Neural Networks Group Case Study: Hand Gesture Recognition Project

**Problem Statement**:

This project involves building a 3D Convolutional Neural Network (CNN) to correctly recognize hand gestures by a user to control a smart TV.

The objective of this projects is to build a hand gesture recognition model that can be hosted on a camera installed in a smart TV that can understand 5 gestures.

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

**About 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 videos have two types of dimensions - either 360x360 or 120x160 (depending on the webcam used to record the videos).

Data Source : <https://drive.google.com/uc?id=1ehyrYBQ5rbQQe6yL4XbLWe3FMvuVUGiL>

**Neural Network Architectures Used:**

For analysing videos using neural networks, two types of architectures are used commonly.

1. **Convolutions + RNN**

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.  
The conv2D network will extract a feature vector for each image, and a sequence of these feature vectors is then fed to an RNN-based network. The output of the RNN is a regular softmax (for a classification problem such as this one).

In place of generic RNN, LSTM and GRU has been used in our experiments.

The image network will just give some feature representation but the LSTM/GRU will be able to decipher the sequence information to classify them as one of the class.

Then dense layer output can be fed in sequence to LSTM/GRU to get the desired output.

An LSTM has 4 gates, while GRU has 3 gates. Using GRU will significantly reduce the training times as it needs to compute values for 3 gates and its performance is at par with the LSTMs.

Another advantage here is we can use the **transfer learning** here, since the state of the art networks are already available for the image classification, we can easily use the already trained weights of resNet or VGGNet and then we can use those networks to transform these images to give the image representation (eg dense layer output).

The dense layer which will be used will be standard models like resnet, VGGnet etc.

1. **3D convolutional network**.

3D convolutions are a natural extension to the 2D convolutions you are already familiar with. Just like in 2D conv, you move the filter in two directions (x and y), in 3D conv, you move the filter 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.

Results of the Experiments done to solve the problem

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model number | Architecture | Description | Result | Observation | Parameters |
| Model 1 | 3D Convolution Model | Batch size = 32 Epochs =20 Number of frames =13  Image size = (80,80) Kernel size = (2,2,2) Activation = relu Dropout = 0.25 optimizer = 'sgd' | Training Accuracy: 81% Validation accuracy: 33% | 1. Training Progress: The model's training accuracy steadily improves, while the validation accuracy stagnates around 40%, indicating potential overfitting or poor generalization.  2. Learning Rate Adjustment: The learning rate decreases multiple times during training due to the ReduceLROnPlateau callback, helping refine the model but also signaling potential plateauing.  3. Validation Loss: The validation loss fluctuates, with no significant improvement despite reducing the learning rate, which suggests that the model is struggling to generalize well to unseen data. | 1733509 |
| Model 2 | 3D Convolution Model | Batch size = 32 Epochs =20 Number of frames =21  Image size = (80,80) Kernel size = (2,2,2) Activation = relu Dropout = 0.25 optimizer = 'Adam' | Training Accuracy: 96% Validation accuracy: 35% | 1. The gap between training accuracy and validation accuracy is wide throughout the training. By epoch 20, the training accuracy reaches ~0.95, but the validation accuracy is still at 0.32.  2. This large gap indicates overfitting: The model is performing very well on the training data but not generalizing effectively to unseen data (validation set).  3. Changing the optimizer did not lead to much improvement | 3371909 |
| Model 3 | 3D Convolution Model | Batch size = 32 Epochs =20 Number of frames =21  Image size = (120,120) Kernel size = (2,2,2) Activation = relu Dropout = 0.25 optimizer = 'Adam' | Training Accuracy: 96% Validation accuracy: 35% | 1. Increasing the image size did not help and neither did adding more frames. Next let us try adding regularization and increasing drop out. There was only a slight increase in performance when image size increased | 7467909 |
| Model 4 | 3D Convolution Model | Batch size = 32 Epochs =20 Number of frames =21  Image size = (120,120) Kernel size = (3,3,3) Activation = relu Dropout = 0.5 for first 2 layers, 0.3 for last layer optimizer = 'Adam' Regularization: L2 | Training Accuracy: 78% Validation accuracy: 30% | 1. Still overfitting is present, but there is some slight improvement | 15069317 |
| Model 5 | 3D Convolution Model | Batch size = 32 Epochs =20 Number of frames =30  Image size = (84,84) Kernel size = (3,3,3) Activation = relu Dropout = 0.5 for first 2 layers, 0.3 for last layer optimizer = 'Adam' Regularization: L2 | Training Accuracy: 73% Validation accuracy: 36% | 1. No significant improvements observed due to having increased number of frames | 10154117 |
| Model 6 | 3D Convolution Model | Batch size = 64 Epochs =30 Number of frames =30  Image size = (84,84) Kernel size = (3,3,3) Activation = relu Dropout = 0.5 for first 2 layers, 0.3 for last layer optimizer = 'Adam' Regularization: L2 | Training Accuracy: 68% Validation accuracy: 25% | 1. With larger batch size its observed that training time reduces.  2. Even with higher number of epochs we dont see a significant improvement in the model.  3. In the above experiments we observed that with 3D convolution network we are not able to get a good model. | 10154117 |
| Model 7 | CNN + LSTM | Batch size = 32 Epochs =20 Number of frames =30  Image size = (84,84) Kernel size = (3,3,3) Activation = relu Dropout = 0.5 for first 2 layers, 0.25 for last layer optimizer = 'Adam' Regularization: L2 | Training Accuracy: 55% Validation accuracy: 30% | 1. The highest validation accuracy is 0.30 (achieved in epoch 18), which is relatively low, and it hasn't shown significant improvement as the epochs progress. 2. For comparison, the training accuracy has been somewhat increasing, but it's still below 0.55 by the final epoch. A performance gap between training and validation accuracy indicates the model might not have learned the underlying patterns of the data well enough. 3. This could be a sign of underfitting 4. Also the model is not generalizing well as validation loss remains high | 6714117 |
| Model 8 | CNN + GRU | Batch size = 32 Epochs =20 Number of frames =30  Image size = (84,84) Kernel size = (3,3,3) Activation = relu Dropout = 0.5 for first 2 layers, 0.25 for last layer optimizer = 'Adam' Regularization: L2 | Training Accuracy: 96% Validation accuracy: 49% | 1. Looks like the Model is underfitting:  2. Training accuracy steadily improves and reaches 96.53% by epoch 20. Validation accuracy, however, lags behind significantly, reaching only 49.00% at epoch 20.  3. If the model were overfitting, you would expect the validation accuracy to increase as training accuracy improves, but instead, it remains quite low.  4. Training loss consistently decreases (good sign of learning), but validation loss fluctuates and stays higher than the training loss, which is typical of underfitting. If the model were overfitting, the validation loss would eventually plateau and increase, but here, it just doesn't improve as much as the training loss. | 2573925 |
| Model 9 | CNN + GRU + Data Augmentation | Batch size = 32 Epochs =20 Number of frames =30  Image size = (84,84) Kernel size = (3,3,3) Activation = relu Dropout = 0.5 for first 2 layers, 0.25 for last layer optimizer = 'Adam' Regularization: L2 Custom data augmentor | Training Accuracy: 96% Validation accuracy: 22% | Model appears to be overfitting, as indicated by the growing gap between training and validation performance. | 2573925 |
| Model 10 | Transfer Learning with LSTM | Batch size = 32 Epochs =20 Number of frames =18  Image size = (120,120) Kernel size = (3,3,3) Activation = relu Dropout = 0.5 for first 2 layers, 0.25 for last layer optimizer = 'Adam' Regularization: L2 Base network: MobileNet Base Network Weights: not trainable | Training Accuracy: 100% Validation accuracy: 79% | 1. Training Accuracy is excellent, nearing 100%. 2. Validation Accuracy is plateauing and fluctuating, indicating overfitting. 3. The model is performing well on the training data but is struggling to generalize effectively to the validation set, likely due to overfitting. | 3840453 |
| Model 11 | Transfer Learning with GRU | Batch size = 32 Epochs =20 Number of frames =18  Image size = (120,120) Kernel size = (3,3,3) Activation = relu Dropout = 0.5 for first 2 layers, 0.25 for last layer optimizer = 'Adam' Regularization: L2 Base network: MobileNet Base Network Weights: trainable | Training Accuracy: 100% Validation accuracy: 93% | The model is performing very well.   Loss & Accuracy Trends: 1. Training Loss: The training loss decreases from 1.2437 at epoch 1 to 0.0040 at epoch 20. This shows rapid improvement in the model's ability to minimize error. 2. Training Accuracy: Training accuracy starts at 49.32% at epoch 1 and rises to 100% by epoch 20, which is a good indicator of the model improving its performance. 3. Validation Loss: The validation loss fluctuates, starting at 1.3791 and dropping to 0.1571 by epoch 20. This suggests good generalization but with some variability (more evident in the first half of training). 4. Validation Accuracy: The validation accuracy starts at 37% and rises steadily, reaching 93% by epoch 20, indicating good performance on the validation set.  This model with Transfer learning LSTM with training mobilenet weights seems to offer the best performance. Hence this model is finalized. | 3840453 |

**Final Model Selected:**

The model is getting saved for all the experiment for each epoch in the form of .h5 file in the disk. The final model chosen is model-00020-0.00397-1.00000-0.15710-0.93000.h5

This model is based on the transfer learning with GRU and re-training the weights. The pre-trained model of MobileNet is used for this purpose with weights of ImageNet.

Since our dataset is kind of subset of the ImageNet dataset, then we are going to transfer the knowledge of this model onto our datasets.

In Keras, MobileNet resides in the applications module. Keras offers out of the box image classification using MobileNet if the category you want to predict is available in the [ImageNet categories](https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a).

Now since the ImageNet dataset does not have exact dataset available as per our problem statement, we have not feezed the layers that we are using for our final model and we have re-trained the weights in our transfer learning.

‘Fine Tuning’, generally, is when we freeze the weights of all the layers of the pre-trained neural networks (on dataset A [e.g. ImageNet]) except the penultimate layer and train the neural network on dataset B [e.g. Fashion-MNIST], just to learn the representations on the penultimate layer. We usually replace the last (softmax) layer with another one of our choice

In other words, we are not using the Fine Tuning as fine-tuning is used when the dataset used to train the pre-trained model is very similar to or the same as the new dataset which is not in our case.

A graph of a training and validation loss

Description automatically generated

**We were able to get the excellent validation accuracy of 93 % for this selected model built using the transfer learning of MobileNet without freezing/fine-tuning.**