Technical Documentation

Project Overview

The goal is to develop a gesture recognition feature for smart TVs, allowing users to control the TV without a remote using five specific gestures. Each gesture corresponds to a command such as increasing or decreasing the volume, jumping forward or backward in a video, or pausing the video.

Dataset Description

The dataset consists of videos categorized into five gesture classes, with each video divided into 30 frames. The data is organized into train and val folders, each containing subfolders for individual videos. Each subfolder contains 30 frames of a gesture. The CSV files (train.csv and val.csv) provide information on the video subfolders, gesture names, and numeric labels.

Data Preprocessing

Preprocessing involves:

1. Loading and shuffling the data.
2. Cropping images to make them square.
3. Resizing images to a standard dimension.
4. Normalizing pixel values.
5. Optionally augmenting images by applying random transformations like shifts and crops.

DataLoader Class

Purpose: Load and shuffle training and validation data from CSV files.

Methods:

__init__(): Initializes the class.

load_data(data_dir): Loads and shuffles data from the specified directory.

get_data(data_dir): Returns the shuffled training and validation data.

DataGenerator Class

Purpose: Generate batches of preprocessed images and labels for training and validation.

Parameters: Includes paths, batch size, sequence length, image dimensions, number of labels, and various flags for debugging and augmentation.

Methods:

__init__(): Initializes the class with specified parameters.

process_images(folder_name, img_indices): Processes and optionally augments images from a given folder.

generator(): Generates batches of image sequences and their corresponding labels, optionally performing ablation (stratified sampling).

ModelManager Class

Purpose: Build, compile, and train the gesture recognition model.

Methods:

__init__(output_path, input_shape, num_labels): Initializes the class with model output path, input shape, and number of labels.

build(model_name, learning_rate): Dynamically imports the model architecture and builds the model.

train(train_generator, val_generator, num_train_steps, num_val_steps, num_epochs, ...): Trains the model using the specified parameters and callbacks for learning rate scheduling and checkpointing.

Trainer Class

Purpose: Manage the entire training process, including data preparation, model building, and training.

Parameters: Includes batch size, sequence length, image dimensions, output path, model name, learning rate, number of epochs, and various flags for debugging and augmentation.

Methods:

__init__(...): Initializes the class with specified parameters.

get_docs(): Loads and returns training and validation documents.

get_num_labels(train_doc, val_doc): Determines the number of unique gesture labels.

get_num_steps(train_doc, val_doc): Calculates the number of training and validation steps per epoch.

build(): Builds the model using ModelManager.

train(): Trains the model using DataGenerator for generating batches and ModelManager for managing the training process.

plot_training_history(): Plots the training and validation accuracy and loss over epochs.

CustomLearningRateScheduler Class

Purpose: Custom callback to adjust the learning rate during training.

Methods:

__init__(...): Initializes the class with parameters for learning rate adjustment.

on_epoch_begin(epoch, logs): Adjusts the learning rate at the beginning of each epoch based on the specified schedule.

Model Architecture Files (model_*.py)

Each model_*.py file defines a different model architecture using the build_model function. The ModelManager class dynamically loads these models based on the provided model name.

Common Components:

Imports: Necessary TensorFlow/Keras modules for building the model.

build_model(input_shape, num_labels, learning_rate):

Defines the network architecture.

Compiles the model with an optimizer and loss function.

Returns the compiled model.

Example: model_15.py:

1. Uses MobileNet for feature extraction from each frame (2D CNN).
2. Processes the sequence of feature vectors using a GRU (Gated Recurrent Unit).
3. Includes additional layers for batch normalization, pooling, flattening, dense layers, and dropout for regularization.

Ablation

Ablation in the code is used to reduce the dataset size for experimental purposes by performing **stratified sampling**. This helps to observe the effects of training on a smaller subset while maintaining the distribution of gesture labels, ensuring the smaller dataset remains representative of the original dataset.

Experiment Summary

Experiment Number	Model	Epochs	Result	Decision + Explanation
1	**Conv3D (ablation run)**	**20**	**OOM Error**	**Reduced the batch size to a smaller value (20)**
2	**Conv3D (ablation run)**	**20**	**Training accuracy:0.4 Validation accuracy:0.1**	**1. Model is clearly not learning anything because of the small ablation size. 2. Increasing ablation size to almost half the samples (300)**
3	**Conv3D(ablation run)**	**20**	**Training Accuracy: 1 Validation Accuracy:0.15**	**1. Model is learning/overfitting, but it's struggling to generalize at all as evident from the validation accuracy.**

				2. Adding more convolutional layers to see if it helps to generalize
4	**Conv3D(ablation run)**	**10**	**Training Accuracy: 0.69 Validation Accuracy: 0.21**	**1. Boosting the model complexity didn't solve the generalization problem. 2. Let's arrange the randomly selected video frames in sequential order to maintain the temporal aspect and test the impact.**
5	**Conv3D(ablation run)**	**10**	**Training Accuracy: 0.72 Validation Accuracy: 0.21**	**1. Seeing some improvment, but overfitting issue persists 2. Let's run for 20 epochs with the entire dataset**
6	**Conv3D**	**20**	**Training Accuracy: 0.72 Validation Accuracy: 0.21**	**Significant improvement is noticeable from 15 epochs, but the model has some drawbacks: • While validation loss has decreased, it fluctuates significantly between 0.5 and 0.8 as epochs progress. • Validation loss doesn't decrease for a long time . Let's implement a custom learning rate scheduler to adjust the learning rate every 3 epochs.**
7	**Conv3D**	**20**	**Training Accuracy: 0.87 Validation Accuracy: 0.72**	**We noticed substantial improvement after adding the learning rate scheduler: 1. Validation accuracy is stable in the later epochs. 2. Validation loss consistently decreases after 15 epochs. 3. Overfitting has been reduced, but it still exists. Let's try the following: 1. Add kernel regularization with values of 0.01 and 0.1 to minimize overfitting. 2. Include all data. 3. Run for 30 epochs.**

8	**Conv3D**	**30**	**Training Accuracy: 0.91 Validation Accuracy: 0.83**	**It's working well, but there's still some degree of overfitting. Increasing l2 norm for both dense layers to 0.1**
9	**Conv3D**	**30**	**Training Accuracy: 0.96 Validation Accuracy: 0.88**	**We got much better accuracy this time but overfitting still exists, and validation accuracy is still under .9 Increasing kernel regularizers to 0.15**
10	**Conv3D**	**30**	**Training Accuracy: 0.94 Validation Accuracy: 0.79**	**Increasing kernel regularizer couldn't boost the model performance. Rather It increased the overfitting. Let's play with kernel initializers, setting kernel initializers to he_normal**
11	**Conv3D**	**30**	**Training Accuracy: 0.95 Validation Accuracy: 0.85**	**Although we got better validation accuracy 2 models back, unlike that model, this model showed consistent increase in accuracy for last few epochs. This seems to be a good model, although still a bit overfitting. Let's see if we can handle the overfitting problem in it by adding more dropouts. Added one more dropout layer.**
12	**Conv3D**	**30**	**Training Accuracy: 0.95 Validation Accuracy: 0.55 (at 24th epoch)**	**jarvislabs got auto-paused during the training, but from the output it's evident that dropouts are not helping let's try playing with the temporal dimension Changing MaxPooling3D(pool_size=(2, 2, 2)) to MaxPooling3D(pool_size=(1, 2, 2))**
13	**Conv3D**	**30**	**Training Accuracy: 0.80**	**As evident from the graph, with the change in maxpooling's temporal dimension, overfitting problem is**

			Validation Accuracy: 0.86	**largely reduced, although training accuracy has dropped slightly.** **Let's run it for 50 epochs**
14	**Conv3D**	**50**	**Training Accuracy: 0.96** **Validation Accuracy: 0.91**	**Awesome, we have a great model this time! We achieved 0.96 training accuracy and 0.91 validation accuracy. Reducing the temporal dimension in the MaxPooling3D layer really worked.** **let's see how the model behaves if we replace flatten() layer with a GlobalAveragePooling3D layer.** **This time we are retaining the temporal dimension change of one MaxPooling3d layer, but adding back the temporal dimension of other MaxPooling3d layers. This will increase the number of parameters again, but it will still be much lesser than the earlier models.** **Let's run it for 50 epochs**
15	**Conv3D**	**30**	**Training Accuracy: 0.85** **Validation Accuracy: 0.85**	1. **Also, this model doesn't show much overfitting in the high epochs.** 2. **Model parameters are still significantly lesser than earlier models.** **Let's try it for 50 epochs**
16	**Conv3D**	**50**	**Training Accuracy: 0.93** **Validation Accuracy: 0.91**	**Our best conv3d model so far.** **We're wrapping up our experiments with Conv3D models. Next, we'll try CNN+RNN.**
17	**CNN+LSTM(ablation run)**	**30**	**Training Accuracy: 0.38** **Validation Accuracy: 0.38**	**Too much of oscillation. Let's try SGD**
18	**CNN+LSTM(ablation run)**	**20**	**Training Accuracy: 0.62**	**We observe reduced oscillation and improved validation accuracy with SGD and momentum.**

			Validation Accuracy: 0.52	**However, the number of parameters is excessively large. To address this, let's attempt to decrease the units in the max-pooling layer, as gesture movements might not require such a high number of units.**
19	**CNN+LSTM(ablation run)**	**20**	**Training Accuracy: 0.61 Validation Accuracy: 0.60**	**As evident from the result above, even with 300 samples (less than half the total samples) and 20 epochs, we are able to achieve 61% traiining and 60% validation accuracy. There's no overfitting at all. That's pretty good.** **Let's pass the entire dataset and try it for 50 epochs.**
20	**CNN+LSTM**	**50**	**Training Accuracy: 0.99 Validation Accuracy: 0.80**	**As evident from the above result, the model is clearly overfittig in the higher epochs** **Let's try GRU**
21	**CNN+GRU(ablation run)**	**20**	**Training Accuracy: 0.72 Validation Accuracy: 0.68**	**Promosing model** **Let's pass the entire dataset**
22	**CNN+GRU**	**20**	**Training Accuracy: 0.93 Validation Accuracy: 0.78**	**Model is overfitting at higher epochs** **Increasing dropout from 0.5 to 0.6 adding kernel regularizer (0.1)**
23	**CNN+GRU**	**20**	**Training Accuracy: 0.77 Validation Accuracy: 0.71**	**1. Model is struggling to get good validation accuracy at higher epochs** **2. Training accuracy Dropped** **3. Seems like model is learning too fast in the initial epochs** **Hence,** **1. Decreasing dropouts back to 0.5** **2. Decreasing kernel regularizer to 0.01** **3. Reducing Learning rate to 0.001**

24	**CNN+GRU**	**20**	**Training Accuracy: 0.66 Validation Accuracy: 0.68**	**1. Although lower learning rate slowed down the learning, overall performanced is better now. 2. Overfitting issue is also resolved Let's run it for 50 epochs**
25	**CNN+GRU**	**50**	**Training Accuracy: 0.85 Validation Accuracy: 0.77**	**jarvislabs got auto-paused again. but we can see the best result from the o/p . It's showing promise in lower epochs, but a bit stagnating in higher epochs. Let's use transfer learning with GRU**
26	**CNN+GRU+Transfer Learning (SGD optimizer)**	**20**	**Training Accuracy: 0.99 Validation Accuracy: 0.90**	**Accuracy is significantly up. Let's try Adam optimizer**
27	**CNN+GRU+Transfer Learning (Adam optimizer)**	**20**	**Training Accuracy: 0.99 Validation Accuracy: 0.98**	**Awesome performance! This is the best model so far!**