Web-camera surveillance system

Table of Contents

1.Introduction

1.1 Context

The objective of this project is to develop, implement, and evaluate a real-time object recognition system using deep learning techniques. Unlike traditional image recognition systems, this project will focus on recognizing objects within images or video streams and providing users with information about those objects.

The system can find applications in various fields such as security and surveillance, autonomous vehicles, robotics, and augmented reality. It can be integrated into devices that require object recognition capabilities or used as a standalone solution in scenarios where real-time identification of objects is essential.

1.2 Specifications

The project will involve both hardware and software components. Hardware may include a webcam or camera for image capture, while software will include deep learning models for object recognition. The system is designed for real-time object recognition, ensuring that objects are identified as they appear in the camera feed. The system should be compatible with various types of cameras or webcams and flexible to accommodate different input sources. The design will allow for easy integration into other systems or devices that require object recognition.

1.3 Objectives

This project represents a compelling exploration of computer vision and deep learning for real-time object recognition. Its potential applications are diverse, making it a valuable tool in numerous fields. Success in this project depends on the quality of the deep learning model, real-time processing capabilities, and the effectiveness of the user interface in providing timely information about recognized objects.

2. Bibliographic study

An object detection model is trained to detect the presence and location of multiple classes of objects. For example, a model might be trained with images that contain various pieces of fruit, along with a label that specifies the class of fruit they represent (e.g., an apple, a banana, or a strawberry), and data specifying where each object appears in the image.

When an image is subsequently provided to the model, it will output a list of the objects it detects, the location of a bounding box that contains each object, and a score that indicates the confidence that detection was correct.

The model is designed to accept input in the form of an image with dimensions of 300x300 pixels, featuring three color channels: red, blue, and green. To utilize this model, it is essential to prepare a flattened buffer composed of 270,000 bytes. In this buffer, each byte signifies a quantized value, ranging from 0 to 255, representing the intensity of the color channels for each pixel. This input format is crucial for ensuring compatibility with the model's requirements.

3.Analysis

3.1 Image Recognition

The process of image recognition is a multifaceted journey that merges data, algorithms, and continual refinement. Within this realm, it is paramount to explore the intricate dynamics that govern the efficacy of recognition systems.

The representation of a digital image as a matrix of numerical values is a fundamental step in the image recognition pipeline. These numerical values encapsulate pixel intensities, capturing the essence of the visual information. It is crucial to note that the data includes not only the intensities but also the spatial arrangement of pixels within the matrix. This spatial information is essential for understanding the context and relationships between image components.

Recognition systems, particularly those built on neural networks, harness this raw pixel data to extract relevant features and patterns. The algorithms employed, such as convolutional layers in Convolutional Neural Networks (CNNs), are adept at recognizing edges, textures, and shapes. This feature extraction is a pivotal precursor to accurate recognition and classification.

The training phase in image recognition is where the recognition system learns to map pixel intensities and features to predefined labels or categories. This phase is characterized by the exposure to a vast dataset of labeled images, facilitating the establishment of associations and patterns. Following this training phase, the performance of the system is meticulously assessed using separate test data. This validation step ensures the system's ability to generalize its acquired knowledge to novel, unseen images.

The pursuit of high accuracy in image recognition is an iterative process. It entails fine-tuning the recognition system by modifying the intermittent weights and parameters of the neural networks. This fine-tuning might involve architectural adjustments or hyperparameter optimization. These refinements are integral to enhancing the system's recognition accuracy, adaptability, and robustness.

The algorithms deployed in image recognition, such as SIFT, SURF, PCA, and LDA, play pivotal roles in addressing specific challenges. For instance, SIFT and SURF excel in handling scale and rotation invariance, making them suitable for object recognition. PCA and LDA, on the other hand, are instrumental in dimensionality reduction and discrimination, thus aiding tasks like face recognition.

In the context of this analysis chapter, it is paramount to delve deeper into the performance of these algorithms, discussing their respective strengths and limitations in varying scenarios and datasets.

Ultimately, image recognition stands at the intersection of complex data processing and cutting-edge technology. The efficacy of recognition systems relies on a profound understanding of data, the judicious selection of algorithms, and a relentless pursuit of improvement to tackle the challenges and possibilities that lie ahead. This analysis will explore these aspects in greater detail, shedding light on the intricacies of image recognition and its broader implications.

3.2 Tensor Flow Object Detection API

The model produces four distinct arrays, meticulously associated with indices ranging from 0 to 4. Notably, arrays 0, 1, and 2 serve as comprehensive descriptors for the identification and characterization of N detected objects within the analyzed data. Each of these arrays is structured such that every individual element corresponds to a specific object, meticulously encapsulating essential attributes and characteristics relevant to the subject of investigation. This methodological approach ensures the thorough and exhaustive representation of the detected objects, contributing to a robust and in-depth analysis of the research findings.

Index	Name	Description
0	Locations	Multidimensional array of [N][4] floating point values between 0 and 1, the inner arrays representing bounding boxes in the form [top, left, bottom, right]
1	Classes	Array of N integers (output as floating-point values) each indicating the index of a class label from the labels file
2	Scores	Array of N floating point values between 0 and 1 representing probability that a class was detected
3	Number of detections	Integer value of N

For example, let's imagine a model that has been trained to detect apples, bananas, and strawberries. When provided an image, it will output a set number of detection results - in this example, 5.

Class	Score	Location
Apple	0.92	[18,21,57,63]
Banana	0.88	[100,30,180,150]
Strawberry	0.87	[7,82,89,163]
Banana	0.23	[42,66,57,83]
Apple	0.11	[6,42,31,58]

To interpret these results, we can look at the score and the location for each detected object. The score is a number between 0 and 1 that indicates confidence that the object was genuinely detected. The closer the number is to 1, the more confident the model is.

Depending on the application, we can decide a cut-off threshold below which we will discard detection results. For the current example, a sensible cut-off is a score of 0.5 (meaning a 50% probability that the detection is valid). In that case, the last two objects in the array would be ignored because those confidence scores are below 0.5:

Class	Score	Location
Apple	0.92	[18,21,57,63]
Banana	0.88	[100,30,180,150]
Strawberry	0.87	[7,82,89,163]

Banana	0.23	[42,66,57,83]
Apple	0.11	[6,42,31,58]

4.Design

The webpage is designed to serve as a user interface to enable and view webcam feeds once TensorFlow and COCO-SSD models have finished loading. The body of the webpage begins with a heading stating the project title. A paragraph beneath the heading informs the visitor that the "TensorFlow" model loading process is ongoing and once it is finished, the button "Enable webcam" will become available(Fig 4.1, Fig 4.2). The webpage contains a section named "cameraSection". This section is initially set as "invisible" with reduced opacity and will become more visible once the model is loaded.

The webpage's styling is defined inside a <style> element. Video elements are displayed as a block. The "invisible" class is used to control the visibility of the camera feed, while "highlighter" is used to spotlight the objects the model has detected. For the script part, the webpage relies on external JavaScript libraries such as "TensorFlow" and "COCO-SSD" (the model used to detect objects). These libraries are essential for the webcam surveillance system to work as expected. The "script.js" file is included with the defer attribute, indicating that the code inside this file should be implemented only after the HTML files have been parsed. Inside the file, there is the logic for enabling the webcam and interpretating the result received from the COCO-SSD model.

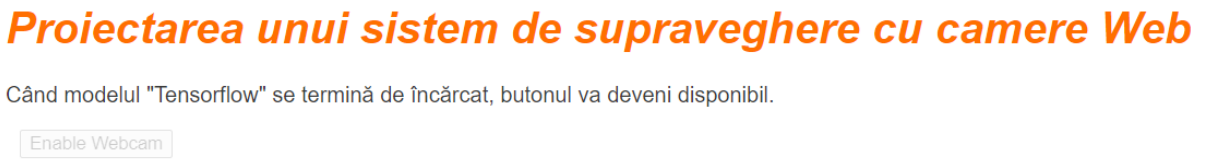


Fig 4.1

Fig 4.2

5.Implementation

5.1 Index.html

The head section contains metadata and links to external resources. Inside the head element, there is a title defined, a meta tag to specify which character encoding the website is using, and an external CSS library import which is defined in the same project.

```
<head>
  <title>Home </title>
  <meta charset="utf-8">
  <link rel="stylesheet" href="style.css">
</head>
```

Inside the body section, the visible content of the webpage is defined. First of all, there is a <h1> tag which prominently displays the title of the project. Next, there is a <p> tag that provides an important piece of information to the user. Next, there is a section named "cameraSection" that is "activated" only after the button "Enable Webcam" and the COCO-SSD model finishes loading. In the end, the relevant JavaScript libraries are imported using the <script> tag, adding a "defer" parameter to make sure the script runs only after the HTML has finished parsing.

```
<body>
  <h1>Proiectarea unui sistem de supraveghere cu camere Web </h1>

  <p>Când modelul "Tensorflow" se termină de încărcat, butonul va deveni disponibil.</p>

  <section id="cameraSection" class="invisible">

    <div id="liveView" class="camView">
      <button id="webcamButton">Enable Webcam</button>
      <video id="webcam" autoplay muted width="640" height="480"></video>
    </div>
  </section>

  <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs/dist/tf.min.js" type="text/javascript"></script>
  <script src="https://cdn.jsdelivr.net/npm/@tensorflow-models/coco-ssd"></script>

  <script src="script.js" defer></script>
</body>
```

5.2 Script.js

The script present in this file is a JavaScript code that uses the browser's WebRTC API to access the user's webcam and performs object detection on the video stream using the COCO-SSD model. The detected objects are then displayed on the web page along with a confidence score, and if a "person" is detected, a sound alert is played.

The most important function here is predictWebcam (Fig. 5.2) which has the following roles:

1. **Object Detection:**

The function uses the model.detect method to perform object detection on the video stream (video element). The detected objects and their information are stored in the predictions array.

2. **Clear Previous Predictions:**

It removes previously displayed elements (highlighters and paragraphs) from the liveView element to prepare for the new predictions.

3. **Display Predictions:**

It iterates through the predictions array and checks if the confidence score for each prediction is greater than 0.66. For each qualified prediction, it creates a paragraph (p) element displaying the class name and confidence percentage. The style of the paragraph is set based on the bounding box information. It also creates a highlighter div element to visually highlight the detected object. The style of the highlighter is set based on the bounding box information.

4. **Sound Alert for "Person":**

If the detected object is a "person," the playSound function is called to play a sound alert.

5. **Append Elements to Live View:**

The paragraph and highlighter elements are appended to the liveView element.

6. **Update 'children' Array:**

The paragraph and highlighter elements are added to the children array for tracking.

7. Continuation with Animation Frame:

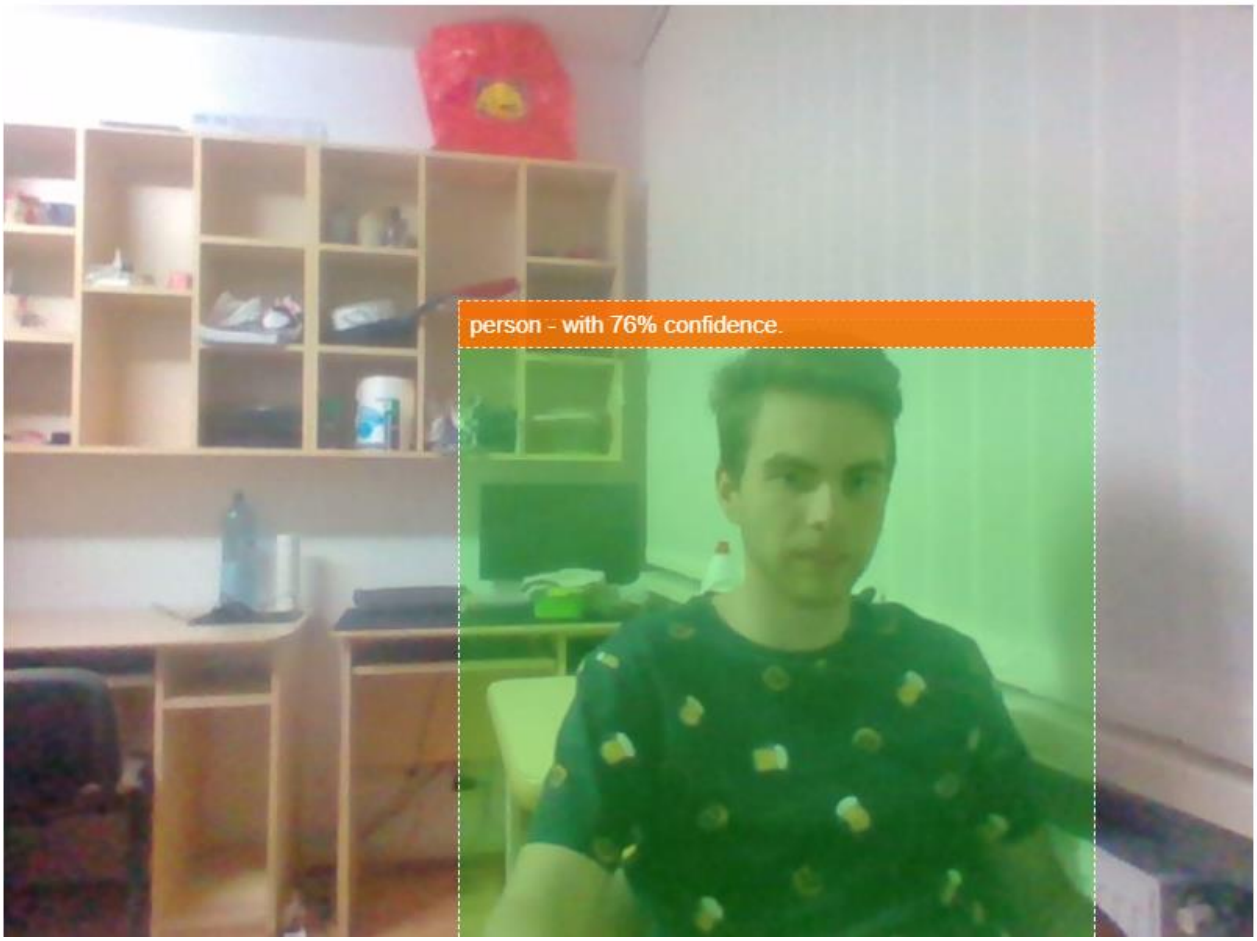
The function uses window.requestAnimationFrame to schedule the next animation frame, effectively creating a loop for continuous object detection and display.

This function is essentially responsible for updating the live view with the latest object detection results from the webcam feed, clearing the previous results, and providing a visual and auditory alert for certain conditions (e.g., detecting a person).

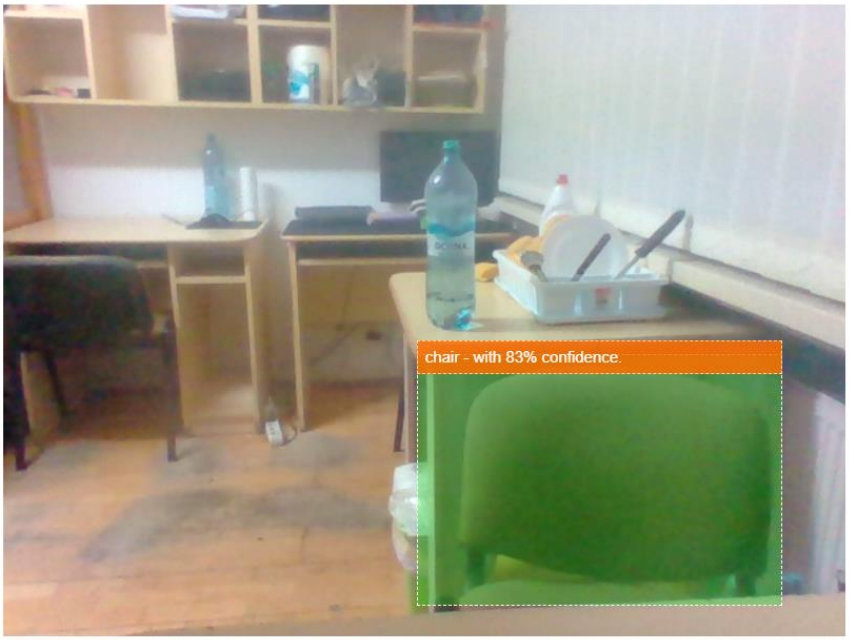
```
function predictWebcam() {
  model.detect(video).then(function (predictions) {
    for (let i = 0; i < children.length; i++) {
      liveView.removeChild(children[i]);
    }
    children.splice(0);


    for (let n = 0; n < predictions.length; n++) {
      if (predictions[n].score > 0.66) {
        const p = document.createElement('p');
        p.innerText = predictions[n].class  + ' - with '
            + Math.round(parseFloat(predictions[n].score) * 100)
            + '% confidence.';
        p.style = 'margin-left: ' + predictions[n].bbox[0] + 'px; margin-top: '
            + (predictions[n].bbox[1] - 10) + 'px; width: '
            + (predictions[n].bbox[2] - 10) + 'px; top: 0; left: 0;';

        const highlighter = document.createElement('div');
        highlighter.setAttribute('class', 'highlighter');
        highlighter.style = 'left: ' + predictions[n].bbox[0] + 'px; top: '
            + predictions[n].bbox[1] + 'px; width: '
            + predictions[n].bbox[2] + 'px; height: '
            + predictions[n].bbox[3] + 'px;';

        if(predictions[n].class == "person"){
          playSound();
        }       You, 4 weeks ago • added sound when detecting a person
        liveView.appendChild(highlighter);
        liveView.appendChild(p);
        children.push(highlighter);
        children.push(p);
      }
    }
```

Fig 5.2

6.Testing and validation



The model detects a person and plays a sound when the threshold is above 66%.



However, it detects other objects as well but it does not play a sound unless it is a person.

7.Conclusions

The objective of this project was to develop a web-camera surveillance system that provides users with real-time access to the camera feed. An additional feature involves triggering an auditory alert when a person is detected by the camera.

Several potential improvements have been identified for the current implementation:

1. **Utilization of an Improved Object Detection Model:**

Implementing a more sophisticated object detection model could enhance the system's accuracy and efficiency in identifying objects, particularly individuals, within the camera feed.

2. **Integration of a Higher-Resolution Camera:**

Upgrading the camera to a model with superior resolution has the potential to improve the system's ability to discern objects within the room more distinctly, contributing to enhanced object detection performance.

3. **Implementation of an On/Off System for Detection:**

Introducing a user-controlled system to initiate or cease object detection provides users with autonomy, enabling them to selectively activate the surveillance system.

Further considerations for refinement include:

- **Alert Customization:**

Facilitating user customization of the alert system, permitting the selection of various warning modalities or the integration of notifications through diverse channels, such as email or mobile applications.

8.Bibliography

[1] "Speed/accuracy trade-offs for modern convolutional object detectors."

Huang J, Rathod V, Sun C, Zhu M, Korattikara A, Fathi A, Fischer I, Wojna Z,

Song Y, Guadarrama S, Murphy K, CVPR 2017

[2] https://github.com/tensorflow/tfjs-models/tree/master/coco-ssd

[3] https://www.mathworks.com/discovery/object-detection.html

[4] https://www.npmjs.com/package/@tensorflow-models/coco-ssd