Home Security Detection System

Zitong Wang
WBooth SEPT
McMaster University
Hamilton, Canada
wangz889@mcmaster.ca

Xuanyu (Liam) Liu
WBooth SEPT
McMaster University
Hamilton, Canada
liux147@mcmaster.ca

Shu Li
WBooth SEPT
McMaster University
Hamilton, Canada
lis228@mcmaster.ca

Ping Li
WBooth SEPT
McMaster University
Hamilton, Canada
li3270@mcmaster.ca

Purva Singh
WBooth SEPT
McMaster University
Hamilton, Canada
singp36@mcmaster.ca

***Abstract*—This paper provides an overview of our IoT design in home security detection systems. The paper explains the user interface and database structure, hardware installation and algorithms, and image classification for host detection.**

***Index Terms*—Raspberry Pi, ultrasonic sensor, PiCam, dbLite, YOLO, OpenCV**

I. Introduction

Computer vision is one of the most important branches in the field of artificial intelligence research, and its face recognition technology has been widely used in many fields, such as security and access control systems. Raspberry Pi is a microcomputer system with an ARM core, which can be used with various general computer peripherals and Internet of Things (IoT) sensors.

In the context of the IoT Home Security Detection System project centered around Raspberry Pi, a comprehensive approach to face recognition has been devised. This method integrates various IoT devices with advanced computer vision algorithms. The project encompasses the development and implementation of several crucial system functionalities, including user management, face data entry, and facial comparison recognition.

II. Methodology

A. User Interface

The user interface consists of the following import module.

1) Express: This is the Express.js framework, which provides a robust set of features for web and mobile applications.

2) dbLite: This is a lightweight database module to interact with SQLite databases.

3) Pug: This template engine displays the user content.

B. Raspberry Pi Hardware

1) Pi Cam: The Pi Cam is an integral component of the project and refers to the camera module specifically designed for Raspberry Pi. It seamlessly integrates with the Raspberry Pi microcomputer system, enabling image and video capture for precise facial recognition.

2) Ultrasonic Distance Sensor: The Ultrasonic Distance Sensor(UDS) can detect movement in the environment to determine the real-time working status of the system. When no motion is detected, the system can enter a low-power mode to conserve energy, which contributes to power efficiency and improved user interaction. It can make the system operate more responsively and intelligently based on the dynamic conditions in its environment.

3) LED: The color of the LED light indicates the result of the system's processing. When the system detects a stranger, the LED lights up red. It also serves as a reminder or warning while recording the result.

C. YOLO-v8

In the development of the home security detection system. We selected YOLO (You Only Look Once) as our computer vision algorithm to detect the identity of the incoming person. We chose the YOLO algorithm because it is compatible with the Raspberry Pi implementation and coordinates with OpenCV to enable photographic/video classification.

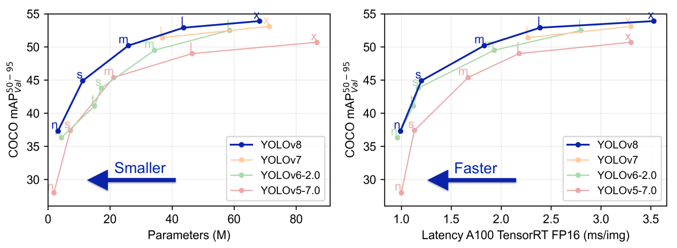


Fig. 1. YOLO version comparison diagram.

As shown in Fig. 1, we chose YOLO-v8n for the facial image classification since the model is smaller and has a higher processing speed than the COCO dataset's test reflection. YOLO-v8 is an improved version of YOLO from the previous YOLO-v5 model. The update includes replacing the C3 module (CSP Bottleneck with 3 convolutions) with the "coarse to find" module to improve feature analysis efficiency, replacing the 6*6 convolutional layer with a 3*3 convolution layer in the backbone to split up the computational effort during input feature extraction, removing bottleneck from the model by swapping all the 1*1 convolutional layer with 3*3 convolutional layer to eliminate the dimensional shrinkage.

Our project uses a binary classification between Robert Downey Jr.(RDJ) and other Unknown person values. The objective is to have the YOLO-v8 neural network model perform the incoming person's classification task and predict the person's identity. If the person is classified as RDJ, It will assign a host label to the detection result. Otherwise, it will return the value "unknown person."

1) Dataset, Data Pre-processing and Augmentation: The self-customized dataset is structured with images from the Kaggle Dataset "Pins Face Recognition." Where the data category is re-integrated with approximately 200 images for RDJ and 200 that are composed of random people.

The project utilizes the Roboflow open-source tool, which specializes in preprocessing and augmenting the dataset for classification assignment with the YOLO model. Labels are assigned in the folder structure to distinguish between classification objectives. Pre-processing solutions include grayscale resizing of the images, which are introduced to decrease training time and increase performance for the entire dataset. Further augmentation, including reshaping the images, adding noise, and color saturation, is implemented to expand training data for the YOLO model to learn by generating augmented versions of each image in the training set. This includes an adjustment in shape and color and adding noisy pixels. 1,059 images are generated as an outcome of the above procedure. Cache data is introduced to the content of Google Colab through Roboflow API to work with the YOLO-v8 model.

2) Model Training and Deployment: The model is trained with a set of parameters. This includes 20 epochs, 16 batches, and an Adam optimizer with learning rates of 0.000714 and 0.9 momentum. The training session took 0.309 hours to complete, and the result is demonstrated in the loss/accuracy ROC curve with confusion matrix for the test set.

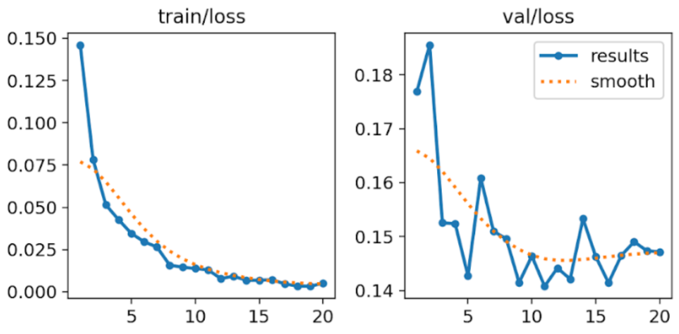


Fig. 2. Training and testing loss comparison

As shown in Fig. 2, the training loss indicates that the model can converge smoothly. Meanwhile, for validation loss, strong fluctuation is observed. The smooth curve also indicated signs of overfitting, which arose around 16 epochs. Suggested early stopping should be enforced to intervene with the process.

As shown in Fig. 3, the confusion matrix consists of 26 true positives, 14 true negatives, 3 false positives, and 1 false negative. Resulted in 90.9% accuracy. The high value observed for true positive and negative supported the fact that the model is reliable for prediction.

III. Implementation

A. UI

Below is the code for Express.js Setup:

1) const app = express(): Initializes an Express application.

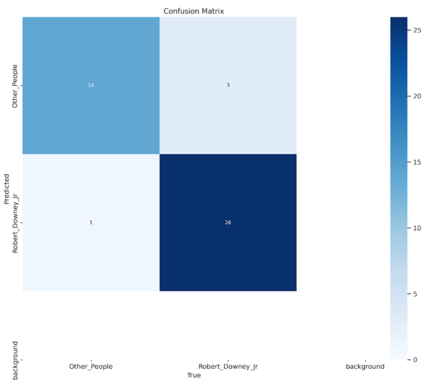


Fig. 3. Test dataset confusion matrix

2) app.use(express.json()), app.use(express.urlencoded(extended: true)): These middleware functions are used to parse incoming request bodies in JSON and URL-encoded format.

3) app.use(express.static("public")): Serves static files (like CSS, JavaScript, images) from the 'public' directory.

4) app.set('view engine', 'pug'): Sets Pug as the template engine for rendering views.

5) app.use('/images', express.static('images')): Serves image files from the 'images' directory.

Below is the descriptions for Express Routes:

Routes like app.get('/') and app.get('/login') define endpoints for the home page, login, and registration, rendering respective Pug templates.

app.post("/logIn") and app.post("/newUser"): Handle login and new user registration functionality. They use findUser and addUser functions for database interactions.

B. Hardware Setup

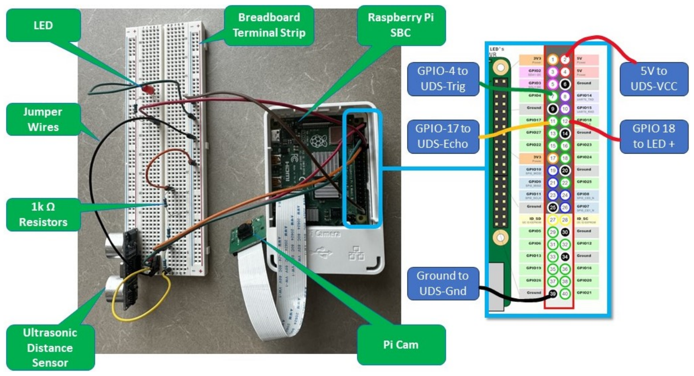


Fig. 4. Hardware setup wiring illustration

Device setup is shown in Fig.4, details are as follows:

1) Connect the PiCam: : Use the provided 15-pin Flexible Flat Cable (FFC) to connect the PiCam to the Raspberry Pi's camera module port.

2) Ultrasonic Distance Sensor Connection: : Place the Ultrasonic Distance Sensor on the breadboard. Then connect the 4 legs of the sensor as follows: VCC leg to the 5V power output on the breadboard; Trig leg to GPIO-4 on the Raspberry

Pi; Echo leg to GPIO-17 on the Raspberry Pi; Gnd leg to the Ground on the breadboard; add 1k ohm resistor between Echo leg to GPIO 17; dd 2k ohm resistor between Gnd to Ground.

3) Red LED Connection: : Connect a red LED to the breadboard. Connect the positive (+) leg of the LED to GPIO-18 on the Raspberry Pi. Connect the negative (-) leg of the LED to the ground on the breadboard.

C. Device Operation

To simulate real-world usage scenarios, this home security detection device is designed to operate continuously unless interrupted by keyboard input for illustration purposes. The application is programmed using Python, incorporating four custom functions and a main loop for continuous detection. To implement a self-trained image classification model (discussed in the following section) for facial recognition, a virtual environment was configured with all the necessary libraries and packages installed. Fig. 5 presents the operation flow chart, followed by a detailed procedural walkthrough, with sample output shown in Fig. 6.

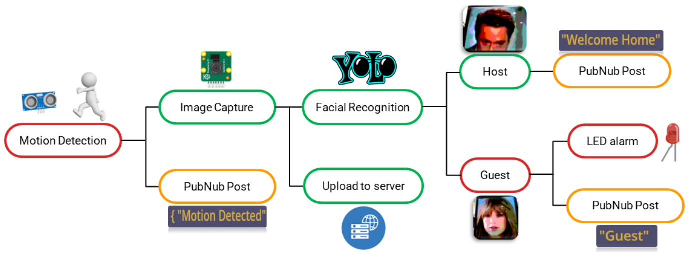
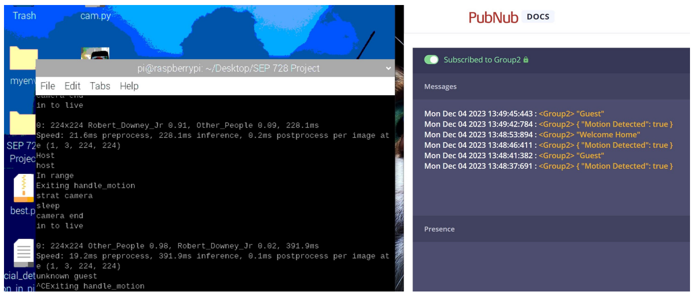


Fig. 5. Operation flow chart



```
in to live

0: 224x224 Robert_Downey_Jr 0.91, Other_People 0.09, 228.1ms
Speed: 21.6ms preprocess, 228.1ms inference, 0.2ms postprocess per image at
e (1, 3, 224, 224)
Host
host
In range
Exiting handle_motion
strat camera
sleep
camera end
in to live

0: 224x224 Other_People 0.98, Robert_Downey_Jr 0.02, 391.9ms
Speed: 19.2ms preprocess, 391.9ms inference, 0.1ms postprocess per image at
e (1, 3, 224, 224)
unknown guest
^CExiting handle_motion
```

```
Mon Dec 04 2023 13:49:45:443 : <Group2> "Guest"
Mon Dec 04 2023 13:49:42:784 : <Group2> { "Motion Detected": true }
Mon Dec 04 2023 13:48:53:894 : <Group2> "Welcome Home"
Mon Dec 04 2023 13:48:46:411 : <Group2> { "Motion Detected": true }
Mon Dec 04 2023 13:48:41:382 : <Group2> "Guest"
Mon Dec 04 2023 13:48:37:691 : <Group2> { "Motion Detected": true }
```

Fig. 6. Sample output

1) Initial setup: : Import PubNub package for message publication and real-time monitoring. Import GPIO, DistanceSensor, Picamera2 for hardware control. Import cv2 and YOLO to run facial recognition model. Initialize GPIO pins, global variables, and PubNub keys.

2) Motion detection: : Implemented the **handle motion**() function to detect motion within a range, utilizing a threshold of 0.2 meters. Once a motion is detected, publish a message "Motion Detected" on the PubNub console.

3) Image capture: : Function **capture image**() will execute to take a picture of the person once motion is detected. The image will be stored on the local drive (Pi drive) to

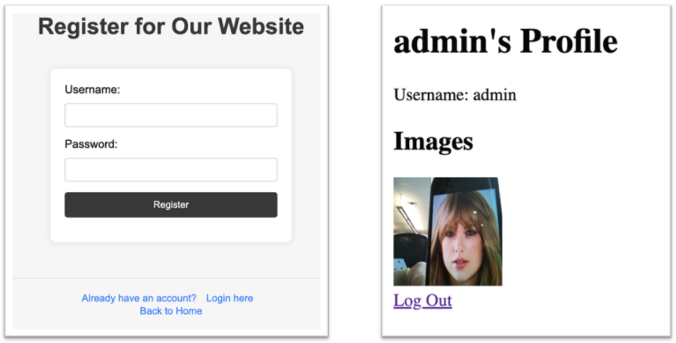


Fig. 7. Server user interface

be sent to the facial recognition model for security checks. Simultaneously, the photo will be uploaded to the web server for user verification. Fig. 7 shows the registration, login , and captured image verification user interface. Once logged in, the user will be able to check the most recent picture captured by the security system for verification.

4) Facial recognition: : The image took by PiCam will be sent to function **live stream detection**() for facial recognition. For demonstration, actor Mr. Robert Downey Jr. was trained as the "Host" of the property, and he was the only person who could pass the security check, with a welcome message published on the PubNub console. For other person, the model will classify as he or she as "Guest", and block by the security check, then post an alert message on PubNub.

5) LED alert: : For security purposes, if a 'Guest' is recognized, a red LED light will be activated as an alarm, as a feature defined with function **check host via video result**().

D. Image Classification Model

The YOLO-v8 image classification model is exported in the format of a best.pt file placed on the Raspberry Pi. After configuring the ultralytics packages and initializing the camera, the OpenCV feature is introduced for the image classification task.

After receiving the returns from the previous function indicating the person is arriving, the motion sensor will detect the person's presence and forward a true statement to activate the Pi Cam. This action will initially reset the parameter for a host counter and a detection string for further processing. The ideal implementation will trigger the function to enable the live streaming feature of the Pi Cam for 20 seconds. During this period, the footage streamed by the camera will proceed with OpenCV to perform an inference analysis with the YOLO-v8 model. OpenCV will loop through each frame with a 30ms period during the live stream and process the confidence score comparison on each particular frame. Since we are only conducting binary classification between RDJ and other people while the scale of the dataset is less than ideal, the threshold determining if the host is RDJ is set to 0.95 to eliminate any potential false cases. During the live cam action, the host counter will continue to add up while receiving a frame with a 0.95 or higher confidence score for RDJ. When the counter reaches 30, it will set the detection string with the value “host” and conclude that the incoming person is the host (RDJ). If during the 20 seconds, the counter does not reach 30, it will set the detection string with the value “unknown person”, and update the unknown person’s arrival, and send a message back to the host. In practice, due to the hardware difficulty in operating under the ideal resolution, the team decided to perform the prediction directly over the captured image with 0.9 confidence score set as the recognition threshold.

IV. Limitations and future work

A. *Image classification*

While implementing the deep learning neural network, the team has also considered alternate options such as huskylens AI vision camera system. Although huskylens offer a wide variety of classification capabilities, including object and multi-facial detection, the detection size and robustness of the AI system are inferior compared to the implementation of the deep learning neural network. On the other hand, despite the neural network’s strong capability to perform large-scale dataset comparison, the detection diversity and training effort is limited and more complex than the implementation of huskylens. Additionally, the cost of implementing PiCam is only 34% compared to the price for the huskylens AI vision camera. This is another important note when considering the specialization in broadcasting the product.

V. Conclusion

The Home Security Detection System integrates computer vision, IoT devices, and artificial intelligence to enhance residential security using the Raspberry Pi, YOLO-v8 image classification model, and various sensors. The YOLO-v8 model, trained with a custom dataset, achieved a stable 90% test accuracy, ensuring reliable classification. The hardware setup, including Pi Cam, Ultrasonic Distance Sensor, and LED, enables seamless device operation, providing real-time security monitoring with motion detection, image capture, and facial recognition.

The system’s implementation features an intuitive user interface using Express.js, ensuring a user-friendly experience. The application incorporates security measures to ensure user authentication and protect sensitive data. The system requires users to register before accessing the server to check captured images, enhancing overall security. Continuous operation and real-time monitoring are achieved through features like motion detection, image capture, and facial recognition. Future considerations include addressing model limitations and exploring cost-effective alternatives.

In conclusion, the Home Security Detection System showcases the practical application of computer vision and IoT technologies for efficient home security. Its robust features and real-time monitoring capabilities offer a reliable solution for residential security, with potential for further advancements in smart home security.

References

[1] https://www.kaggle.com/datasets/justin900429/3d-printer-defected-dataset

[2] https://docs.roboflow.com/datasets/image-preprocessing

[3] https://docs.roboflow.com/datasets/image-augmentation

[4] https://docs.ultralytics.com/modes/train/

[5] https://github.com/ultralytics/ultralytics

[6] https://arxiv.org/abs/1512.00567

[7] https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202

[8] https://www.researchgate.net/publication/335188554/figure/fig2/AS:792153403977733@1565875497541/Module-structure-of-Inception-v1-Inception-v2-and-Inception-v3-a-Inception-v1-b.png

[9] https://www.kaggle.com/competitions/early-detection-of-3d-printing-issues

[10] https://projects.raspberrypi.org/en/projects/physical-computing/12

[11] https://projects.raspberrypi.org/en/projects/rpi-connect-led

[12] https://projects.raspberrypi.org/en/projects/getting-started-with-picamera/2