Hand Gesture Recognition for Smart TV Control

Problem Overview:

The goal of this project is to build a 3D Convolutional Neural Network (CNN) capable of recognizing hand gestures for controlling a smart TV. The system should respond to five specific gestures captured by a webcam mounted on the TV:

- **Thumbs Up:** Increase the volume
- **Thumbs Down:** Decrease the volume
- **Left Swipe:** Rewind by 10 seconds
- **Right Swipe:** Fast forward by 10 seconds
- **Stop Gesture:** Pause the video

The system continuously monitors these gestures and executes corresponding actions based on real-time input.

Dataset Details:

The dataset consists of several hundred videos, each representing one of the five gestures. Each video is typically 2-3 seconds long, divided into a sequence of 30 frames, and recorded by various individuals in front of a webcam. This data is recorded with different webcam resolutions, either 360x360 or 120x160, depending on the equipment used.

Neural Network Architectures:

Two main types of neural network architectures are often used for analyzing video data:

1. **CNN + RNN (Recurrent Neural Networks):**
    - In this architecture, each frame of the video is passed through a Convolutional Neural Network (CNN) to extract features. The sequence of extracted features is then processed using an RNN, such as LSTM (Long Short-Term Memory) or GRU (Gated Recurrent Units). This allows the model to capture the temporal dynamics of the gesture sequence and classify them accordingly.
    - The advantage of using GRU over LSTM is that GRUs have fewer gates, leading to faster training times while maintaining comparable performance. Additionally, this architecture benefits from **transfer learning**, where pretrained models like ResNet or VGGNet are used to extract image features, which are then processed by the RNN layers.
2. **3D Convolutional Networks:**
    - The alternative method is to extend traditional 2D convolutions into 3D convolutions. In a 3D CNN, a filter moves across the spatial dimensions (x, y) and the temporal dimension (z), effectively analyzing video frames as a 3D structure (height, width, time).
    - The input to a 3D CNN consists of a sequence of frames (e.g., 30 frames of size 100x100x3), forming a 4D tensor. This type of architecture is particularly well-

suited for processing videos, as it can capture both spatial and temporal features simultaneously.

Data Pipeline and Custom Data Generators:

A crucial component in training deep learning models is data ingestion. Given the size and complexity of video data, a custom data generator is essential. Although frameworks like Keras offer built-in image data generators, they may not be sufficient when handling video data.

The custom generator in this project is implemented using Python's yield function, which allows us to load batches of videos into memory efficiently. This batch-wise approach is essential for training large models, as it minimizes memory usage while allowing for effective gradient descent.

Custom data generators offer several advantages:

- **Memory Efficiency:** Generators load one batch of data at a time, so they consume less memory compared to loading the entire dataset at once.
- **Performance:** The lazy evaluation mechanism allows for faster data loading and processing, which is critical when working with large datasets.
- **Flexibility:** Custom generators can handle various types of data, including images, text, and audio, ensuring compatibility with different input formats.

This generator facilitates efficient training and prevents memory overload, especially when working with very large datasets that cannot fit into memory in their entirety.

Experiments Conducted:

During the model training process, it's common to encounter situations where the validation loss does not decrease even after several epochs, or it may even increase. This indicates that the model's gradient updates have plateaued, and the optimization process is no longer converging towards the minimum.

When this happens, the learning rate is typically reduced. A high learning rate may cause the optimization process to oscillate around the minimum without making significant progress. In our experiments, we reduced the initial learning rate to 0.0002 for the Adam optimizer wherever necessary to ensure smoother convergence.

The attached file provides detailed information for each model, including the model type, number of images, image size, number of parameters, batch size, number of epochs, training time, model performance results, and the reasoning behind each model's decision.

Final Model Selection:

For each experiment, the model was saved at every epoch in the form of a .h5 file. The final model chosen was model-00020-0.03254-0.98793-0.15436-0.95000.h5.

This model was based on transfer learning using GRU and re-training the weights. We utilized the pre-trained MobileNet model with ImageNet weights for this purpose.

Given that our dataset is a subset of ImageNet, we leveraged the knowledge learned from ImageNet and applied it to our dataset. In Keras, MobileNet can be accessed via the applications module, which provides out-of-the-box image classification using MobileNet when the categories align with those in ImageNet.

However, since the categories in ImageNet do not perfectly match our problem, we did not freeze the layers in our final model. Instead, we retrained the weights in the transfer learning process.

Fine-tuning typically involves freezing the weights of the pre-trained layers (from dataset A, like ImageNet) except for the penultimate layer, and retraining the model on a new dataset (dataset B, like Fashion-MNIST). In this process, only the penultimate layer is adapted to learn the new task. We did not apply fine-tuning in our case because our dataset differs significantly from ImageNet.

Through this approach, we achieved excellent validation accuracy of 95% with the model trained using transfer learning from MobileNet, without fine-tuning the layers.