An Intelligent Detection System for Window Opening and Closing Based on Deep Learning Techniques

xinmingFeng

Github: https://github.com/xinming-Feng/casa0018/tree/main/Assessment

Edge Impulse: https://studio.edgeimpulse.com/public/672207/live

Introduction

This project presents a vision-based system designed to detect whether a window is open or closed. When the camera identifies that the window is in an open state, a red LED light is activated. Conversely, when the system detects that the window is closed, a green LED light is illuminated. To date, there has been little precedent for using camera-based systems specifically to monitor the open or closed state of windows within this domain. However, related work has explored the use of depth cameras to identify the state of doors (Ramôa et al., 2020), which serves as an inspiring foundation for this investigation.

Motivated by this prior research, we propose that a similar approach could be extended to window state detection. Unclosed windows can be a source of significant inconvenience in everyday life—for example, external noise may disturb sleep, especially for individuals residing near busy roads. Additionally, open windows may allow insects to enter or permit rain to be blown indoors during inclement weather. This system aims to mitigate such issues by providing users with timely visual cues regarding window status, thereby addressing a seemingly trivial yet often overlooked aspect of daily living.

Research Question

The central problem this study seeks to address is the automatic recognition of whether a window has been properly closed. In my daily life, I frequently forget to close windows during the day, only to be awakened in the middle of the night by the disruptive sound of passing vehicles. This recurring issue has caused considerable discomfort and has motivated the development of a reliable system to monitor window status in a timely and intelligent manner.

Application Overview

The development of this project begins with the collection of a dedicated dataset, as it is fundamentally a vision-based recognition system reliant on camera input. Accordingly, images serve as the primary input for the dataset. Once the dataset is established, the overall deep learning framework is outlined. After data preprocessing, an appropriate classification model is selected along with the output strategy—this project formulates the task as a binary classification problem, identifying whether a window is "open" or "closed".

The process consists of several key stages, starting with data preprocessing, which involves preparing the images for effective feature extraction. Specific preprocessing steps include cropping images to uniform dimensions, optionally converting images to grayscale, and extracting relevant visual features.

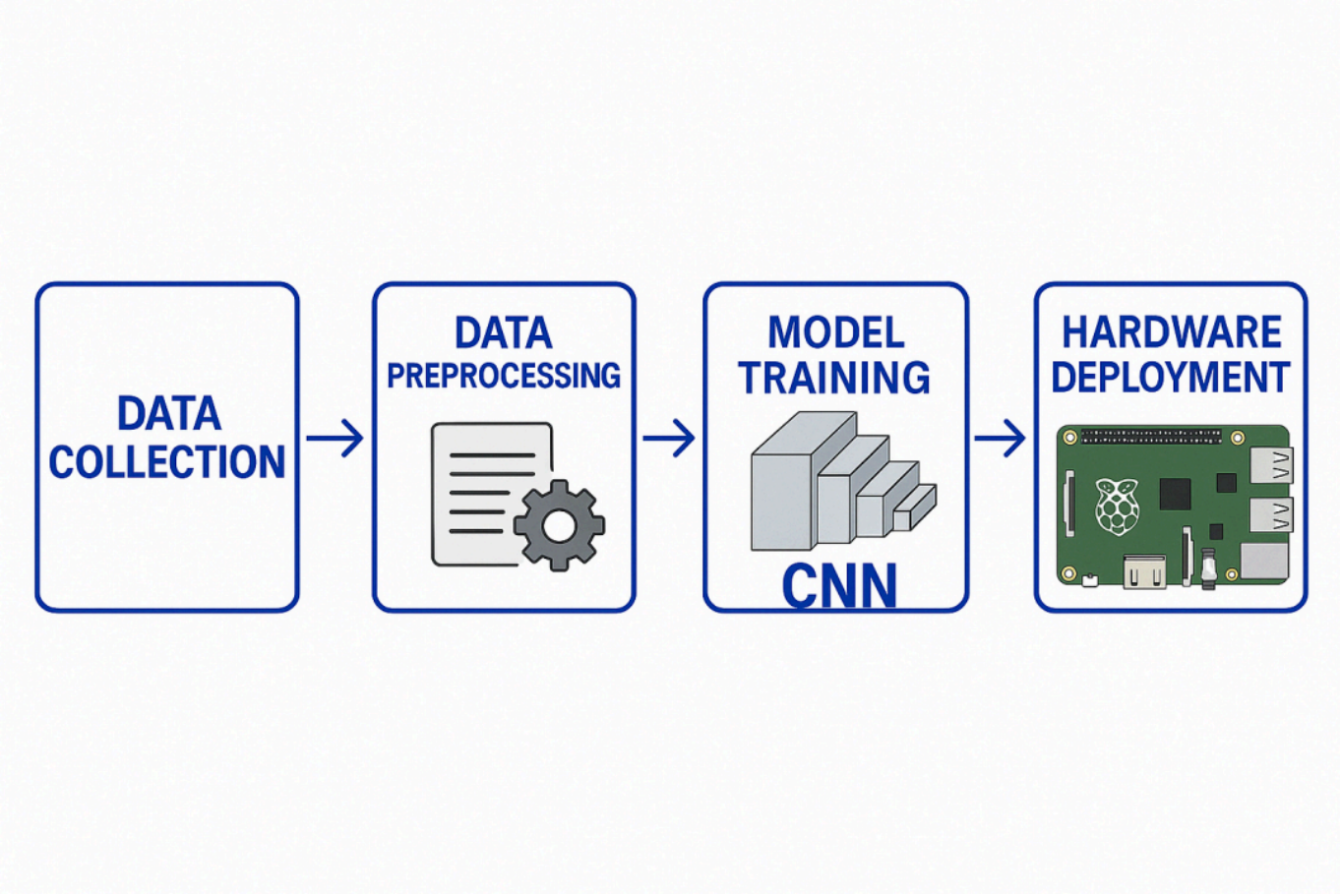
Subsequently, the project proceeds to model training, in which the chosen model is trained on a designated training set and validated on a separate validation set to fine-tune the model parameters. For this project, a Convolutional Neural Network (CNN) is adopted due to its strong performance and learning capabilities in computer vision applications.

Following the training phase, the model's true performance is evaluated on a test set to determine its final classification accuracy. Once satisfactory performance is achieved, the model is deployed onto hardware for real-world testing.

Hardware Components Used:

- Raspberry Pi 4B
- Raspberry Pi Camera Module 3
- Two LED indicators (Red and Green)

The overall project workflow is illustrated in the following process diagram:



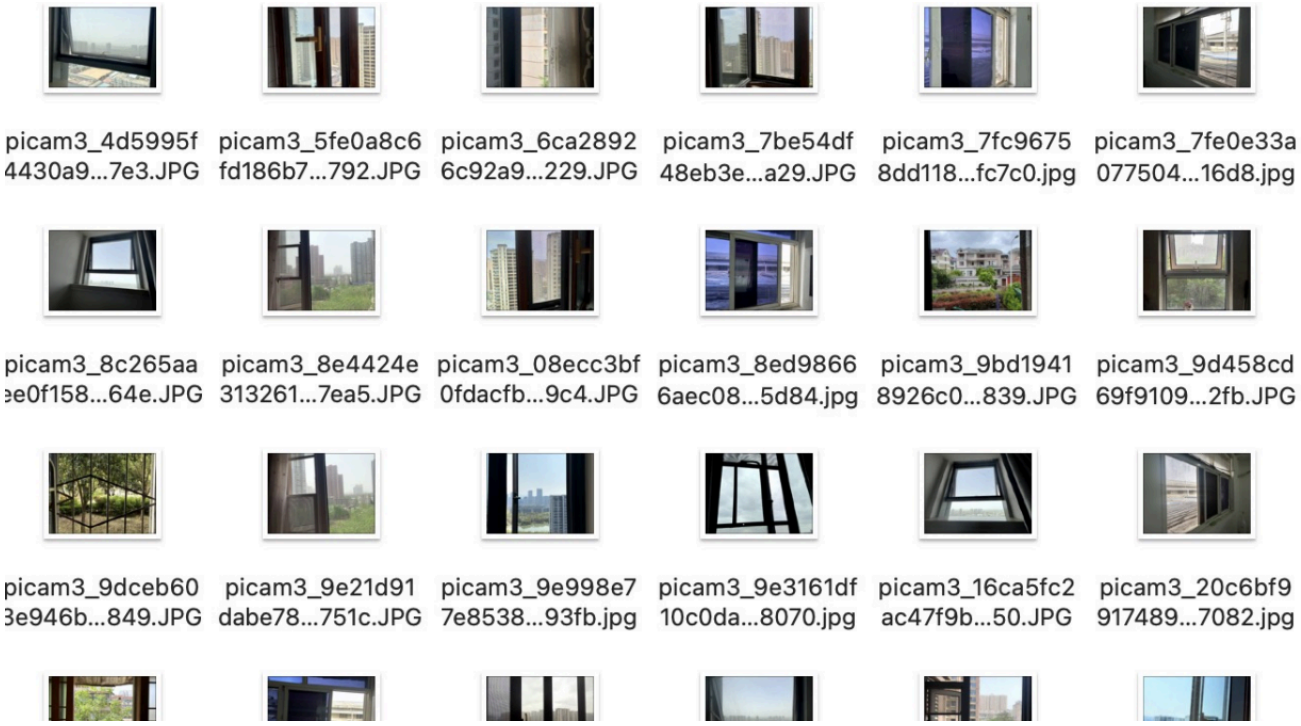
Data

The dataset used in this project was self-collected to ensure diversity in the input features. A total of 228 images were gathered, encompassing 35 distinct window samples, with each window photographed in both open and closed states from various angles. To increase the robustness of the model, images were collected under different lighting conditions, including daytime, dusk, and nighttime.

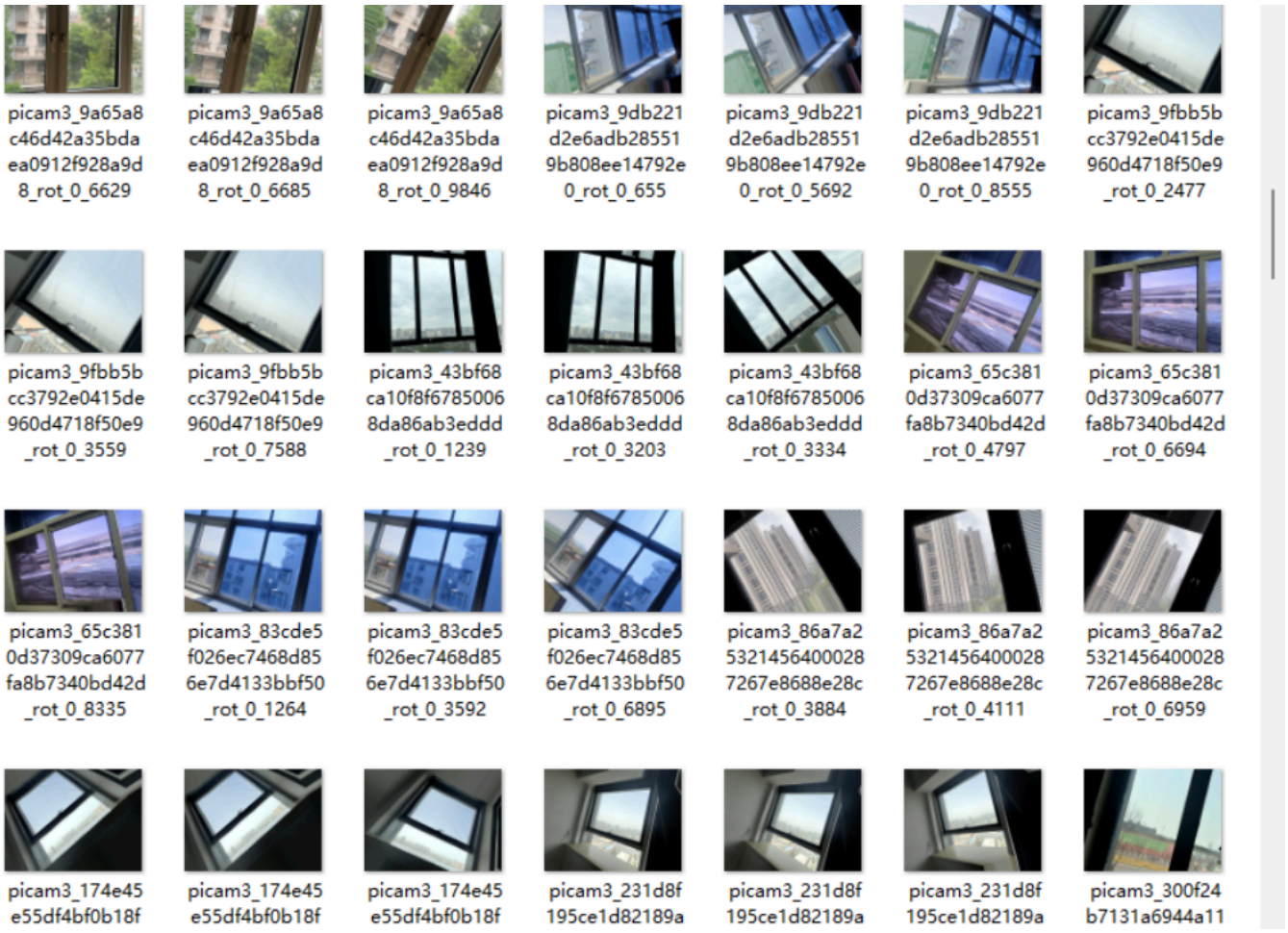
The majority of these images were captured by friends and classmates upon request. However, the final deployment environment relies on the Raspberry Pi Camera Module 3, which differs from the devices

used during the initial data collection. To address this discrepancy, a Python script named camera.py was developed to preprocess the original dataset and simulate the imaging characteristics of the Raspberry Pi Camera Module 3. The script adjusted key image parameters such as exposure and sharpness, and performed preliminary cropping to standardize all images to the same dimensions.

This preprocessing step ensured that the training data more closely resembled the images captured by the actual deployment hardware, thereby enhancing the model's generalization and applicability to real-world conditions.



The original dataset of 228 images was too small, leading to overfitting during training. To address this, a Python script (extend.py) was used to augment the data by rotating images, generating 2–3 variants per original image. This increased the dataset size to 534 images, improving model generalization and performance.



After data expansion, the model will no longer be overfitted. Meanwhile, I attempted to process the data into grayscale images for model training and compared the training results with those of RGB images.

Model

The core model employed in this project is a Convolutional Neural Network (CNN), selected for its strong performance in image classification tasks. Throughout the development process, the architecture was iteratively refined to better align with the project's requirements. These refinements involved modifying the number and arrangement of convolutional and pooling layers, as well as experimenting with the placement and number of dense (fully connected) layers around the dropout layer to improve regularization and learning capacity.

After multiple rounds of experimentation and performance evaluation, the final and most effective architecture was determined as follows:

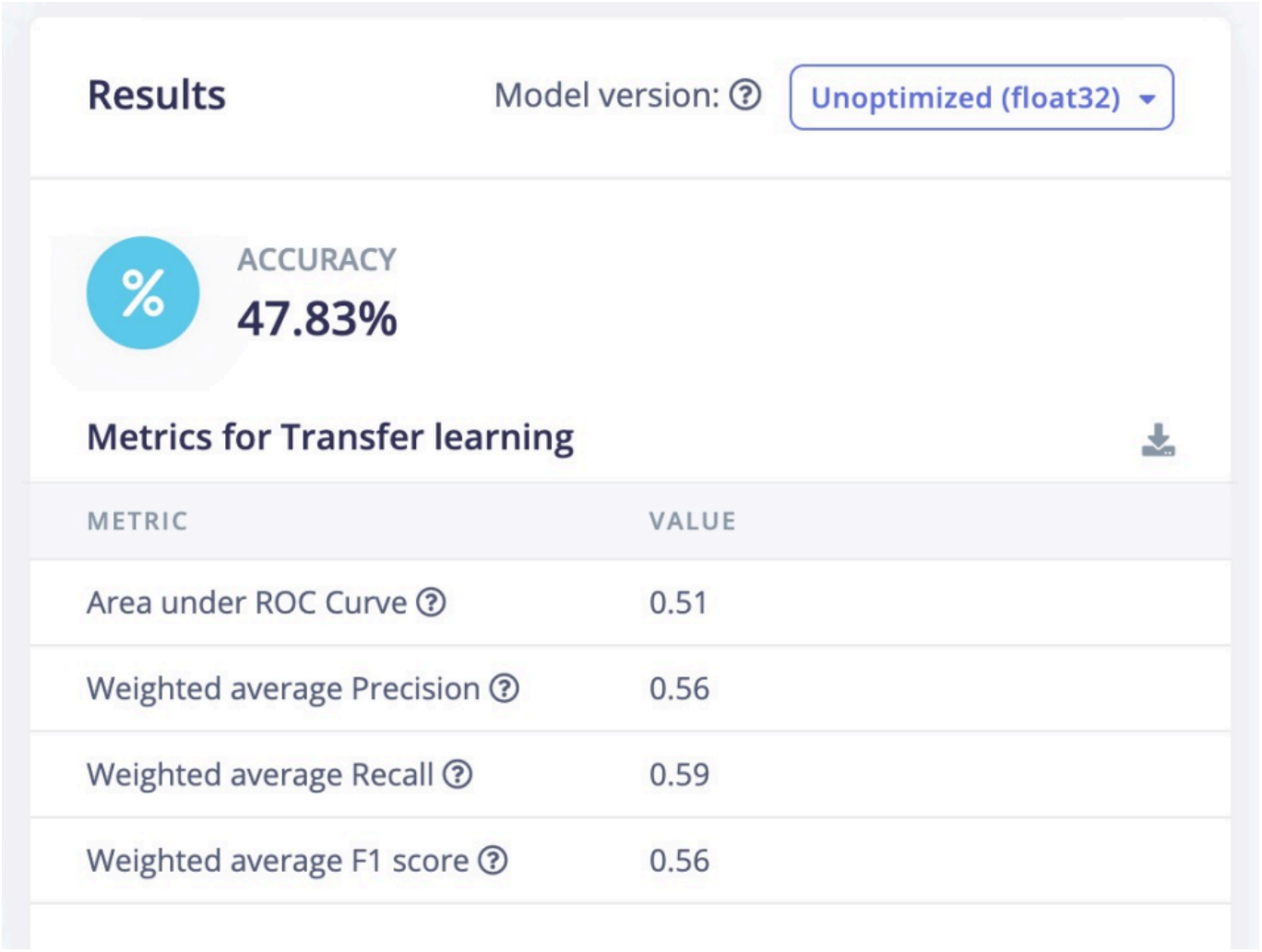
Input Layer → 2D Convolution + Pooling Layer → Flatten → Dropout → Dense → Output Layer

This configuration yielded the best results on the test set, achieving a maximum classification accuracy of 71.56%. The balance between model complexity and overfitting was carefully managed, and this architecture proved to be the most robust given the dataset and the binary classification task at hand.

The decision to adopt a Convolutional Neural Network (CNN) as the core model architecture was driven by the nature of the project—a camera-based computer vision application. CNNs are particularly well-suited for image analysis tasks due to their ability to extract local features through the application of convolutional kernels that slide across the input image. This mechanism is highly effective at capturing important visual characteristics such as edges, textures, and localized patterns, which are crucial for distinguishing between open and closed window states.

Furthermore, CNNs are known for their robustness in handling variations in visual input. Common image transformations such as translation, rotation, or scaling typically introduce minimal disruption to CNN performance, making the architecture especially appropriate for deployment on embedded hardware like the Raspberry Pi Camera Module, where the captured images may vary in angle or lighting conditions.

In addition to CNN, transfer learning was explored as an alternative approach. Specifically, the MobileNetV1 (96x96, width multiplier 0.25) model was selected due to its lightweight nature and efficiency in mobile vision applications. However, experimental results showed that its performance was suboptimal for this task: the test accuracy remained around 50%, which is insufficient for a binary classification problem and indicates limited learning capability on this dataset.



Experiments

What experiments did you run to test your project? What parameters did you change? How did you measure performance? Did you write any scripts to evaluate performance? Did you use any tools to evaluate performance? Do you have graphs of results?

Tip: probably ~300 words and graphs and tables are usually good to convey your results!

Results and Observations

Synthesis the main results and observations you made from building the project. Did it work perfectly? Why not? What worked and what didn't? Why? What would you do next if you had more time?

Tip: probably ~300 words and remember images and diagrams bring results to life!

Bibliography

If you added any references then add them in here using this format:

1. Last name, First initial. (Year published). Title. Edition. (Only include the edition if it is not the first edition) City published: Publisher, Page(s). http://google.com
2. Last name, First initial. (Year published). Title. Edition. (Only include the edition if it is not the first edition) City published: Publisher, Page(s). http://google.com

Tip: we use https://www.citethisforme.com to make this task even easier.

Declaration of Authorship

I, AUTHORS NAME HERE, confirm that the work presented in this assessment is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Digitally Sign by typing your name here

ASSESSMENT DATE

Word count: