Fall-Watch

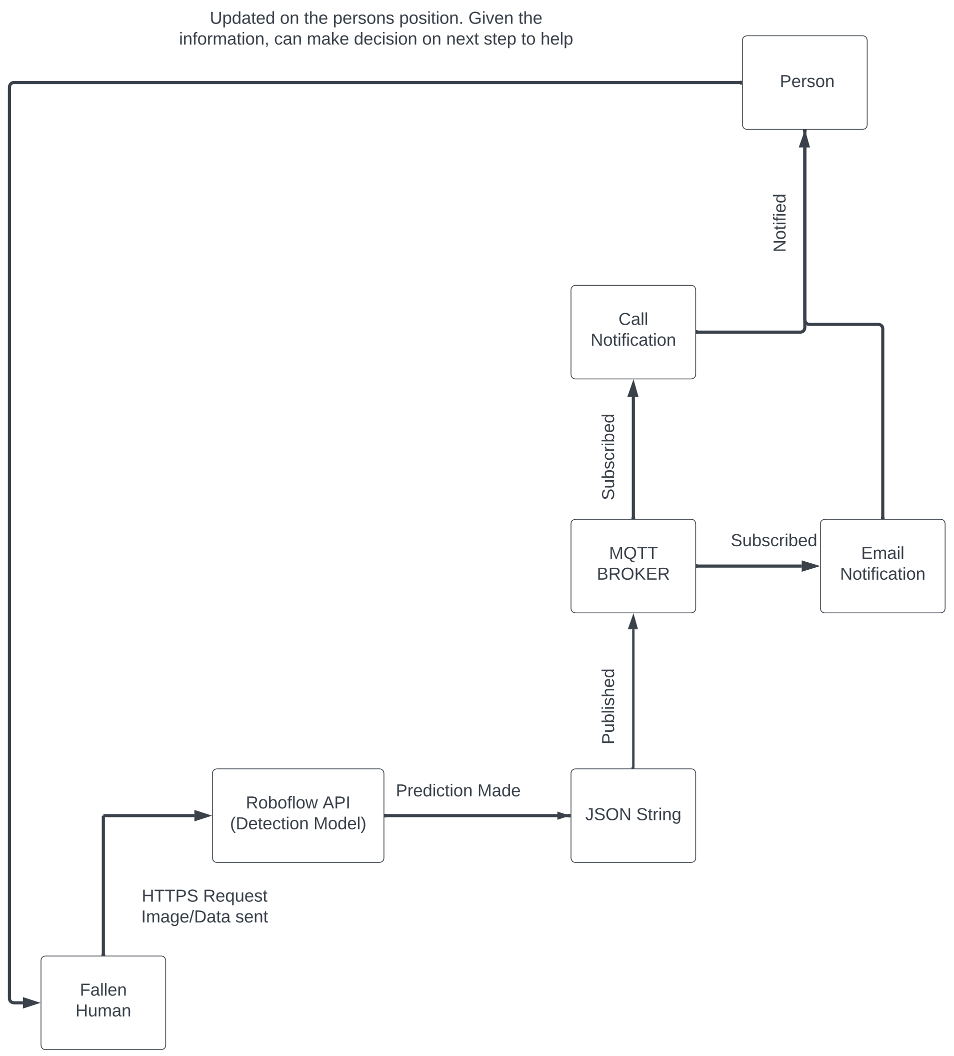
Daniel Lawton

20084608

Concept

- Using an Object detection algorithm (YOLOv5). I wanted to be able to determine the position of a person, whether standing or falling. Publish this data to a MQTT Broker, then use this published code to send notifications.
- The vision was a house of an older person being fitted with cameras. That can monitor the older person that incase they take a fal, rendered immobile. The algorithm will recognize this and be able to ask for help.

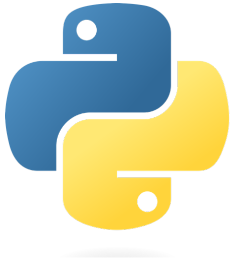
Flow-Chart

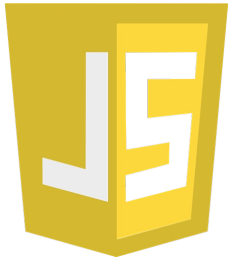

Updated on the persons position. Given the information, can make decision on next step to help
Person
Notified
Call Notification
Subscribed
MQTT BROKER
Subscribed
Email Notification
Published
Roboflow API (Detection Model)
Prediction Made
JSON String
HTTPS Request Image/Data sent
Fallen Human

What I used

- YOLOv5 (Object Bounding-Boxes)
- RoboFlow
- HiveMQ (MQTT Broker)
- Twilio

Languages I Used:





Why RoboFlow?

- RoboFlow is a computer vision platform.
- That offers a web-based service interface that provides:

❑Tools for managing and preprocessing image datasets.

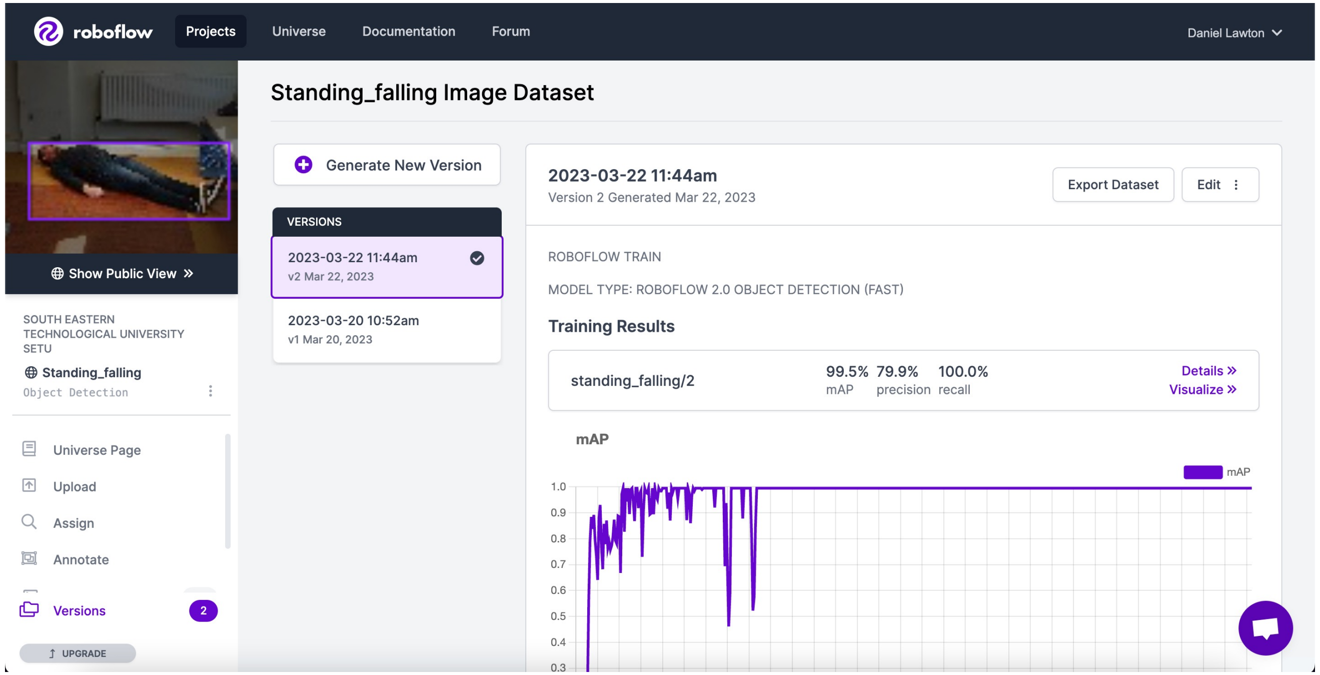
❑Training custom object detection models.

❑Allows for deployment into real-world applications.

- It streamlines the process of training and deploying models, cutting out the time-consuming aspects, enabling developers to worry about the application, and leave the tedious data-management to Roboflow.

What Roboflow looks like

The website makes it easy to create different versions of datasets, allowing for further training of models by uploading more images.



SOUTH EASTERN TECHNOLOGICAL UNIVERSITY SETU

Standing_falling

Object Detection

Standing_falling Image Dataset

VERSIONS

2023-03-22 11:44am
v2 Mar 22, 2023

2023-03-20 10:52am
v1 Mar 20, 2023

2023-03-22 11:44am
Version 2 Generated Mar 22, 2023

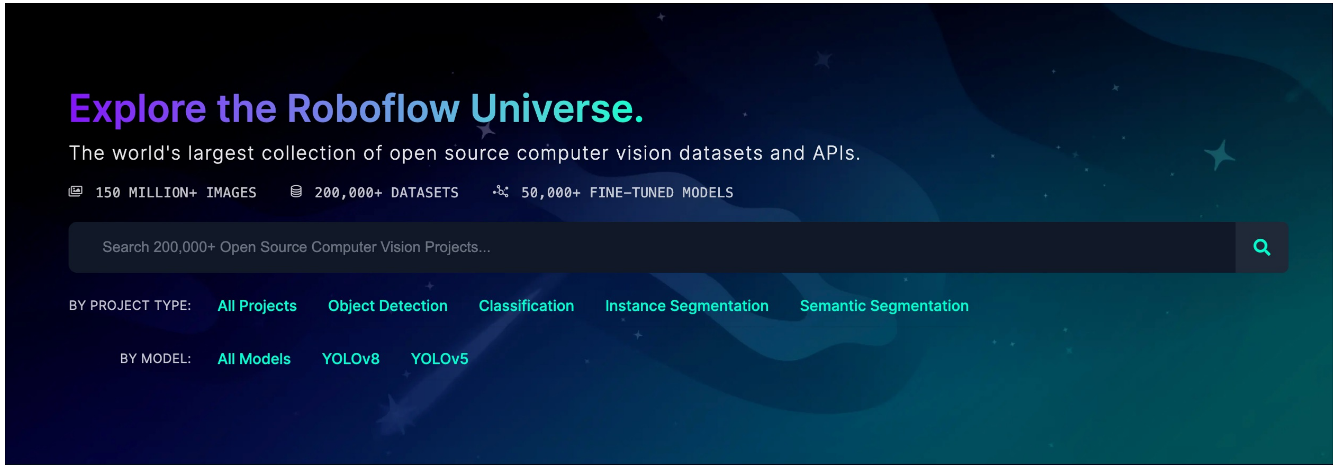
ROBOFLOW TRAIN

MODEL TYPE: ROBOFLOW 2.0 OBJECT DETECTION (FAST)

Training Results

	mAP	precision	recall
standing_falling/2	99.5%	79.9%	100.0%

Roboflow Universe is a library of datasets and pre-trained models that are open to anyone to take and build upon without having to start from scratch on their own models.



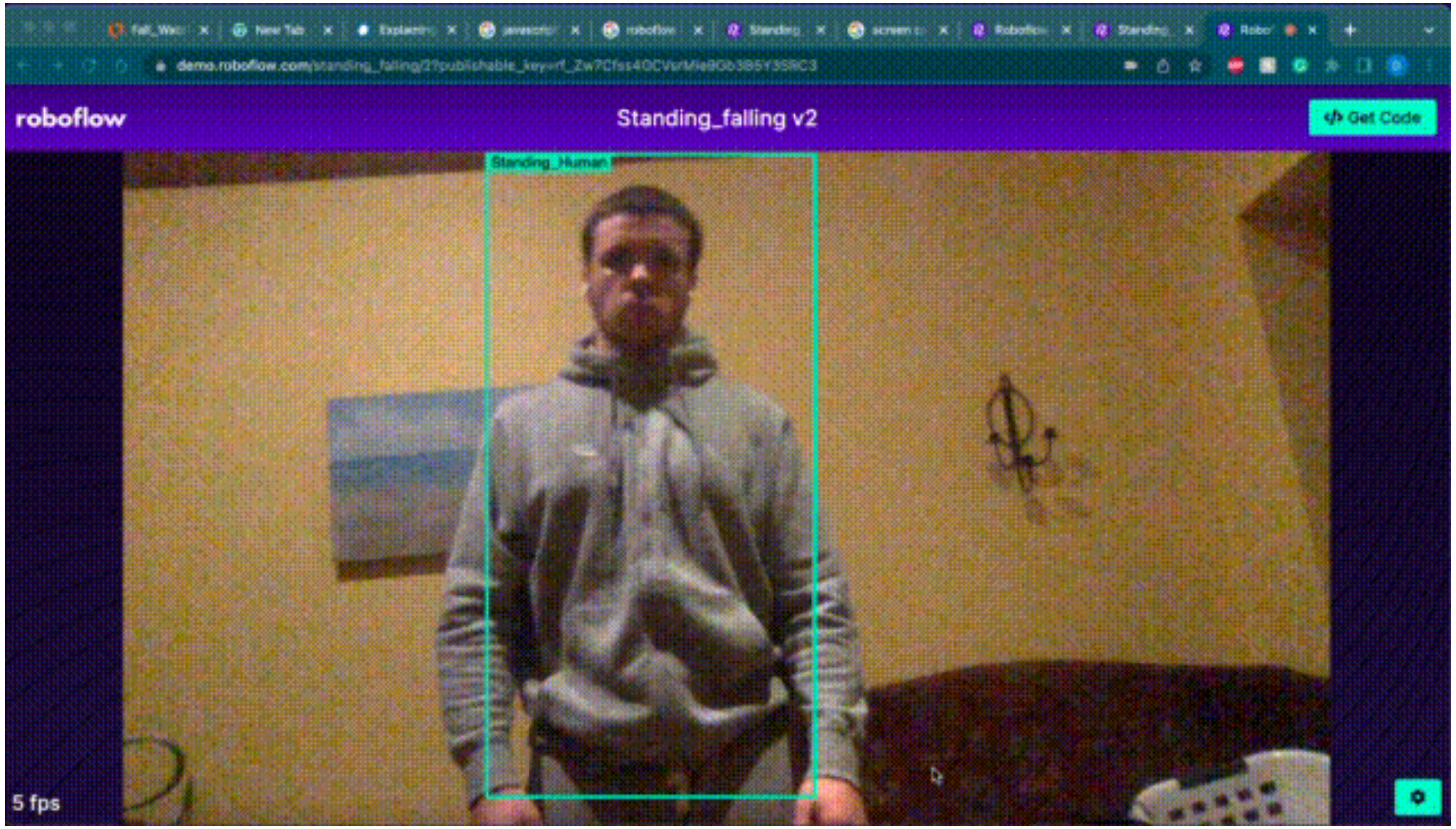
Explore the Roboflow Universe.

The world's largest collection of open source computer vision datasets and APIs.

150 MILLION+ IMAGES 200,000+ DATASETS 50,000+ FINE-TUNED MODELS

YOLOv5 (OBB)

- Standing for (You Only Look Once version 5) object detection algorithm using oriented Bounding Boxes
- YOLOv5 can be used to detect a wide range of objects such as people, animals, cars and other moving objects.


demo.roboflow.com/standing_falling/2?publishable_key=rf_Zw7Cfsx4OCVsrMie9Gb3B5Y3SRC3
roboflow
Standing_falling v2
</> Get Code
Standing_Human
5 fps

Steps to train model

Create a dataset

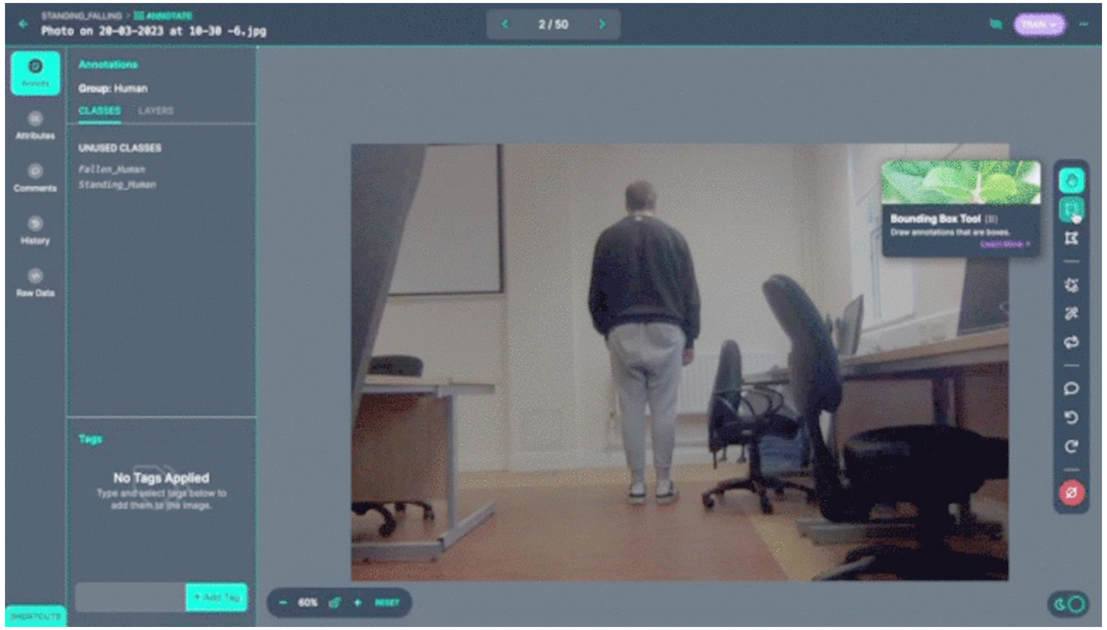
Augment your Data

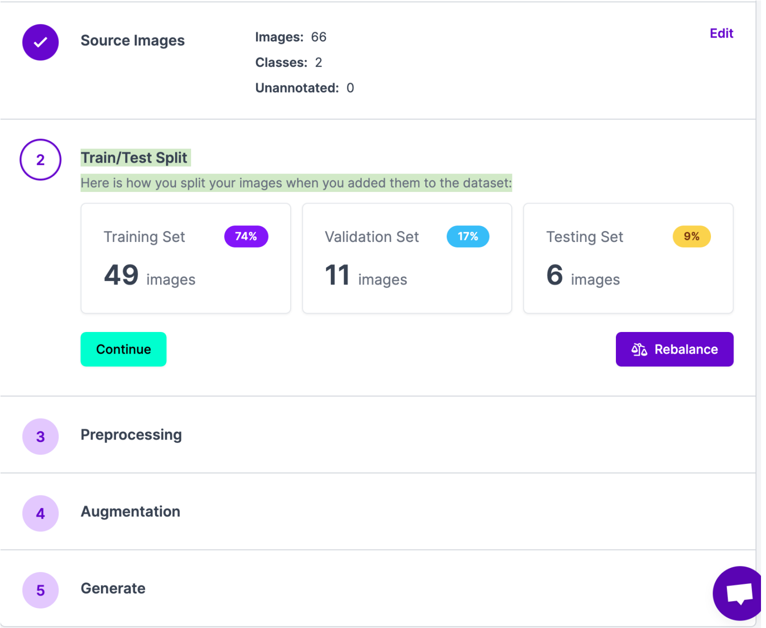
Configure the model (YOLOv4,YOLOv5)

Train your model

Evaluate your model

Export your model to be used in applications





✓ Source Images
Images: 66
Classes: 2
Unannotated: 0
Edit

2 Train/Test Split

Here is how you split your images when you added them to the dataset:

Training Set 74%	Validation Set 17%	Testing Set 9%
49 images	11 images	6 images

Continue
Rebalance

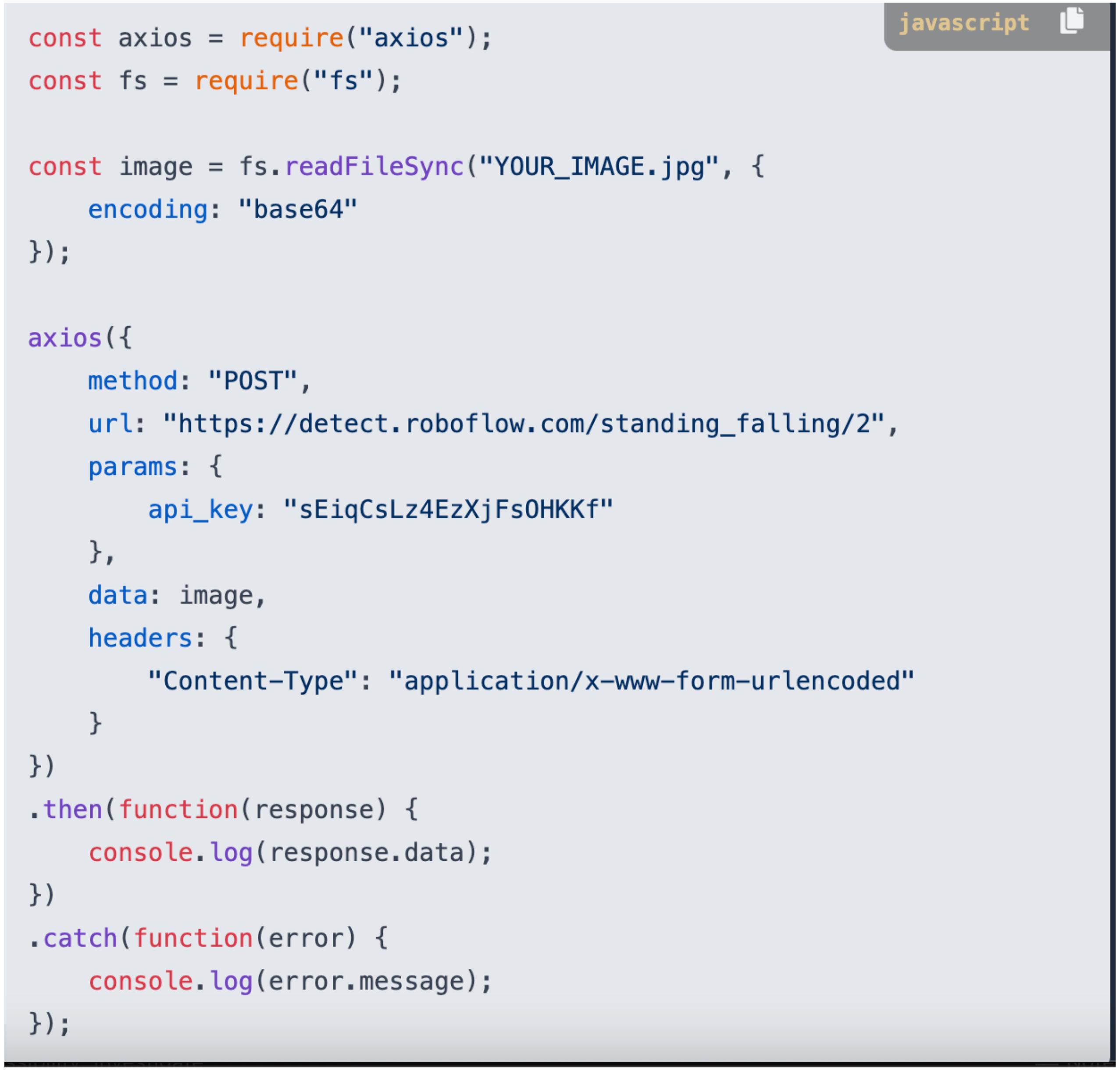
3 Preprocessing

4 Augmentation

5 Generate

Applying the Model

After training and testing the model in Roboflow, the trained model can be deployed to a production environment where it can be used to make predictions on new data.

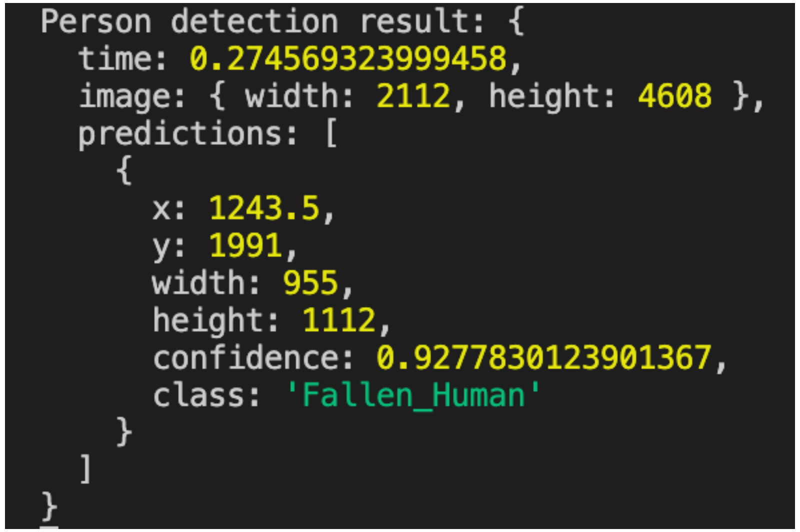


```javascript
const axios = require("axios");
const fs = require("fs");

const image = fs.readFileSync("YOUR_IMAGE.jpg", {
    encoding: "base64"
});

axios({
    method: "POST",
    url: "https://detect.roboflow.com/standing_falling/2",
    params: {
        api_key: "sEiqCsLz4EzXjFsOHKKf"
    },
    data: image,
    headers: {
        "Content-Type": "application/x-www-form-urlencoded"
    }
})
.then(function(response) {
    console.log(response.data);
})
.catch(function(error) {
    console.log(error.message);
});
```

Making Predictions

Image of a Fallen Human ->

Posted to Roboflow API ->



```
Person detection result: {
  time: 0.274569323999458,
  image: { width: 2112, height: 4608 },
  predictions: [
    {
      x: 1243.5,
      y: 1991,
      width: 955,
      height: 1112,
      confidence: 0.9277830123901367,
      class: 'Fallen_Human'
    }
  ]
}
```

Prediction made returned as JSON String

MQTT

- MQTT (Message Queuing Telemetry Transport) is a messaging protocol created by IBM that sends Telemetry data from device to a broker.
- MQTT is both lightweight in bandwidth and its code footprint, using a publish/subscribe mechanism.

Being open source, there are many MQTT brokers available.

mosquitto

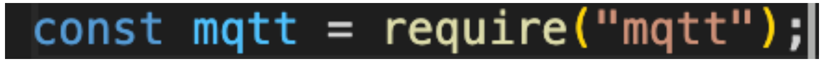
HIVEMQ

EMQX

AWS IoT

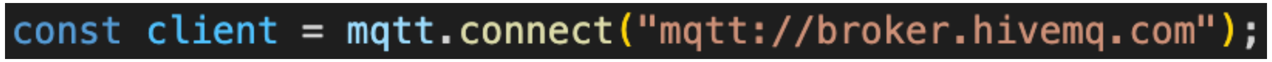
Implementing MQTT

Installing the appropriate MQTT client library.



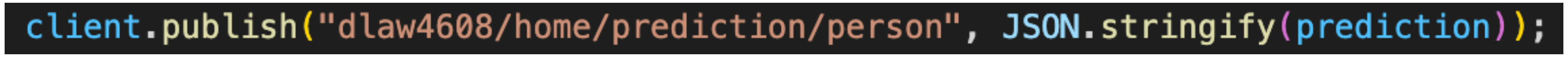
```
const mqtt = require("mqtt");
```

Connect to an MQTT



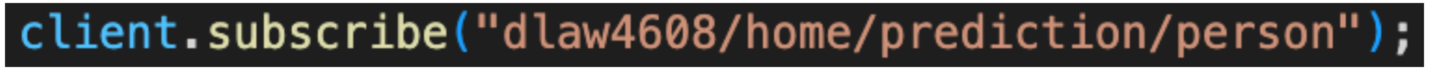
```
const client = mqtt.connect("mqtt://broker.hivemq.com");
```

Publish message to topic



```
client.publish("dlaw4608/home/prediction/person", JSON.stringify(prediction));
```

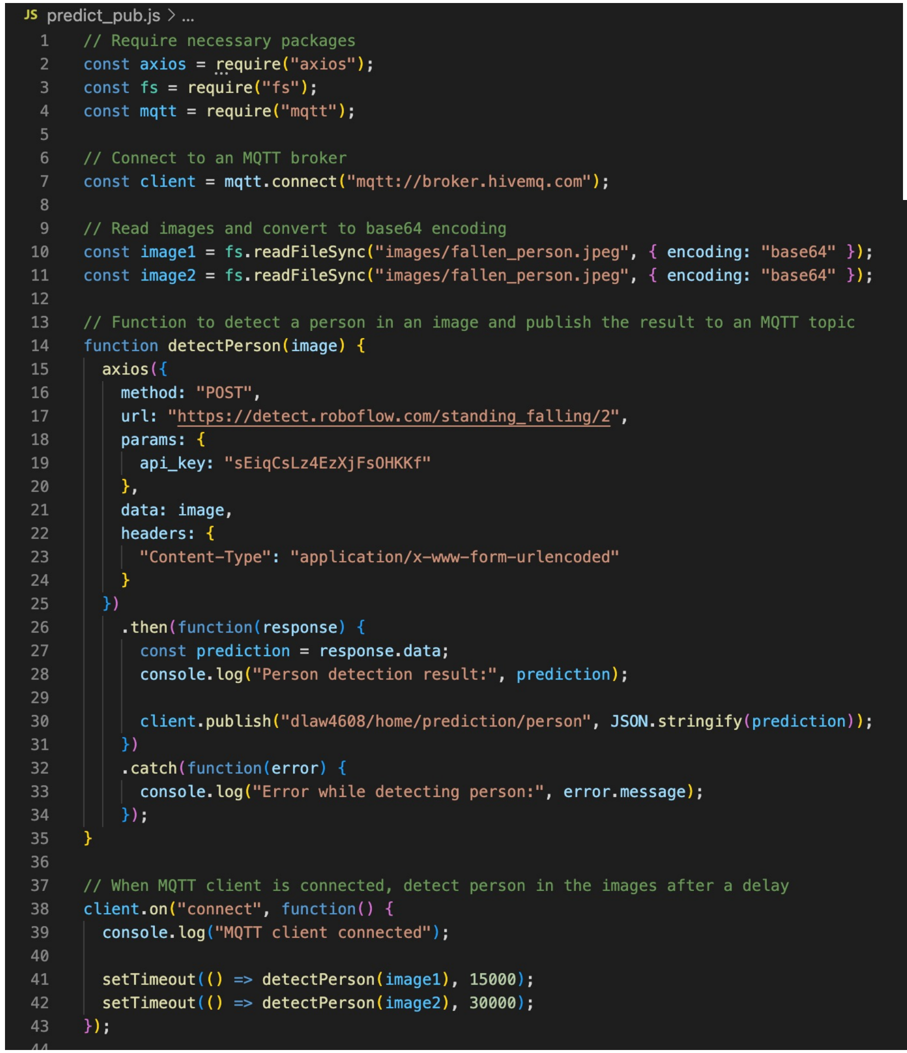
Subscribe to same topic to receive message



```
client.subscribe("dlaw4608/home/prediction/person");
```

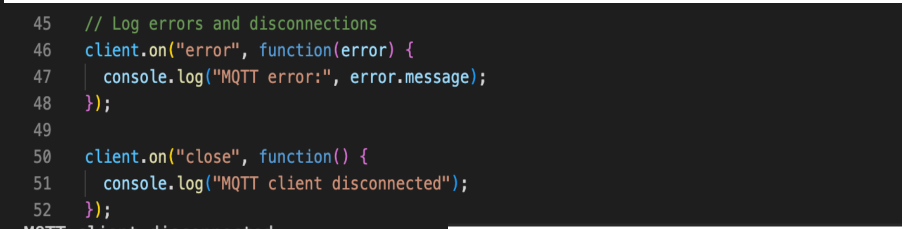
Publishing Predictions to Broker

The Code:

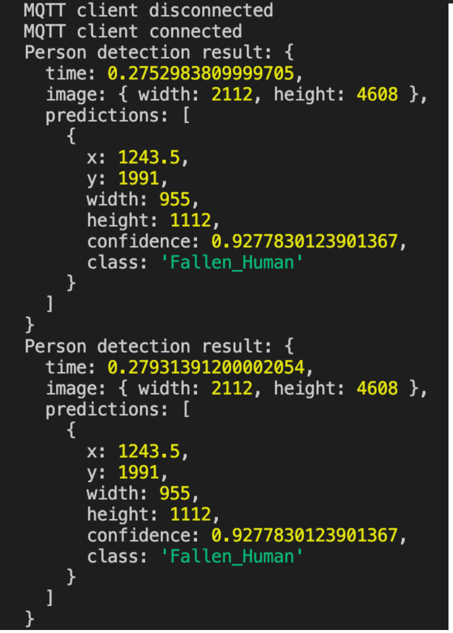


```
// Require necessary packages
const axios = require("axios");
const fs = require("fs");
const mqtt = require("mqtt");

// Connect to an MQTT broker
const client = mqtt.connect("mqtt://broker.hivemq.com");

// Read images and convert to base64 encoding
const image1 = fs.readFileSync("images/fallen_person.jpeg", { encoding: "base64" });
const image2 = fs.readFileSync("images/fallen_person.jpeg", { encoding: "base64" });

// Function to detect a person in an image and publish the result to an MQTT topic
function detectPerson(image) {
  axios({
    method: "POST",
    url: "https://detect.roboflow.com/standing_falling/2",
    params: {
      api_key: "sEiqCsLz4EzXjFsOHKKf"
    },
    data: image,
    headers: {
      "Content-Type": "application/x-www-form-urlencoded"
    }
  })
    .then(function(response) {
      const prediction = response.data;
      console.log("Person detection result:", prediction);

      client.publish("dlaw4608/home/prediction/person", JSON.stringify(prediction));
    })
    .catch(function(error) {
      console.log("Error while detecting person:", error.message);
    });
}

// When MQTT client is connected, detect person in the images after a delay
client.on("connect", function() {
  console.log("MQTT client connected");

  setTimeout(() => detectPerson(image1), 15000);
  setTimeout(() => detectPerson(image2), 30000);
});
```

The next step was to publish predictions made by the Roboflow API to an MQTT broker.



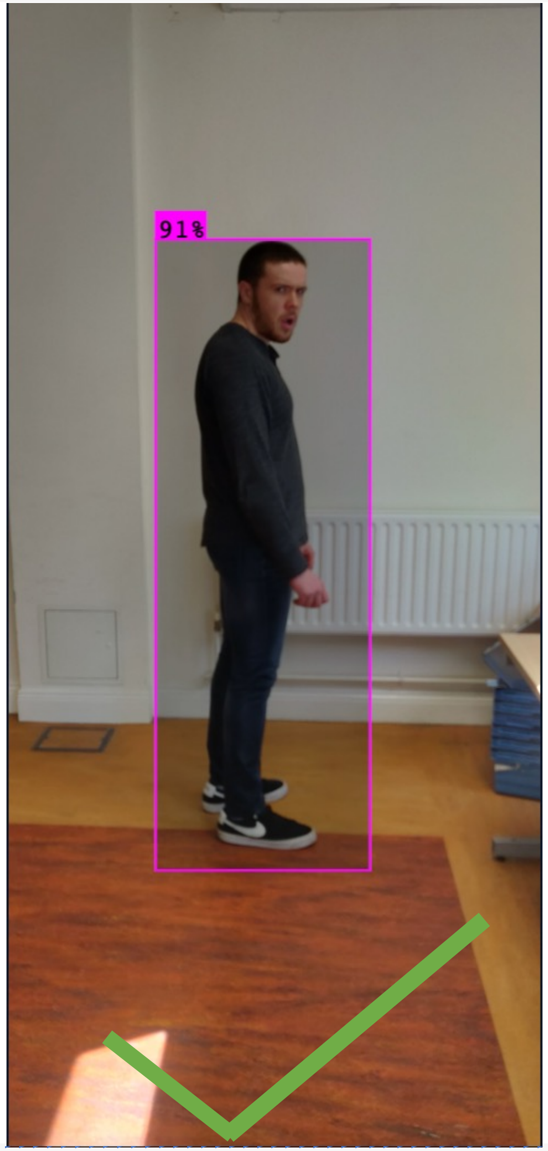
```
// Log errors and disconnections
client.on("error", function(error) {
  console.log("MQTT error:", error.message);
});

client.on("close", function() {
  console.log("MQTT client disconnected");
});
```

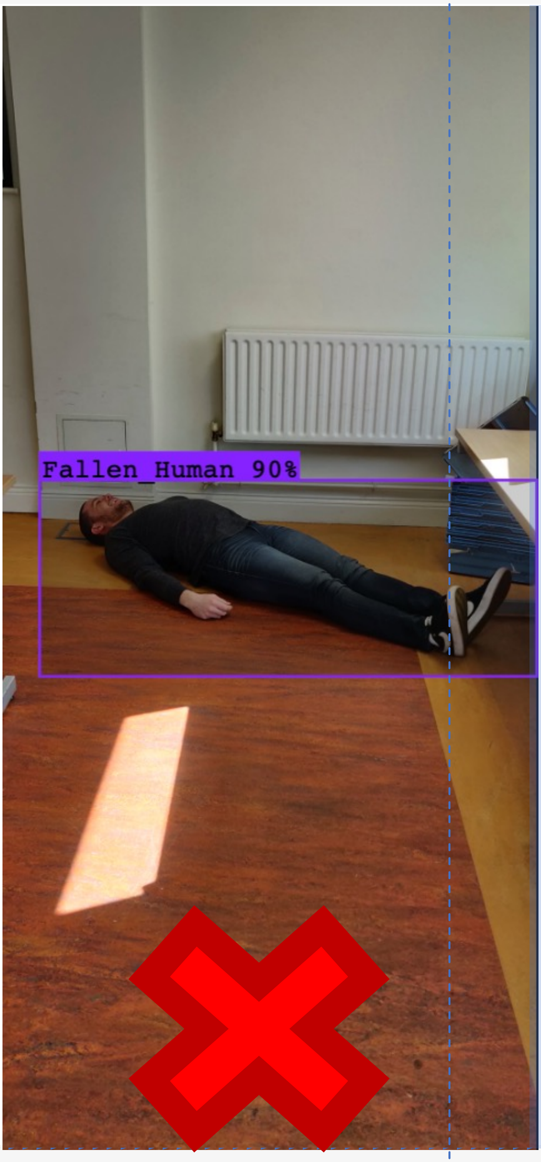


```
MQTT client disconnected
MQTT client connected
Person detection result: {
  time: 0.2752983809999705,
  image: { width: 2112, height: 4608 },
  predictions: [
    {
      x: 1243.5,
      y: 1991,
      width: 955,
      height: 1112,
      confidence: 0.9277830123901367,
      class: 'Fallen_Human'
    }
  ]
}
Person detection result: {
  time: 0.27931391200002054,
  image: { width: 2112, height: 4608 },
  predictions: [
    {
      x: 1243.5,
      y: 1991,
      width: 955,
      height: 1112,
      confidence: 0.9277830123901367,
      class: 'Fallen_Human'
    }
  ]
}
```

Notification

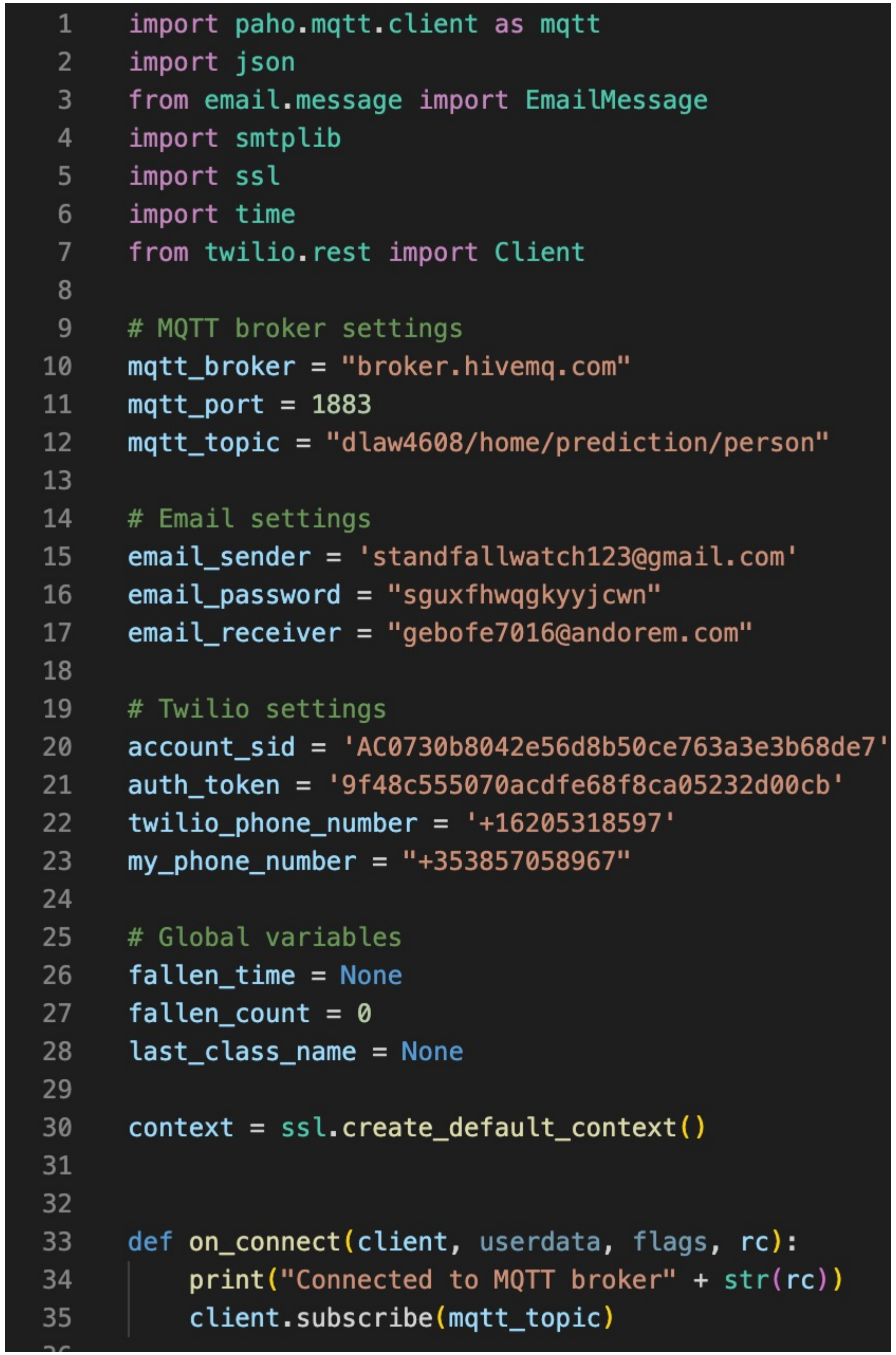
- The idea was, depending on the image predictions published, to send notifications using Email and Twilio.
- I created simulated scenarios that would represent how Fall Watch would function like in real life scenarios.
- The first step in completing this was creating a separate python Script which subscribes to the same topic the predictions are being published to and depending on the class found in the JSON string, the email code will decide what to send.





The Code Breakdown

- The start of the code connects to an MQTT broker using the Paho MQTT client library, that subscribes to a topic on that broker.
- The on_connect function is a callback function that will be called when the client successfully connects to the broker.
- In MQTT, the specific topic is used to route messages to the correct subscribers.

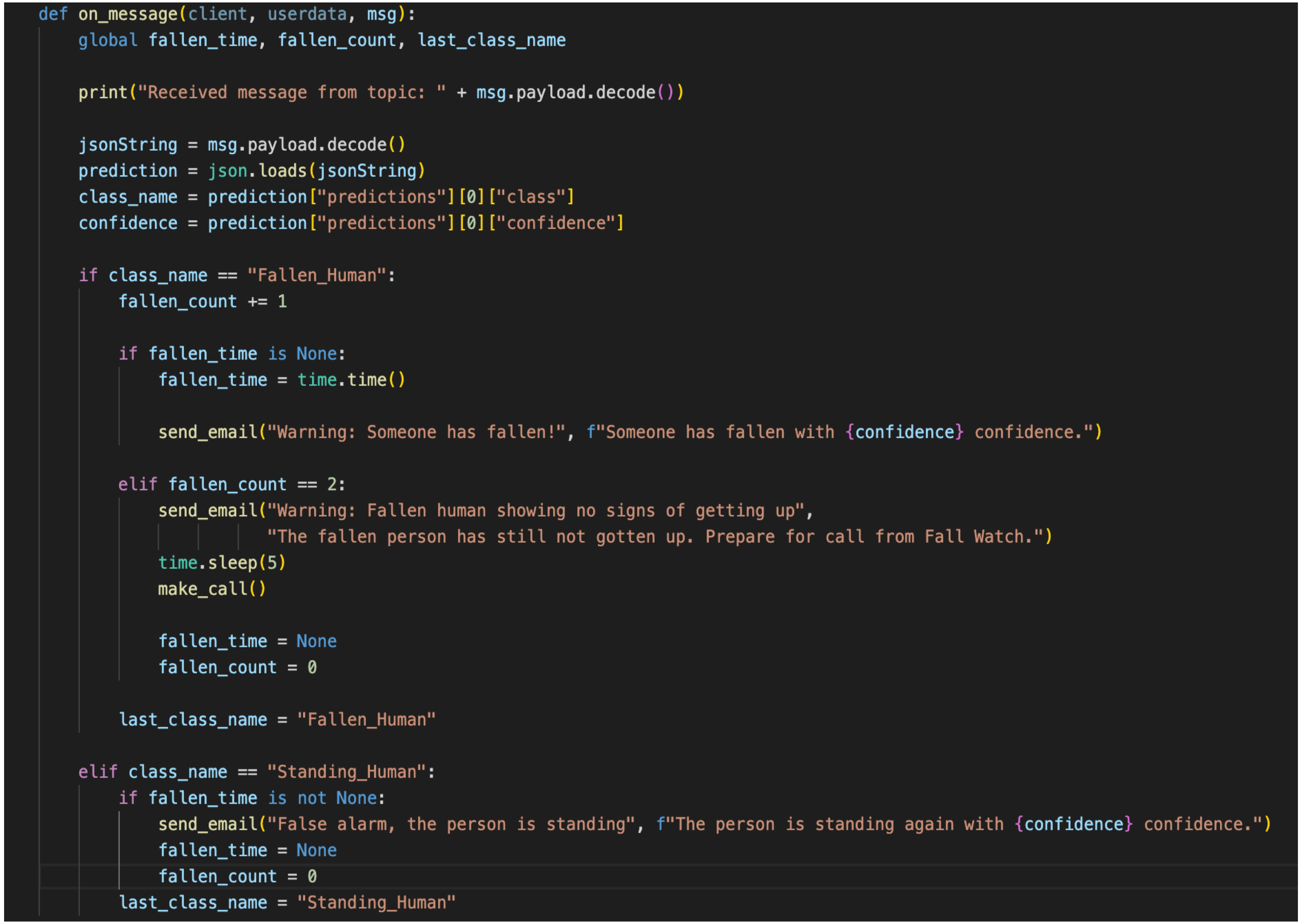


```python
import paho.mqtt.client as mqtt
import json
from email.message import EmailMessage
import smtplib
import ssl
import time
from twilio.rest import Client

# MQTT broker settings
mqtt_broker = "broker.hivemq.com"
mqtt_port = 1883
mqtt_topic = "dlaw4608/home/prediction/person"

# Email settings
email_sender = 'standfallwatch123@gmail.com'
email_password = "sguxfhwqgkyyjcwn"
email_receiver = "gebofe7016@andorem.com"

# Twilio settings
account_sid = 'AC0730b8042e56d8b50ce763a3e3b68de7'
auth_token = '9f48c555070acdfe68f8ca05232d00cb'
twilio_phone_number = '+16205318597'
my_phone_number = "+353857058967"

# Global variables
fallen_time = None
fallen_count = 0
last_class_name = None

context = ssl.create_default_context()


def on_connect(client, userdata, flags, rc):
    print("Connected to MQTT broker" + str(rc))
    client.subscribe(mqtt_topic)
```

Notification code continued…….

The Logic

- The code detects if a person has fallen.
- Sends an email warning if a person has fallen
- Starts a fallen counts
- Sends another email and makes a phone call using Twilio if the fallen count reaches 2.
- Sends a “False Alarm” email if the second prediction is a standing human
- Resets the fallen count.

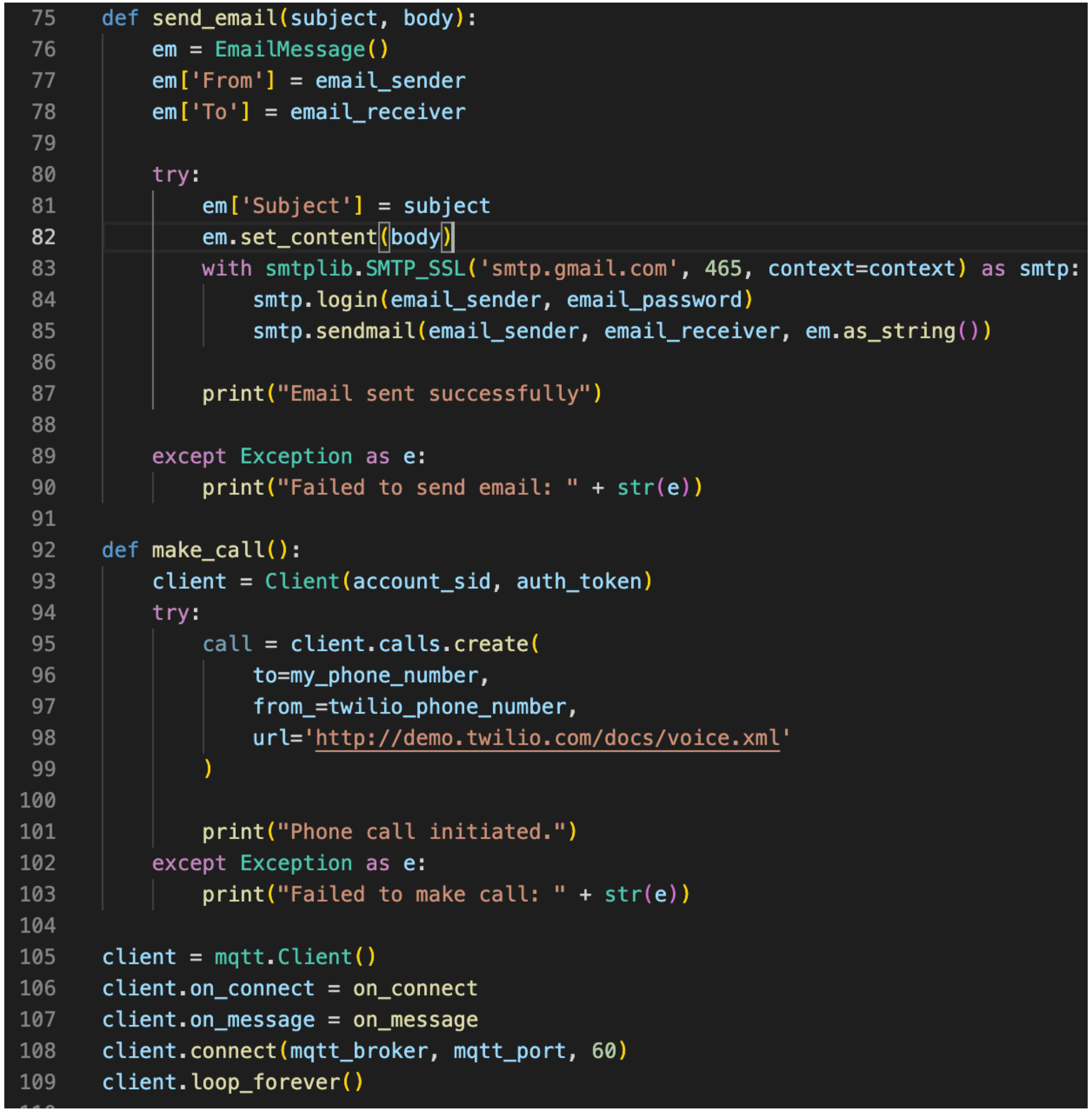


```python
def on_message(client, userdata, msg):
    global fallen_time, fallen_count, last_class_name

    print("Received message from topic: " + msg.payload.decode())

    jsonString = msg.payload.decode()
    prediction = json.loads(jsonString)
    class_name = prediction["predictions"][0]["class"]
    confidence = prediction["predictions"][0]["confidence"]

    if class_name == "Fallen_Human":
        fallen_count += 1

        if fallen_time is None:
            fallen_time = time.time()

            send_email("Warning: Someone has fallen!", f"Someone has fallen with {confidence} confidence.")

        elif fallen_count == 2:
            send_email("Warning: Fallen human showing no signs of getting up",
                       "The fallen person has still not gotten up. Prepare for call from Fall Watch.")
            time.sleep(5)
            make_call()

            fallen_time = None
            fallen_count = 0

        last_class_name = "Fallen_Human"

    elif class_name == "Standing_Human":
        if fallen_time is not None:
            send_email("False alarm, the person is standing", f"The person is standing again with {confidence} confidence.")
            fallen_time = None
            fallen_count = 0
        last_class_name = "Standing_Human"
```

Final part of notification script

- The “send_email” function sends the email. Creating an object and sets the sender, receiver and the specific subject and body., logging into the email server using an SMTP library.

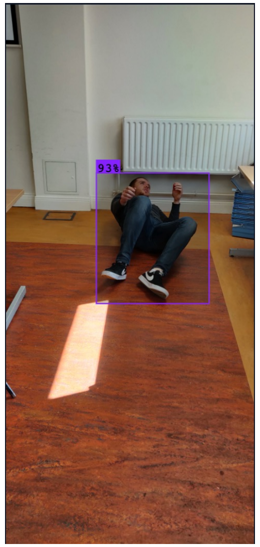
- The “make_call” implements Twilio to make the phone call. Creating a client using the Twilio account SID and the authentication token.



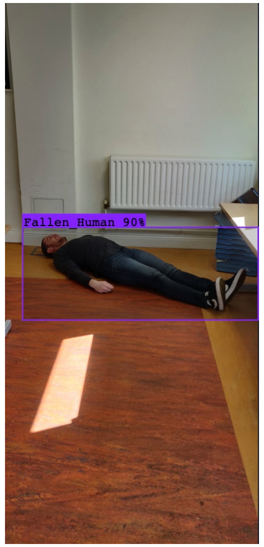
```python
def send_email(subject, body):
    em = EmailMessage()
    em['From'] = email_sender
    em['To'] = email_receiver

    try:
        em['Subject'] = subject
        em.set_content(body)
        with smtplib.SMTP_SSL('smtp.gmail.com', 465, context=context) as smtp:
            smtp.login(email_sender, email_password)
            smtp.sendmail(email_sender, email_receiver, em.as_string())

        print("Email sent successfully")

    except Exception as e:
        print("Failed to send email: " + str(e))

def make_call():
    client = Client(account_sid, auth_token)
    try:
        call = client.calls.create(
            to=my_phone_number,
            from_=twilio_phone_number,
            url='http://demo.twilio.com/docs/voice.xml'
        )

        print("Phone call initiated.")
    except Exception as e:
        print("Failed to make call: " + str(e))

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
client.connect(mqtt_broker, mqtt_port, 60)
client.loop_forever()
```

Visualizing the scenarios

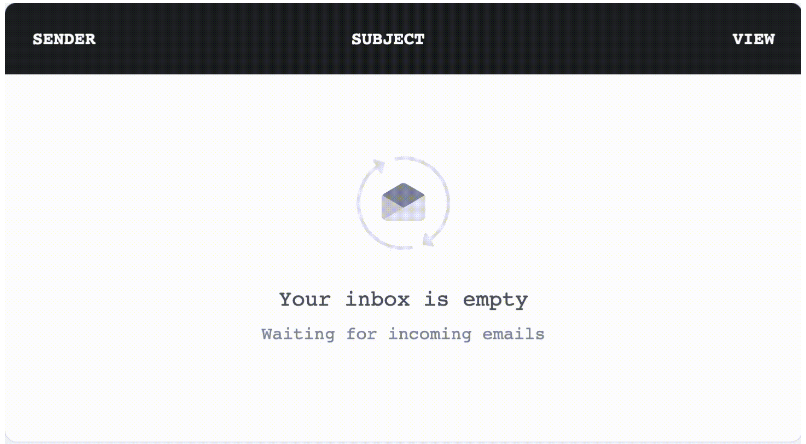
Real Positive



+



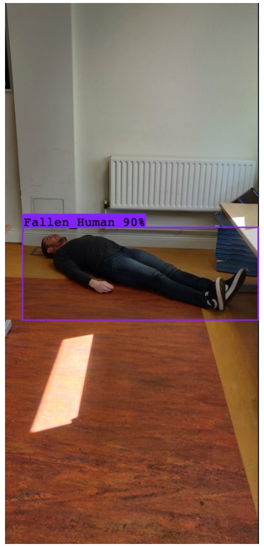
=



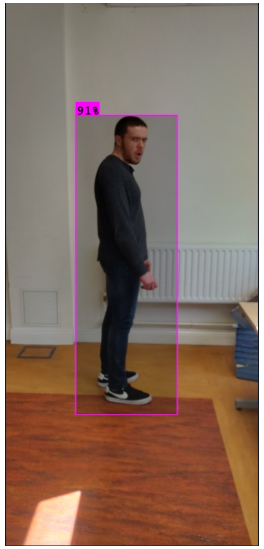
Your inbox is empty

Waiting for incoming emails

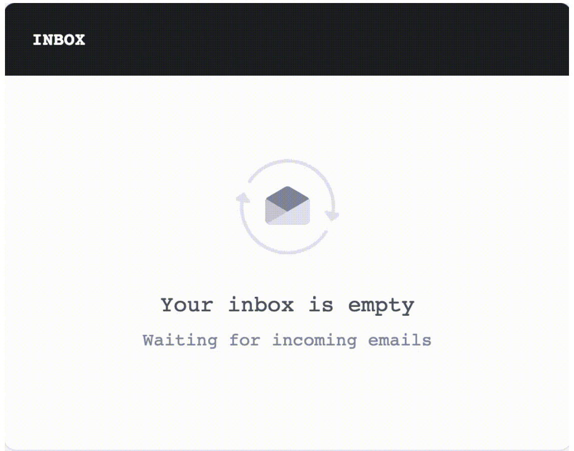
False Positive



+



=



Your inbox is empty

Waiting for incoming emails

Functionality Synopsis

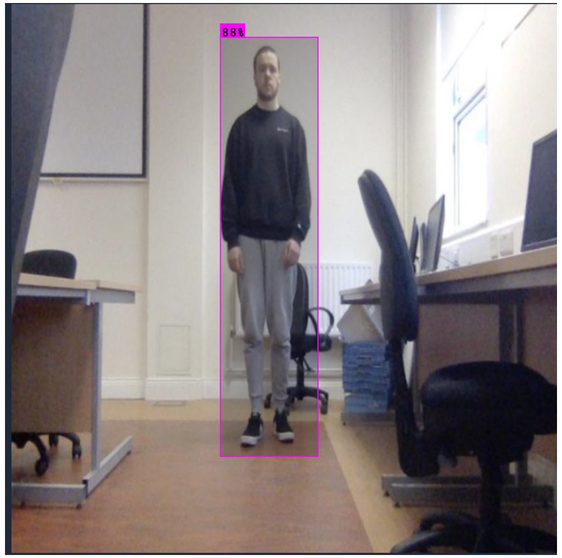
- Implements an Object Detection algorithm.
- Trained said algorithm to detect whether someone is standing or fallen.
- Successfully Use Roboflow API to make predictions.
- Publish Predictions to an MQTT Broker
- Have a working “scenario” python script which notifies by email and by phone call.

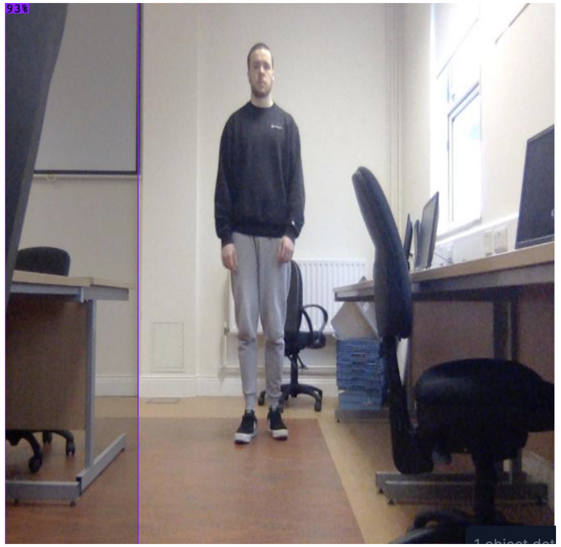
Challenges

- The biggest challenge that eventually had to be removed, was trying to get the RoboFlow API to work using a webcam and the console to pick up predictions in real time, predictions that can be published to a broker using WebSocket’s.

- Training the Models to be able to differentiate between standing and falling.

- General Troubleshooting, script rewrites to achieve functionality.





The next step

- The next step would be to move on from the proof-of-concept simulation, to integrate live footage to make predictions.
- Better training of models, increasing the testing of images form just one person to multiple people in multiple environments, etc.
- Another use case that I believe would be interesting is the use of speech to text API from google cloud. This would act as another warning as if someone was to fall and had their phone they could ask for help.

Google Cloud
Speech API

Final Thoughts

Having achieved a good foundation to build on.

Developing the core features of the use case I wanted to complete.

I believe overall, the project was a successful proof of concept.

Resources used

- IoT Standard and protocols Notes
- Steve’s Internet Guide
- RoboFlow Blog(Ex. How to Train a YOLOv5 Model on a custom dataset)

- GitHub Repo: https://github.com/dlaw4608/Fall-Detect-IOT