Adaptive CNN Gesture Recognition

Subhasis Biswas

January 28, 2024

Abstract

Gesture Recognition with Evolving Training Data presents a real-time hand gesture recognition system that uses deep learning for computer vision. Utilizing Convolutional Neural Networks (CNNs), the project deals with real-time processing, achieving a practical accuracy in classifying gestures ranging from null activity to numbers 1 to 5. The system is also optimized for efficient recording of new data and retraining of the model, ensuring a prompt and responsive training process simultaneously keeping a track on the entirety of the data that the model is trained upon.
Keywords: *Hand Gesture Recognition, Computer Vision, Machine Learning, Webcam, CNN, Human-Computer Interaction, Null Activity, Gestures 1-5,Evolving Training Data, Model Retraining*

Downloadable zip and GitHub Repository

- Zip: https://github.com/Subhasis-Biswas/Gesture_Recognition/archive/refs/tags/Static1.0.0.zip
- Repository: https://github.com/Subhasis-Biswas/Gesture_Recognition/
- Can be installed via: `pip install GestureProj` (as of v1.0.7, only Linux is Supported)

Clicking on the Package link will directly download the file `Gesture_Recognition-Static1.0.0` into the default download folder of the browser. Extract the `.zip` and navigate to the folder "GestureProj".

1 Introduction

This project is centered on real-time hand gesture recognition, utilizing OpenCV for webcam input processing. A pivotal element is the effective acquisition of personalized training data through the webcam, optimizing the process for dynamic hand gesture recognition. The integration of a strategically positioned bounding box, measuring 224x224 pixels, facilitates the efficient capture of diverse gestures in real-time.

A noteworthy feature of this project is the creation of a custom dataset by recording individual gestures through the webcam. To address storage concerns, the captured images are scaled down to a compact 32x32 resolution. This not only aids in minimizing file size but also significantly contributes to reducing the storage requirements. For example, considering 10000 images, each occupying 32x32x3 bytes (for RGB, unsigned 8bit integers; images stored in `.png` format), the total storage is approximately 30 megabytes.

The subsequent training of the model, leveraging the Keras/TensorFlow framework, involves this streamlined dataset. It's noteworthy that the deep learning model, designed for hand gesture recognition, comprises around 35,000 trainable parameters. The combination of a small image size and a relatively low number of parameters results in remarkably short training times even when starting from scratch.

This trained model is then poised for deployment, enabling real-time hand gesture recognition with broad applications in human-computer interaction. In the upcoming sections, we will explore the intricacies of dataset creation, model training, and the deployment of the hand gesture recognition system. This comprehensive exploration will provide insights into the efficient management of training data and the model's optimal performance.

1.1 Key Components

1. **Live Webcam Feed:** The system harnesses the capabilities of a user's webcam to capture a continuous live feed. This feed becomes the real-time input for the gesture recognition system, allowing users to control applications seamlessly.

2. **Gesture Classification:** The core functionality revolves around the accurate classification of hand gestures. The system recognizes a set of predefined gestures, including pointing, forming a V sign, numerical gestures (e.g., three, four, five), and custom gestures based on user training. This classification is fundamental for mapping user gestures to specific commands or actions within an application.

3. **User-Friendly Interface:** To facilitate user interaction, the system employs a modular and straightforward interface. Users are presented with options to either engage in gesture detection using the webcam or explore additional functionalities, namely recording new images to train upon, refreshing the model and removal of data. The interface is designed to accommodate users with varying levels of technical expertise, promoting accessibility.

4. **Dynamic Gesture Training:** Users can contribute to the system's flexibility by training it to recognize custom gestures. This dynamic training capability enhances the system's adaptability to individual user preferences and expands its range of applications.

5. **Real-time Visual Feedback:** During gesture detection, the system provides real-time visual feedback by printing predictions on the console. This feature enhances user engagement and helps users understand how the system interprets their gestures.

6. **Session Management:** The system allows users to manage gesture training sessions efficiently. Users can selectively delete sessions, removing associated training data, and view log files to track system interactions and performance over time.

2 Implementation

2.1 File Organization

```
GestureProj/                        ## Working Directory
.............
     main.py
     remover.py
     readme.txt
     gestures.png
     rootfold/
 ...........
         0/                         ## 0-5 folders contain training images
         1/
         2/
         3/
         4/
         5/
         cvfeed.py
         data_capture.py
         global_main.py
         remover__.py
         trainer.py
         model_lowres.keras
         log.csv
```

A detailed explanation of the tasks handled by individual files is given in the appendix.

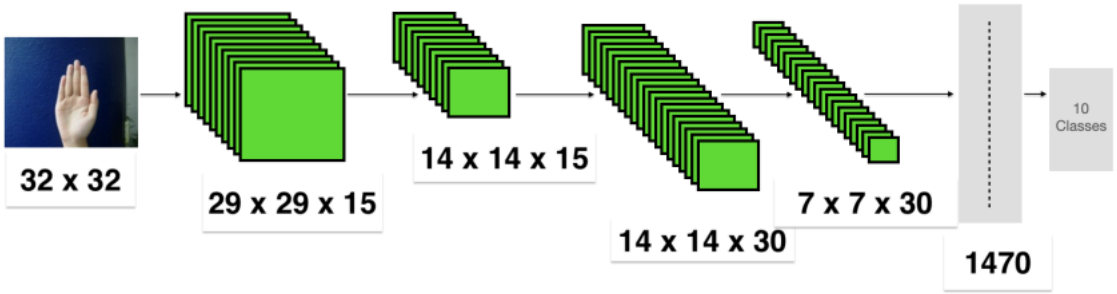
2.2 CNN Model Architecture

The CNN model, inspired by the architecture described in the paper [Eid and Schwenker(2023)], titled *Visual Static Hand Gesture Recognition Using Convolutional Neural Network*, is designed as follows:

- Input Layer: 32x32 pixel image.
- **Convolutional Layer 1:**
  - Filters: 15 filters of size 6x6.
  - Output Maps: 15 maps of size 29x29.
  - Activation: ReLU.
- **Max-Pooling Layer 1:**
  - 2x2 max-pooling, resulting in 15 output maps of size 14x14.
- **Convolutional Layer 2:**
  - Filters: 10 filters of size 3x3.
  - Output Maps: 30 maps of size 14x14.
  - Activation: ReLU.
- **Max-Pooling Layer 2:**
  - 2x2 max-pooling, resulting in 30 output maps of size 7x7.
- **Fully Connected Layer:**
  - Concatenates the output maps from Layer 2.
  - Dropout Probability: 0.5.
- **Output Layer:**

– Number of Classes: 6.

This architecture aims to capture hierarchical features through convolutional and pooling operations, with dropout used to reduce overfitting during training.



Our CNN model follows the architecture closely, with the following specifications:

Layer (type)	Output Shape	Param #
conv2d	(None, 27, 27, 15)	1635
max_pooling2d	(None, 13, 13, 15)	0
activation	(None, 13, 13, 15)	0
conv2d_1	(None, 11, 11, 10)	1360
max_pooling2d_1	(None, 5, 5, 10)	0
activation_1	(None, 5, 5, 10)	0
flatten	(None, 250)	0
dense	(None, 128)	32128
dropout	(None, 128)	0
dense_1	(None, 6)	774

Table 1: Model Architecture Summary
(Notice the slight differences from the original)

Total params: 35897 (140.22 KB), **Trainable params:** 35897 (140.22 KB),**Non-trainable params:** 0 (0.00 Byte)

Compiled with:
– **Optimizer:** Adam optimizer
– **Loss Function:** Sparse categorical crossentropy
– **Metrics for Evaluation:** Accuracy

3 Training from scratch

Device Specifications:

- **Operating System:** Ubuntu 20.04.6 LTS (Focal Fossa)
- **BIOS:**
  - Version: 1.14.0
  - Vendor: Dell Inc.
  - Date: 08/10/2023
- **CPU:**
  - AMD Ryzen 3 3250U (2 cores, 4 threads)
  - Clock Speed: 1349MHz (base), 2600MHz (max)
- **Memory:**
  - 12GiB DDR4
    - Bank 0: 4GiB, 3200 MHz
    - Bank 1: 8GiB, 2667 MHz
- **Storage:**
  - WDC PC SN530 NVMe 256GB
    - Partitions:
      - /: 238GiB (EXT4)
      - /boot/efi: 277MiB (FAT32)

- **USB:**
  - USB 3.1 Controller: Raven2 USB 3.1
    - Devices: Logitech USB Receiver, Webcam HD, HP Wired Keyboard, USB 2.0 Hub (Bluetooth)
- **GPU:**
  - Integrated Picasso GPU (amdgpu driver)
  - Display: VGA compatible controller

Training outcomes

80-10-10 Train-Validation-Test Split in all cases.

- 6000 images, 1000 per class. The images are gestures against a blank background; from scratch.

Epoch	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
1	0.7796	0.7390	0.1069	0.9783
2	0.1036	0.9694	0.0200	0.9983
3	0.0430	0.9881	0.0089	0.9983
4	0.0225	0.9948	0.0033	0.9983
5	0.0131	0.9973	0.0022	0.9983
6	0.0112	0.9979	9.0203e-04	1.0000
7	0.0070	0.9983	2.6514e-04	1.0000
8	0.0097	0.9973	5.3103e-04	1.0000
9	0.0043	0.9992	4.7233e-04	1.0000
10	0.0024	0.9998	2.0151e-04	1.0000

Table 2: Training and Validation Metrics

Training Time: 22.47 seconds **Test Accuracy:** 1.0

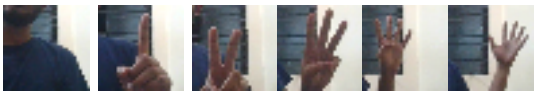
Sample images: 

- After adding 6000 more images with varying backgrounds; model learns on top of earlier version:

Epoch	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
1	0.5314	0.7900	0.2917	0.8917
2	0.2857	0.8853	0.1851	0.9375
3	0.2077	0.9201	0.1386	0.9517
4	0.1662	0.9382	0.1130	0.9608
5	0.1292	0.9542	0.0800	0.9708
6	0.1228	0.9546	0.0798	0.9683
7	0.0997	0.9623	0.0601	0.9808
8	0.0794	0.9707	0.0524	0.9858
9	0.0714	0.9719	0.0362	0.9892
10	0.0724	0.9750	0.0453	0.9842

Table 3: Training and Validation Metrics

Training Time: 45.08 seconds **Test Accuracy:** 0.9842

New sample images: 

- After another 1200 such images; model loads from previous saves:

Epoch	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
1	0.1399	0.9554	0.0535	0.9855
2	0.0828	0.9734	0.0401	0.9855
3	0.0776	0.9732	0.0368	0.9899
4	0.0603	0.9806	0.0224	0.9928
5	0.0583	0.9802	0.0310	0.9913
6	0.0547	0.9812	0.0313	0.9906
7	0.0516	0.9817	0.0253	0.9920
8	0.0425	0.9857	0.0293	0.9913
9	0.0417	0.9858	0.0182	0.9942
10	0.0428	0.9851	0.0142	0.9957

Table 4: Training and Validation Metrics

Training Time: 52.10 seconds **Test Accuracy** 0.9964

4 Deployment Steps

1. **Setup Environment:** - Ensure Python 3.x is installed. - Install required packages: numpy, pandas, tensorflow, keras, opencv, and matplotlib.
2. **Run the Program:** - Set the working directory to "./GestProj" and execute `python3.x main.py`. - Follow the prompts to choose between gesture detection and other functionalities.
3. **Reset Training Data:** - To reset training data, run `python3.x remover.py` and enter the provided random number for confirmation.
4. **Model Reset (Optional):** - If a complete model reset is desired, manually delete the file "model_lowres.keras" in "./GestProj/rootfold".

Notes

- The project seems well-structured and modular, making it easy to understand and update.
- Consider updating packages and dependencies based on the current Python and library versions.

Various Limitations and Options for Future Improvements

1. **Limited Gesture Vocabulary:** The project recognizes a specific set of hand gestures (0 to 5). Expanding the gesture vocabulary could enhance the system's versatility and user experience.
2. **Sensitivity to Hand Placement:** The hand gesture detection relies on a fixed bounding box (100x100 to 324x324 pixels). If a user's hand goes beyond this region, the system will not effectively recognize gestures. Consider incorporating more dynamic hand tracking for robustness.
3. **Manual Data Capture:** The data capture process involves manual recording of gestures, which might be time-consuming and subject to variations in lighting conditions. Automatic data augmentation techniques or collecting data from diverse sources could improve model generalization.
4. **Limited Model Architecture:** The simplicity of the model architecture (two convolutional layers, one fully connected layer) might limit its ability to capture complex hand gestures. Exploring more advanced architectures or transfer learning approaches could lead to better performance.
5. **Dependency on External Libraries:** The project relies on external libraries like OpenCV, TensorFlow, and Keras. Changes or updates to these libraries might impact the project's compatibility in the long term.
6. **Limited Error Handling:** The code lacks comprehensive error handling mechanisms, making it more susceptible to crashes or unexpected user inputs. Incorporating robust error handling could improve the system's stability.

Appendix

.1 Architecture

Main Module (`main.py`)

Responsibilities:
- Main Execution: Serves as the entry point for the Gesture Recognition System.
- Orchestrates the initialization of the main system components.

Noteworthy Features:
- System Initialization: Coordinates the initialization of modules, ensuring a smooth start of the application.

Main Module (`rootfold/global_main.py`)

Responsibilities:
- Welcome Message: Greets users and introduces them to the main page.
- User Interaction: Prompts users to choose between detecting gestures or exploring other options.
- Conditional Routing: Directs users to the live webcam feed or other functionalities based on their choice.

Noteworthy Features:
- Dynamic Model Loading: If the gesture recognition model is not found or fails to load, the system gracefully handles the situation by generating a new model through the training module.

Webcam Feed Module (`rootfold/cvfeed.py`)

Responsibilities:
- Webcam Access: Interfaces with the user's webcam to capture live video.
- Model Integration: Incorporates the gesture recognition model for real-time predictions.
- Bounded Region Overlay: Enhances user experience by visually indicating the region of interest within the webcam feed.

Noteworthy Features:
- Model Reloading: Attempts to reload the model and, if unsuccessful, triggers model generation and training via the training module.

Data Capture Module (`rootfold/data_capture.py`)

Responsibilities:
- Instruction Display: Guides users on available options and interactions.
- User Input Handling: Manages user choices for recording gestures, refreshing the model, deleting sessions, and quitting.
- Session Management: Enables users to delete specific sessions or view log files.

Noteworthy Features:
- Session Deletion: Allows users to selectively delete sessions, removing associated training data and updating the log file.

Gesture Training Module (`rootfold/trainer.py`)

Responsibilities:
- Model Loading: Attempts to load a pre-existing model or generates a new one.
- Dataset Management: Shuffles and splits the dataset into training and validation sets.
- Training Process: Executes the training process, utilizing a Convolutional Neural Network (CNN).
- Model Saving: Persists the trained model for future use.

Noteworthy Features:
- Model Refresh: Provides users with the option to refresh the model, accommodating new gestures or data.

Complete Deletion Script (`remover.py`)

Responsibilities:
- Data Deletion: Executes the deletion of all saved data, including training images and log files, by calling `remover__.py` module within "rootfold".
- User Confirmation: Ensures user intent by prompting for a confirmation code.

Noteworthy Features:
- Cancellation Handling: Allows users to cancel the deletion process through keyboard interrupt (Ctrl+C).

Deletion Module (`rootfold/remover__.py`)

Responsibilities:

- Data Deletion: Executes the deletion of all saved data in folders $0, 1, ..., 5$ and resets the log file.

.2 User Interface

Different aspects of this console-UI program are handled by different scripts. Here some functionalities are mentioned and within parenthesis, the scripts responsible for doing the respective work behind the scenes.

Main Page (`global_main.py`)

Interactive Features:

- Welcome Message: Greets users and sets the tone for interaction.
- Options Presentation: Displays options to either detect gestures or explore other functionalities (0 or 1).
- User Choice Prompt: Requests user input to determine the selected course of action.
- Conditional Routing: Guides users to the live webcam feed or other functionalities based on their choice.

Webcam Feed (`rootfold/cvfeed.py`)

Interactive Features:

- Webcam Initialization: Accesses the user's webcam for live video capture.
- Model Integration: Overlays predictions from the gesture recognition model on the live feed.
- Bounding Box Display: Highlights the region of interest for enhanced visualization.
- User Exit: Allows users to exit the webcam feed by pressing 'q'.

Data Capture (`rootfold/data_capture.py`)

Interactive Features:

- Instruction Display: Provides a menu-driven interface with clear instructions.
- User Input Handling: Captures user choices for recording gestures, refreshing the model, deleting sessions, or quitting.
- Capturing New Images: Asks for a specific number of frames for all the gestures (to avoid the problem of class imbalances). New frames are recorded by keeping the r key pressed and confirming after reaching desired count of every gesture whether the user is satisfied with the quality of the data recorded. Data is updated only when the user confirms that the data is satisfactory across all the gesture classes. User can quit a recording session by pressing $q/esc/Ctrl + C$ when the program is actively asking frames/asking for user satisfaction.
- Logging activity: Update data is logged in `log.csv` file, with the unix format of the start of a session being the identifier of the data recording sessions. If some data is being deleted through the program (the user is not supposed to modify the dataset by manually, as to maintain proper functionality), it removes the entire record of data from log after removal of the irrelevant training images.
- Post-Interaction Options: After each operation, users are prompted to either return to the main page or exit the application.

Model Training/Refreshing (`rootfold/trainer.py`)

Interactive Features:

- Model Loading: Attempts to load a pre-existing model or generates a new one in the absence of the *model_lowres.keras* file.
- Dataset Initialization: Reads and organizes the dataset, preparing it for training.
- Training Execution: Trains the gesture recognition model using a CNN architecture.
- Model Saving: Persists the trained model.
- Also offers an option to refresh the model (if the dataset has been modified without training the model immediately afterwards).
- User Prompt: Offers users the option to return to the main page or exit the application.

Session Deletion (`rootfold/data_capture.py`)

Interactive Features:

- Displaying the different sessions recorded all throughout.
- Providing the user with the option to view/delete a particular session deemed irrelevant for the model.

- Asks the user for confirmation after proceeding with the deletion choice. If the other option is selected, the complete file list of that particular session is displayed, so that the user can go ahead and look up the samples themselves.
- One drawback is that while displaying the log records, some columns are not displayed properly due to the long names. So, it's better to note down the session ID and look for the file names in an excel sheet handling software like Microsoft Office/LibreOffice.

References

[Eid and Schwenker(2023)] Ahmed Eid and Friedhelm Schwenker. Visual static hand gesture recognition using convolutional neural network. https://www.mdpi.com/1999-4893/16/8/361, 2023.