VIETNAMESE - GERMAN UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Introduction to Information Technology Project

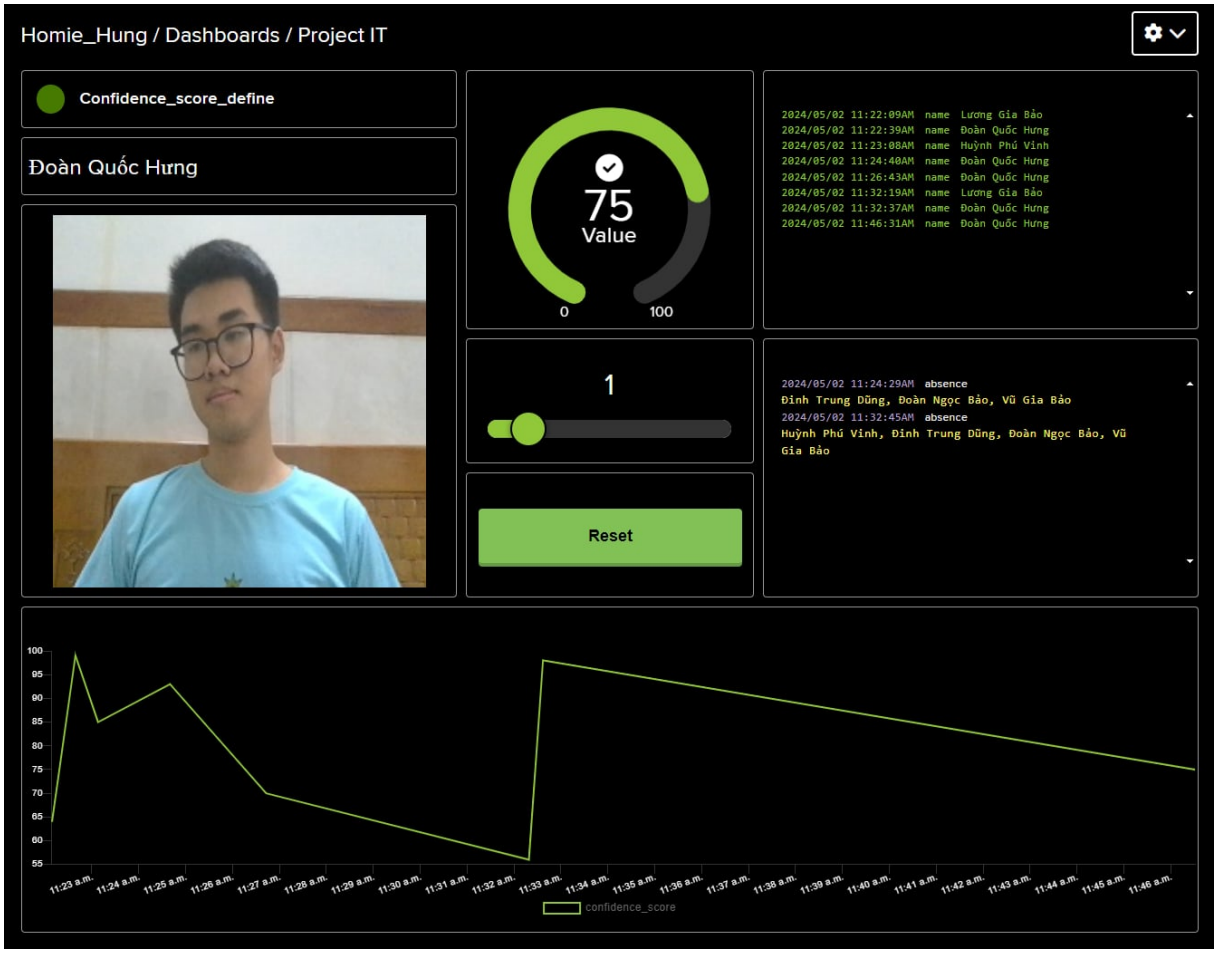
Student 1: Đoàn Quốc Hưng - 10423049
Student 2: Lương Gia Bảo - 10423010
Student 3: Huỳnh Phú Vinh - 10423124
Student 4: Đinh Trung Dũng - 10423132
Student 5: Đoàn Ngọc Bảo - 10423009
Student 6: Vũ Gia Bảo - 10423014

HỒ CHÍ MINH CITY, 05/ 2024

Content

1 Introduction

In a world driven by the rapid pace of technological advancements these days, artificial intelligence (AI) plays a crucial role that impacts various aspects of our daily lives, ranging from facial recognition to medical diagnosis. As an integral part of our Information Technology curriculum, we present a leading-edge project report that deals with the integration of AI facial recognition systems using Google's Teachable Machine, Python programming as well as Adafruit technology. These components have contributed to our growing understanding of the complexities of facial recognition. By doing this, we developed the ability to read the users' faces and maintain attendance records, which has proved the importance of AI integration in educational institutions. The figure below illustrates the outcome of our project showcased on Adafruit IO after successful facial recognition



Hình 1: The result of our project

2 Teachable Machine Overview

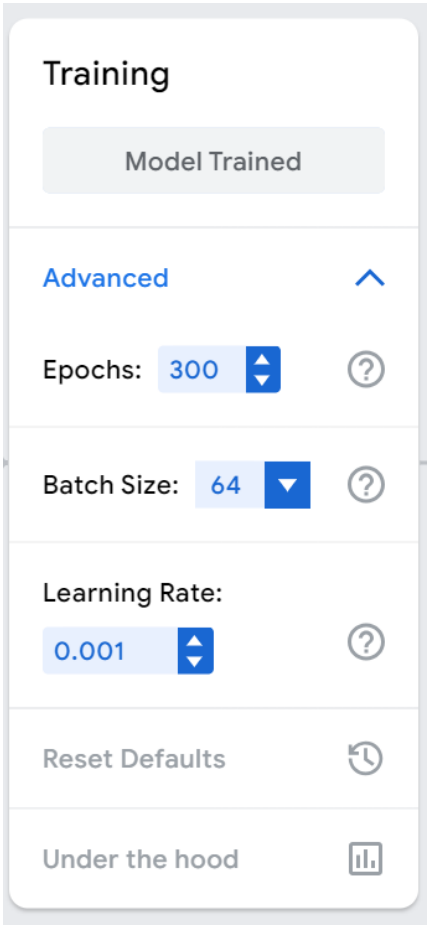
Teachable Machine is a breakthrough of Google Creative Lab's introduction to the space of machine learning and it also is an advance that makes this sphere accessible to everyone. Thanks to its user-friendly interface this web-based application becomes the way of making it intimidating

to create ML models even for beginners or people with no ML experience. The so-called knowledge computer obviates the necessity for prior familiarity in the machine learning domain people can enable the device to figure out the images, sounds and poses. As simple as that, Teachable Machine gives a brick on the wall for those who have no or very basic knowledge, and who want to experience AI directly, with no hardships of coding. After the training of models is done, the created outputs can be used by different types of activities as websites, applications and other artistic projects, so the machine learning technology is accessible to not only the professionals but also the general public in different spheres.

3 Dataset Collection

3.1 Data Collection

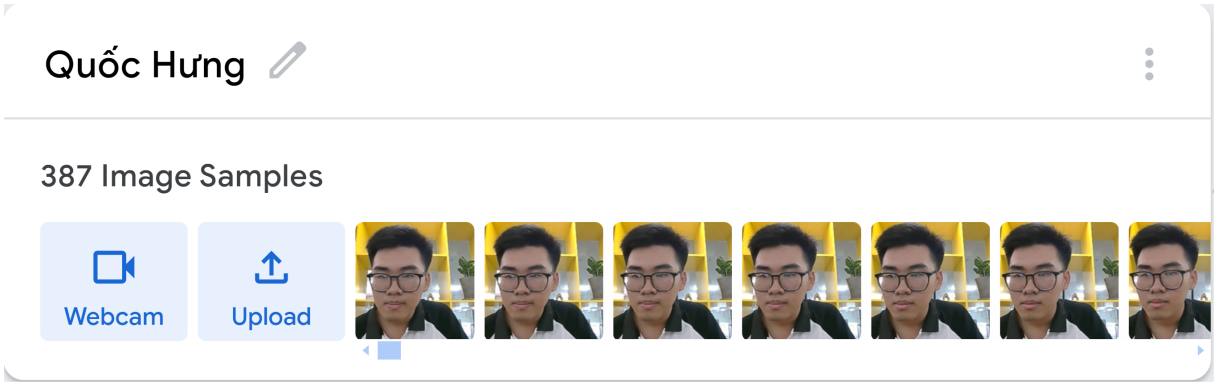
- The data for this project was collected by capturing images of our group members and uploading them to Google Teachable Machine to train the AI model. The data is categorized into six classes, each representing a different member.
- To ensure the AI could detect with high confidence score, our group adjusted the epoch to 300 and the batch size to 64. The larger the batch size, the better the model will learn.



Hình 2: Batch size and number of epochs used for the training process

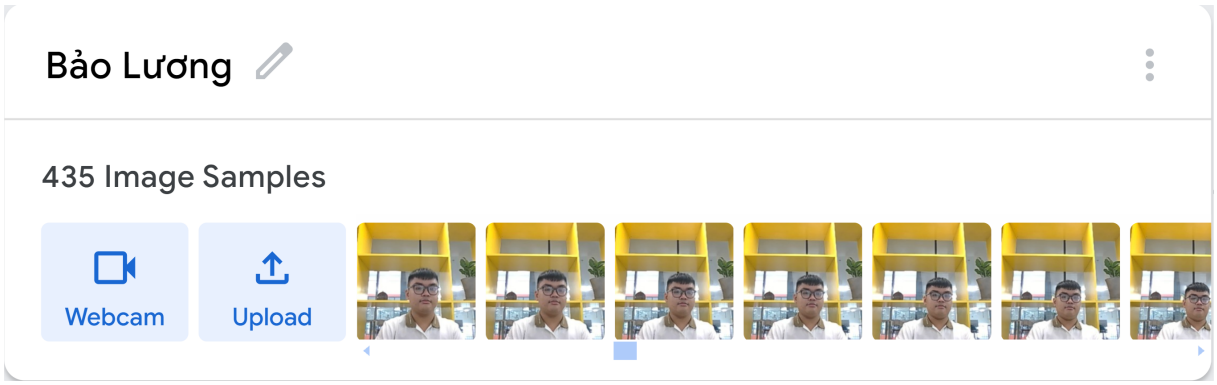
3.1.1 Class 1

387 images, capturing various facial angles of Quốc Hưng, were taken as samples for Class 1.



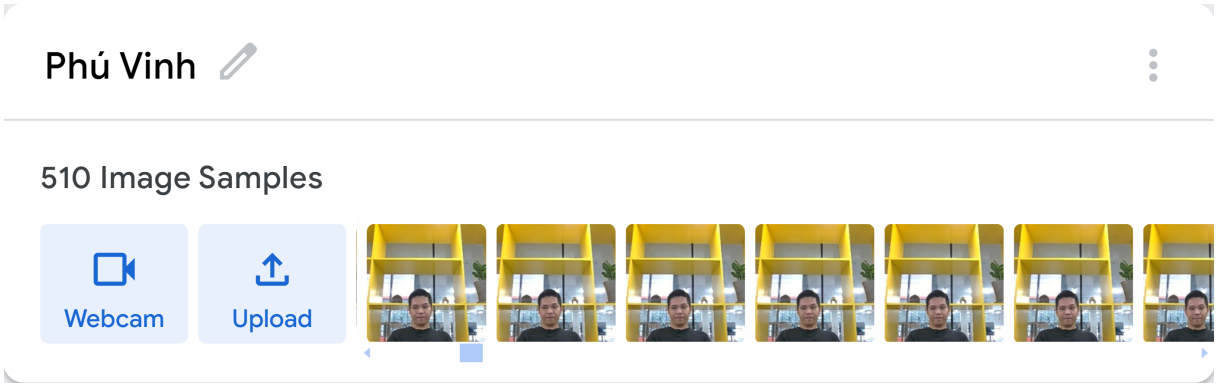
3.1.2 Class 2

435 images, capturing various facial angles of Bảo Lương, were taken as samples for Class 2.



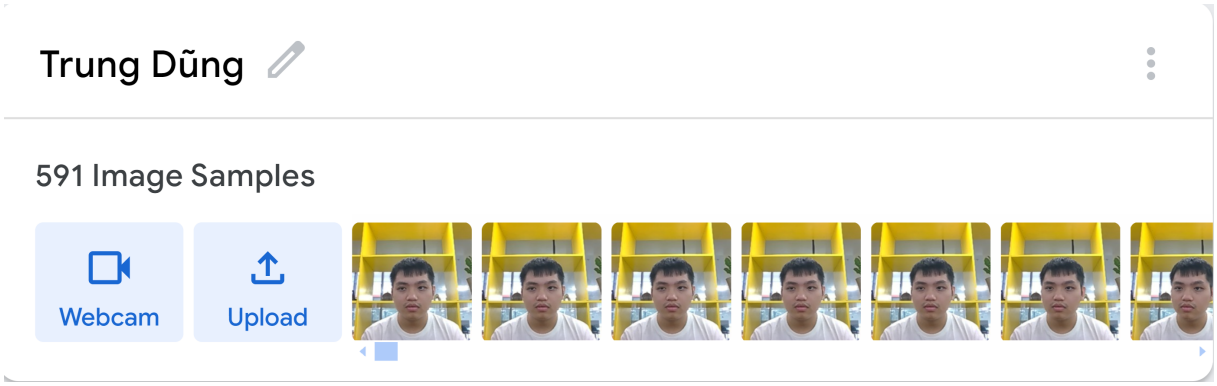
3.1.3 Class 3

510 images, capturing various facial angles of Phú Vinh, were taken as samples for Class 3.



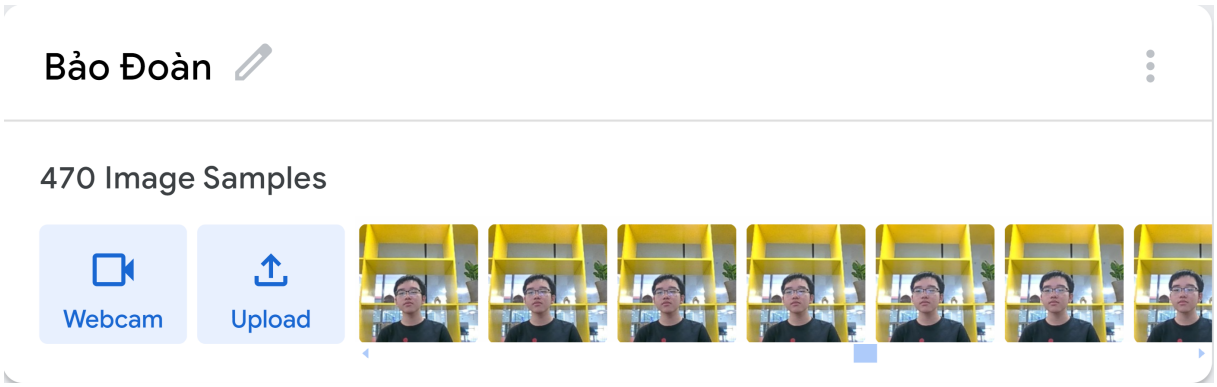
3.1.4 Class 4

591 images, capturing various facial angles of Trung Dũng, were taken as samples for Class 4.



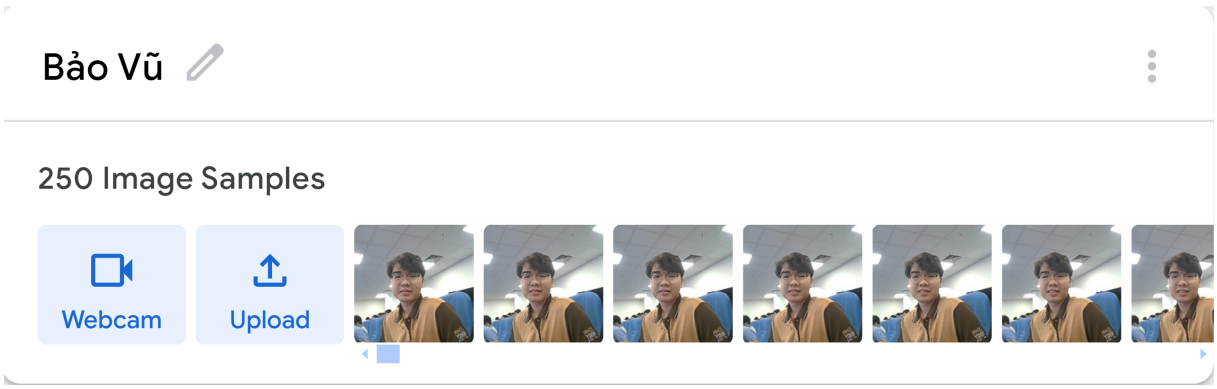
3.1.5 Class 5

470 images, capturing various facial angles of Bảo Đoàn, were taken as samples for Class 5.



3.1.6 Class 6

250 images, capturing various angles of Bảo Vũ, were taken as samples for Class 6.

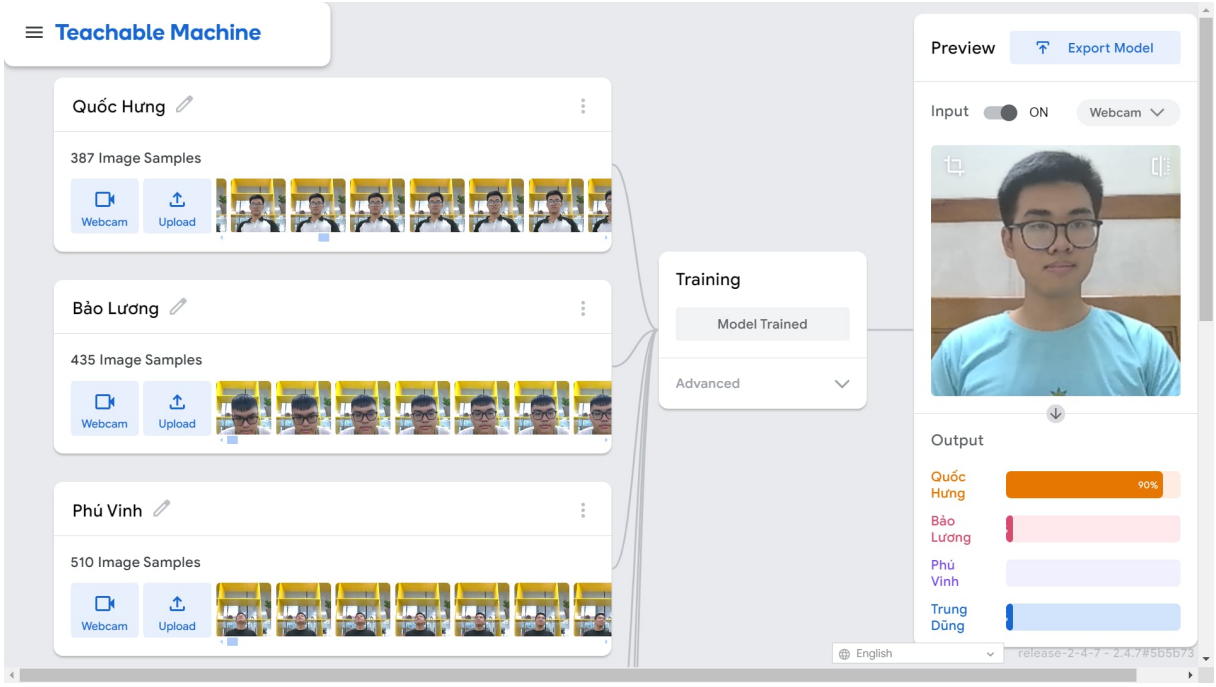


4 AI Training

After collecting the data and training the AI model, the system is now capable of accurately identifying individual members. Here are the screenshots displaying the obtained results:

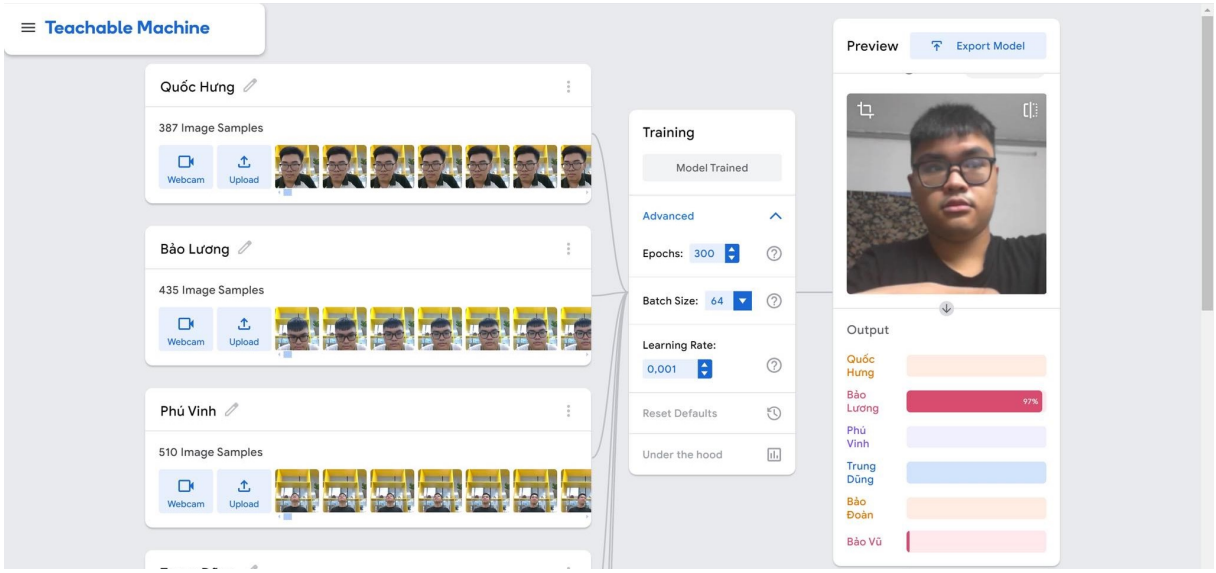
4.1 Result 1: Quốc Hưng

The result demonstrates that the model has identified the facial features of Quốc Hưng:



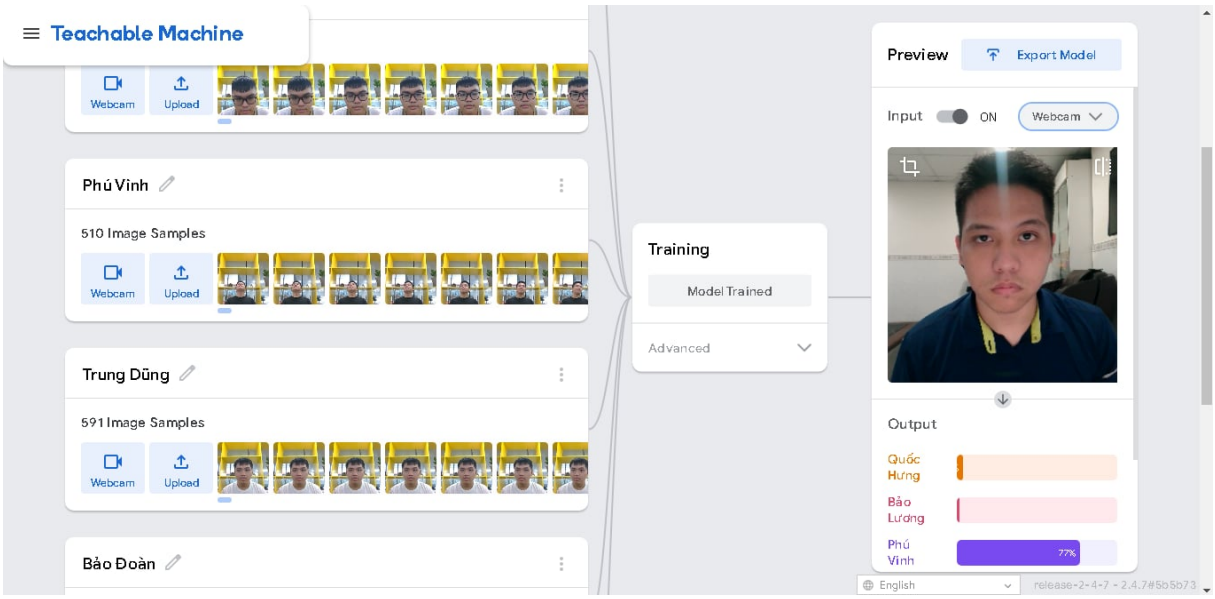
4.2 Result 2: Bảo Lương

The result demonstrates that the model has identified the facial features of Bảo Lương:



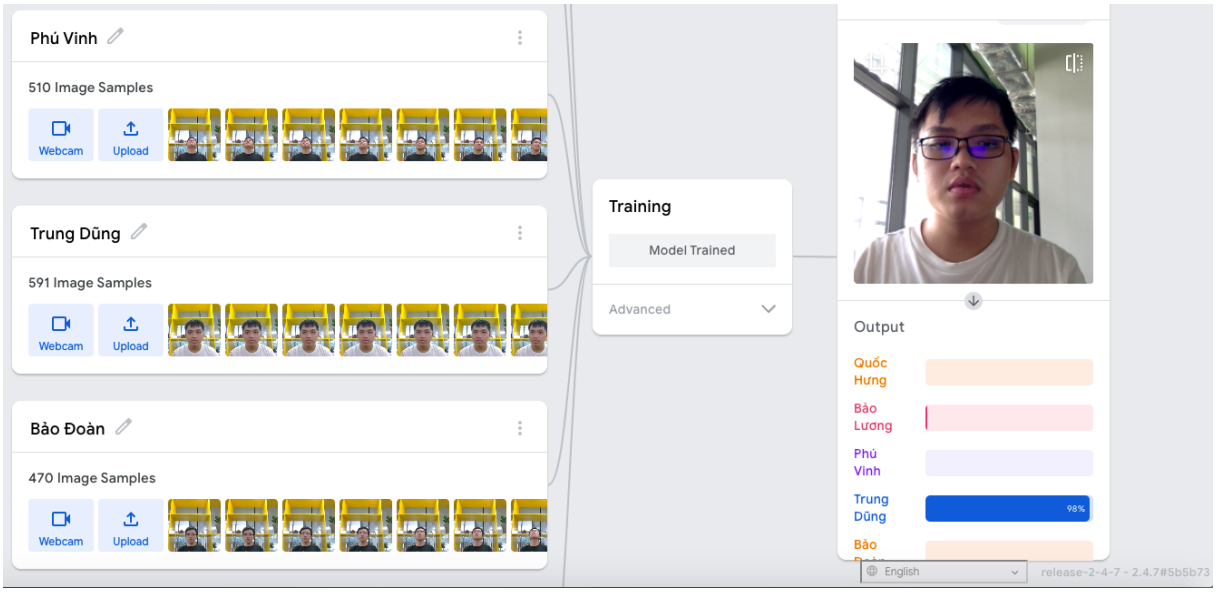
4.3 Result 3: Phú Vinh

The result demonstrates that the model has identified the facial features of Phú Vinh:



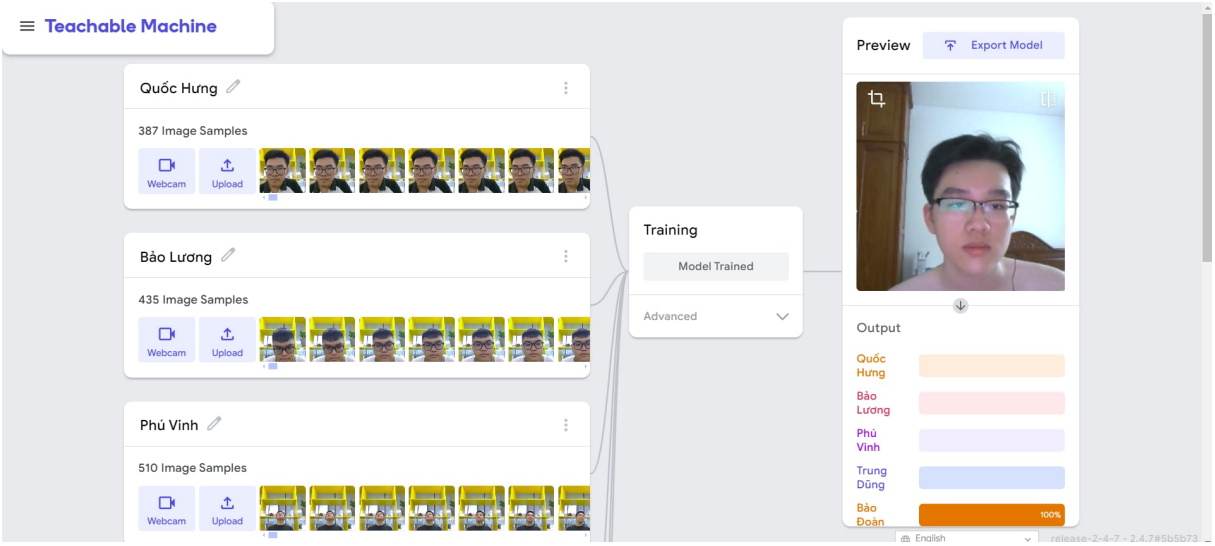
4.4 Result 4: Trung Dũng

The result demonstrates that the model has identified the facial features of Trung Dũng:



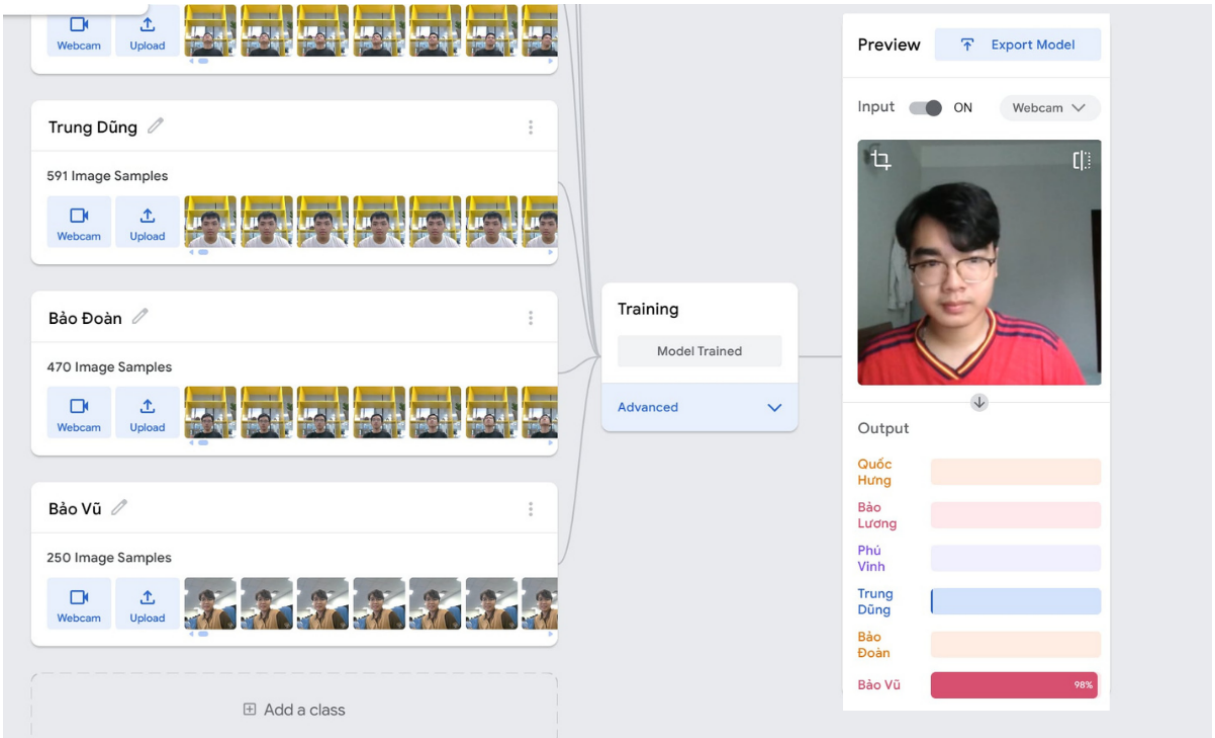
4.5 Result 5: Bảo Đoàn

The result demonstrates that the model has identified the facial features of Bảo Đoàn:



4.6 Result 6: Bảo Vũ

The result demonstrates that the model has identified the facial features of Bảo Vũ:

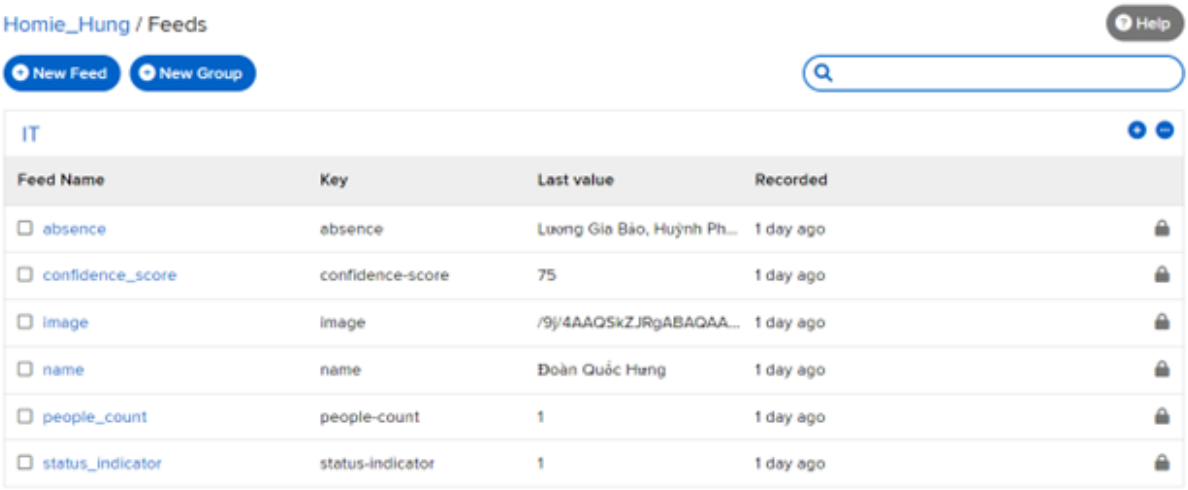


5 Adafruit

5.1 Adafruit Feeds

Feeds are used to store data. We have created 6 feeds on the Adafruit IO server. Each feed has its own name and data channel.

- Absence: Count the number of absent members after completing the program
- Confidence Score: Measure the confidence score of each member
- Image: Take the image of each member when scanning
- Name: Print out the name of that member
- People Count: Count the number of people when scanning
- Status Indicator: to measure how high or low the confidence score is. If confidence score is above 70, it gives a value of 1, otherwise it gives a value of 0

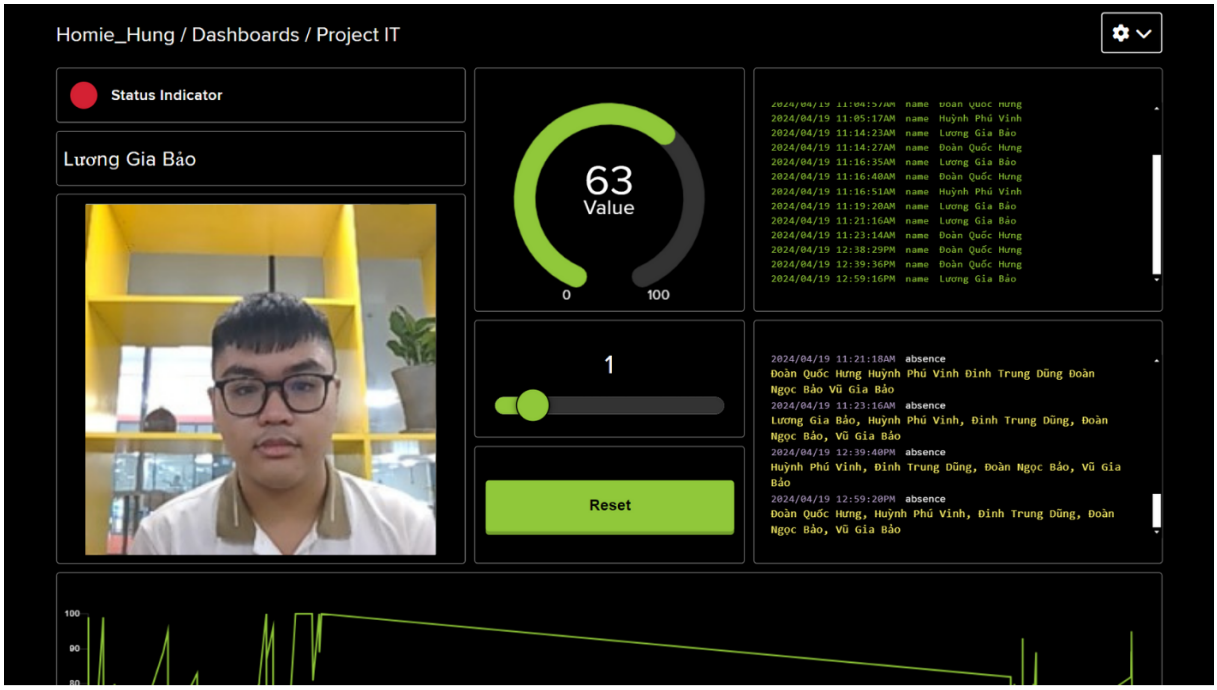


Hình 3: Feeds on Adafruit IO

5.2 Adafruit Dashboard

- We use the dashboard in order to illustrate all essential information of the system. Through the dashboard, we can effectively showcase information in various forms such as momentary buttons, sliders, texts, streams, photos, line charts, icons, and indicators.
- The momentary button has the function of resetting the number of members that the machine has scanned after each run of the program. Then the slider will return to 0 and the system will continue to calculate the number of people who were scanned. After scanning a person's face, the image block is responsible for capturing the scanned person's face and displaying that person's name in the text block.

- Depending on the level of confidence score, it gives different indicator values as mentioned in the Adafruit Feeds section. The first block stream displays the time the program scanned someone, the second block stream displays the number of absent members after completing the program. Finally, the line chart block displays confidence score values over time.



Hình 4: Dashboard on Adafruit IO

6 Python Source Code

The python source code of our project is presented in this part:

Listing 1: Python Source Code

```
1  import cv2
2  import numpy as np
3  import time
4  import sys
5  import base64
6  import os
7  from keras.models import load_model
8  from Adafruit_IO import MQTTClient, Client
9
10 # Your Adafruit IO credentials
11 ADAFRUIT_IO_USERNAME = 'Homie_Hung'
12 ADAFRUIT_IO_KEY = 'aio_jsRH99ZX7Ayn3HIJlvpGrmcTNvZQ'
13
14 # Adafruit IO feed names
```

```
AIO_FEED_ID = "name"
CONFIDENCE_FEED = "confidence_score"
COUNT_FEED = "people_count"
STATUS_FEED = "status_indicator"
ABSENCE_FEED = "absence"

# Initialize Adafruit IO Client
aio = MQTTClient(ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)

# Define the folder where the images are saved
image_folder = "D:/Project IT for me/Project for my laptop/images"

# Initialize camera
camera = cv2.VideoCapture(0)

# Load the model
model = load_model("keras_Model.h5", compile=False)

# Load the labels
with open("labels.txt", "r", encoding="utf-8") as file:
    class_names = file.readlines()

# Dictionary to track recognized faces and their confidence scores
recognized_faces = {}

# Variable to count the number of people
people_count = 0

# Variable to keep track of whether to continue detecting faces
continue_detection = True

# Function to capture face using webcam
def capture_face():
    ret, frame = camera.read()
    return frame

# Function to save image to a file
def save_image(image, filename):
    with open(os.path.join(image_folder, filename), 'wb') as file:
        file.write(image)

# Connect callback functions
def connected(client):
    print("Connected to Adafruit IO!")

```

```
60 def disconnected(client):
61     print("Disconnected from Adafruit IO!")
62     sys.exit(1)
63 
64 def message(client, feed_id, payload):
65     print("Received message from feed {}: {}".format(feed_id, payload)↩
       )
66 
67 # Set up Adafruit IO callbacks
68 aio.on_connect = connected
69 aio.on_disconnect = disconnected
70 aio.on_message = message
71 
72 # Connect to Adafruit IO
73 aio.connect()
74 
75 # Function to check for existing images in the folder
76 def check_existing_images():
77     existing_images = []
78     try:
79         for filename in os.listdir(image_folder):
80             if filename.endswith(".jpg"):
81                 existing_images.append(os.path.join(image_folder, ↩
                       filename))
82     except FileNotFoundError:
83         pass
84     return existing_images
85 
86 # Variable to store the detected labels
87 detected_labels = []
88 
89 while continue_detection:
90     # Capture webcam image
91     ret, image = camera.read()
92 
93     # Display the captured frame
94     cv2.imshow('Webcam Feed', image)
95 
96     # Wait for a short amount of time to allow OpenCV to process GUI ↩
       events
97     cv2.waitKey(1)
98 
99     # Resize image
100    image_resized = cv2.resize(image, (224, 224), interpolation=cv2.↩
       INTER_AREA)
```

```
101
102     # Convert image to numpy array and reshape
103     image_array = np.asarray(image_resized, dtype=np.float32).reshape↩
            (1, 224, 224, 3)
104
105     # Normalize image
106     image_normalized = (image_array / 127.5) - 1
107
108     # Predict
109     prediction = model.predict(image_normalized)
110     index = np.argmax(prediction)
111     class_name = class_names[index].strip()
112
113     # Check if face has been recognized before
114     if class_name not in recognized_faces:
115         recognized_faces[class_name] = True
116
117         # Add detected label to the list
118         detected_labels.append(class_name)
119
120         # Increment the people count
121         people_count += 1
122
123         # Print prediction
124         print("Class:", class_name)
125
126         # Check for black color in the image
127         black_color_detected = np.all(image_resized == [0, 0, 0], axis↩
                =-1).any()
128
129         # Determine status indicator value based on confidence score
130         status_indicator = 1 if prediction[0][index] > 0.7 else 0
131
132         # Generate unique filename based on current timestamp
133         timestamp = int(time.time())
134         image_filename = "captured_face_{}.jpg".format(people_count)
135
136         # Save the resized face image to a file
137         save_image(cv2.imencode('.jpg', image_resized)[1], ↩
                image_filename)
138
139         # Open the captured face image
140         with open(os.path.join(image_folder, image_filename), "rb") as↩
                file:
141             # Read the image data
```

```
142             image_data = file.read()
143
144         # Encode the image data as base64
145         encoded_image = base64.b64encode(image_data)
146
147         # Convert to string
148         image_str = encoded_image.decode('utf-8')
149
150         # Publish data to Adafruit IO feeds
151         try:
152             aio.publish(AIO_FEED_ID, class_name)
153             aio.publish(CONFIDENCE_FEED, str(int(round(prediction[0][↩
                    index] * 100))))
154             aio.publish(COUNT_FEED, str(people_count))
155             aio.publish(STATUS_FEED, str(status_indicator))
156             aio.publish("image", image_str)
157             print("Data uploaded successfully!")
158         except Exception as e:
159             print("Adafruit IO request failed:", e)
160
161         # Ask if user wants to detect more faces
162         while True:
163             choice = input("Do you want to detect more faces? (yes/no)↩
                    : ").lower()
164             if choice == 'no':
165                 continue_detection = False
166                 break
167             elif choice == 'yes':
168                 break
169             else:
170                 print("Invalid choice. Please enter 'yes' or 'no'.")
171
172 # Publish missing labels if any after the user chooses not to detect ↩
       more faces
173 if not continue_detection:
174     missing_labels = ', '.join([label.strip() for label in class_names↩
            if label.strip() not in detected_labels])
175     try:
176         aio.publish(ABSENCE_FEED, missing_labels)
177     except Exception as e:
178         print("Failed to publish missing labels:", e)
179
180 # Release camera and close windows
181 camera.release()
182 cv2.destroyAllWindows()
```

7 Extra features

7.1 Source Code Explanation

Listing 2: Libraries

```
import cv2
import numpy as np
import time
import sys
import base64
import os
from keras.models import load_model
from Adafruit_IO import MQTTClient, Client
```

- cv2: Work with images in Python
- base64: Format the captured images
- Keras: distribute training of deep learning models
- Adafruit IO: a platform for IoT-based applications, to manage data streams over MQTT.
- MQTT client: An instance of the MQTT client is initialized using user credentials, and various feeds are declared to publish and subscribe data like confidence scores and attendance status.

Listing 3: Format Captured Images

```
# Define the folder where the images are saved
image_folder = "D:/Project IT for me/Project for my laptop/images"

# Initialize camera
camera = cv2.VideoCapture(0)

# Load the model
model = load_model("keras_Model.h5", compile=False)

# Load the labels
with open("labels.txt", "r", encoding="utf-8") as file:
    class_names = file.readlines()

# Dictionary to track recognized faces and their confidence scores
recognized_faces = {}

```

```
# Variable to keep track of whether to continue detecting faces
continue_detection = True

# Function to capture face using webcam
def capture_face():
    ret, frame = camera.read()
    return frame

# Function to save image to a file
def save_image(image, filename):
    with open(os.path.join(image_folder, filename), 'wb') as file:
        file.write(image)
```

The code initializes a webcam for capturing video and loads a pre-trained Keras model (‘keras_model.h5‘). It reads class names (labels) from a text file, which presumably correspond to different students or attendance statuses, and saving the captured images to disk. These images can then be processed for face detection and recognition using the loaded model.

Listing 4: Check confidence score level

```
  # Check if face has been recognized before
    if class_name not in recognized_faces:
        recognized_faces[class_name] = True

        # Add detected label to the list
        detected_labels.append(class_name)

        # Increment the people count
        people_count += 1

        # Print prediction
        print("Class:", class_name)

        # Check for black color in the image
        black_color_detected = np.all(image_resized == [0, 0, 0], axis=-1).any()

        # Determine status indicator value based on confidence score
        status_indicator = 1 if prediction[0][index] > 0.7 else 0

        # Generate unique filename based on current timestamp
        timestamp = int(time.time())
        image_filename = "captured_face_{}.jpg".format(people_count)

        # Save the resized face image to a file
```

```
25          save_image(cv2.imencode('.jpg', image_resized)[1], ↩
                image_filename)
```

- The code checks if the detected face has been recognized before by comparing it with the existing entries. If the detected face has not been recognized before, the code marks it as recognized by setting its value to True in the dictionary.
- The detected face is then added to the list and the 'count' variable is incremented to keep track of the number of people detected. Then the code prints out the detected class name, which likely corresponds to the recognized person's identity
- The code determines the status indicator value based on the confidence score. If the confidence score is above 0.7, it sets the status indicator to 1. Otherwise, it sets it to 0. If our confidence score is higher than 0.7, then the image of our faces will be saved to a file in the computer

Listing 5: conditions for scanning

```
1       # Ask if user wants to detect more faces
2         while True:
3             choice = input("Do you want to detect more faces? (yes/no)↩
                  : ").lower()
4             if choice == 'no':
5                 continue_detection = False
6                 break
7             elif choice == 'yes':
8                 break
9             else:
10                print("Invalid choice. Please enter 'yes' or 'no'.")
11
12  # Publish missing labels if any after the user chooses not to detect ↩
        more faces
13  if not continue_detection:
14      missing_labels = ', '.join([label.strip() for label in class_names↩
            if label.strip() not in detected_labels])
15      try:
16          aio.publish(ABSENCE_FEED, missing_labels)
17      except Exception as e:
18          print("Failed to publish missing labels:", e)
```

The program will ask the user if they want to scan the next person face. Pick "Yes" if there is still someone who need to be scanned, the program will continue normally and ask the next person after scanning their face. If the user pick "No" the program will stop scanning face and won't continue to detect. It will then label the person as "missing".

Listing 6: Receiving and Publishing to Adafruit IO

```
# Connect callback functions
def connected(client):
    print("Connected to Adafruit IO!")

def disconnected(client):
    print("Disconnected from Adafruit IO!")
    sys.exit(1)

def message(client, feed_id, payload):
    print("Received message from feed {}: {}".format(feed_id, payload))

# Set up Adafruit IO callbacks
aio.on_connect = connected
aio.on_disconnect = disconnected
aio.on_message = message

# Connect to Adafruit IO
aio.connect()

# Publish data to Adafruit IO feeds
try:
    aio.publish(AIO_FEED_ID, class_name)
    aio.publish(CONFIDENCE_FEED, str(int(round(prediction[0][index] * 100))))
    aio.publish(COUNT_FEED, str(people_count))
    aio.publish(STATUS_FEED, str(status_indicator))
    aio.publish("image", image_str)
    print("Data uploaded successfully!")
except Exception as e:
    print("Adafruit IO request failed:", e)
```

- The connected, disconnected, and message functions are defined as callback functions for the Adafruit IO MQTT client. These functions are called when the client connects to Adafruit IO, disconnects from Adafruit IO, and receives a message from a feed on Adafruit IO, respectively.
- These callback functions are then assigned to the corresponding events of the Adafruit IO MQTT client
- The MQTT client is then connected to Adafruit IO using the connect() method.
- Inside the try block, data is published to various feeds on Adafruit IO using the publish() method of the MQTT client. The data includes the detected class name (class name), con-

fidence score (prediction[0][index]), people count (people count), status indicator (status indicator), and the captured image encoded as a base64 string (image str).

7.2 Sharing Code on GitHub

This project is accessible via GitHub:
`https://github.com/Tony-Downey/IT_RosterCall.project.git`

8 Conclusion

In conclusion, this project aims to check people's attendance through a webcam using both Artificial Intelligence (AI) and the Internet of Things (IoT). By combining advanced facial recognition algorithms with IoT connectivity for real-time data transmission, our project is capable of accurately identifying different individuals and tracking their attendance. To ensure the efficiency of our system, a large amount of data needs to be used, code segments are optimized based on existing data, and a logical algorithm is implemented for the smooth execution of events in the code. Through this project, we have improved our knowledge of AI and IoT gateway technologies, in addition to learning how to use Teachable Machine and Adafruit. By integrating these technologies, we not only learned how to automate attendance checking but also gained more experience for similar projects in the future.