training Accuracy : 0.19  Validation Accuracy : 0.19 | * Built a simple model with the combination of CNN and RNN. * The model accuracy scores were very low suggesting that the model struggled to generalize effectively from the training data. |
| **Model: 6** | **CNN + GRU** | Training Accuracy : 0.28  Validation Accuracy : 0.27 | * Performed Batch Normalization in this model also added Dropouts. * Over 30 epochs, training accuracy improved to around 33%, with validation accuracy peaking at 35%, indicating some learning progress but overall inconsistency. * Model's performance on the validation set remained relatively low, with the loss reducing only slightly to 1.56, suggesting potential issues with feature extraction or class representation in the dataset. |
| **Model: 7** | **CNN + LSTM** | Training Accuracy : 1.00  Validation Accuracy : 0.86 | * Increased the Number of neurons and also the number of Epochs to 50 in this model. * The model looks like it has Overfitted, it has achieved a training accuracy of 100% but the validation accuracy is only 86%. * The difference between the Training and validation accuracy is also High. |
| **Model: 8** | **ResNet50 + GRU(64)** | Training Accuracy : 0.53  Validation Accuracy : 0.55 | * Experimented with * multiple batch sizes * varying architectures (hidden layers and bidirectional GRUs) * normalization techniques (batch normalization and L2 normalization of weights) * Varying dropout percentage between 30 to 70% * Model works better than the regular CNN+GRU architecture although it is showing higher validation accuracy than training accuracy (random result). |
| **Model: 9** | **ResNet50 + LSTM(64)** | Training Accuracy : 0.85  Validation Accuracy : 0.56 | * Experimented with * multiple batch sizes * varying the number of hidden layers * normalization techniques (batch normalization and L2 normalization of weights) * Varying dropout percentage between 30 to 70% * Model appears to have overfitted despite the attempts to fine tune hyperparameters for sufficient generalization. |
| **Final Model** | **Conv3D (with amsgrad algorithm)** | Training Accuracy : 0.96  Validation Accuracy : 0.91 | * Chose better performing model to build and train the final model. * Number of Epochs = 50 * Performed Batch Normalization and added Dropouts to the model. * Used Adam optimizer with the amsgrad algorithm, which has performed the best of the models that we have trained. * The model exhibited minimal disparity between Training and Validation loss. |

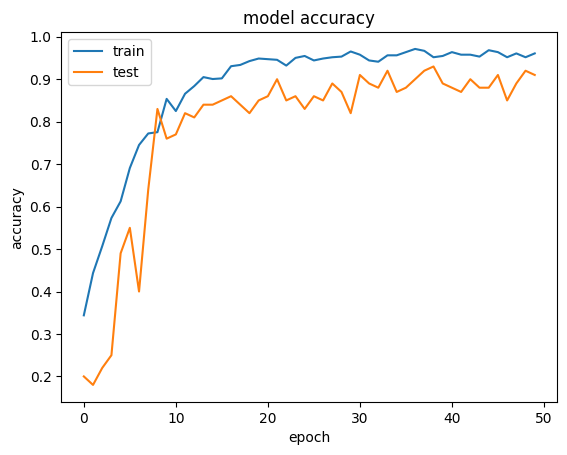
# Final Model Architecture

****

**Fig1: Model Architecture**

****

**Fig2: Model accuracy vs Epoch**

****

**Fig3: Model loss vs Epoch**