EdgeAI

Mutlu, Berat Emir - Kocyigit, Kazim Efe

Contents

1 Introduction

1.1 Project Information

Group 2 is responsible for two main components of the project:

- Developing specialized tasks (color and direction detection) on edge devices as part of the federated learning framework.
- Designing and implementing a web-based graphical user interface (GUI) to visualize the outputs of these tasks in an interactive and user-friendly manner.

1.2 Goals and Content

This project aims to explore and implement real-time visual perception tasks under a federated learning environment. The specialized tasks address detection and analysis of object attributes (e.g., color and direction), while the web interface provides visual insight into model behavior.

1.3 Project Organization

The responsibilities were distributed across subgroups, with Group 2 focusing on the perception and visualization components. Regular meetings ensured synchronization across development milestones.

1.4 Structure of the Report

This report is divided into the following sections:

- Team tasks and implementation details
- Specialized edge-device functionalities
- Visualization tool overview
- Summary and future work

2 Team Tasks and Responsibilities

2.1 Project Planning

Initial planning involved defining concrete tasks, allocating team members, and setting up shared repositories for collaboration.

2.2 Design

We followed a modular architecture: Edge devices handle real-time detection tasks and periodically send results to a central server or dashboard for analysis.

2.3 Implementation

Our main tasks were:

- Color detection using image processing and clustering.
- Direction detection using geometric vector estimation.
- Web-based interactive visualization of ML outputs.

2.4 Results

Both specialized tasks and the frontend tool were implemented successfully. Live data visualization enables efficient model validation and debugging.

2.5 Summary

The group has delivered functioning components that are integrable into the larger federated learning system.

2.6 Next Steps

Possible extensions include integrating more types of visual attributes and supporting more advanced user interaction modes in the frontend.

2.7 References

References used throughout development are listed in Section 5.

3 Specialized Tasks

3.1 Color Detection

Objective: Determine the dominant visible color of each tracked vehicle.

Methodology:

1. Detection of vehicles using YOLOv5.
2. Extraction of bounding box regions via `cropVehicle()`.
3. Filtering out transparent pixels using RGBA masking.
4. RGB pixel clustering using K-Means.
5. Mapping the dominant RGB value to an XKCD color label (e.g., *light blue*, *dark red*).
6. Updating the detected color label every 1 second.

Example Python Implementation:

Listing 1: Color Detection via KMeans Clustering

```
def cropCar(image):
 .
 .
 .
largest = max(contours, key=cv2.contourArea)

    mask = np.zeros(image.shape[:2], dtype=np.uint8)
    cv2.drawContours(mask, [largest], -1, color=(255,), thickness=-1)

    masked = cv2.bitwise_and(image, image, mask=mask)
    x, y, w, h = cv2.boundingRect(largest)
    car_cropped = masked[y:y + h, x:x + w]
    mask_cropped = mask[y:y + h, x:x + w]

    if car_cropped.shape[0] == 0 or car_cropped.shape[1] == 0:
        print("Cropped␣image␣is␣empty.")
        return None, None, None
```

```
    b, g, r = cv2.split(car_cropped)
    alpha = mask_cropped
    rgba = cv2.merge((r, g, b, alpha))
```

3.2 Direction Detection

Objective: Estimate the movement direction of each vehicle.

Methodology:

1. Compute center point of each vehicle per frame.
2. Estimate motion angle based on the last two frames.
3. Map angle to one of the eight cardinal directions (N, NE, E, etc.).
4. Update direction label every 2 seconds.

Example Python Implementation:

Listing 2: Direction Estimation Using Angle Between Frames

```
def get_center(x1, y1, x2, y2):
    return int((x1 + x2) / 2), int((y1 + y2) / 2)

def compute_angle(start, end):
    dx = end[0] - start[0]
    dy = end[1] - start[1]
    return math.degrees(math.atan2(-dy, dx)) % 360
```

4 Web-Based Visualization Tool

4.1 Purpose and Functionality

The tool visualizes ML predictions from video data using a JSON file containing:

- Frame metadata
- Object class predictions
- System usage stats (CPU, memory)

Core Features:

- Track classification over time
- Observe model behavior frame-by-frame
- Compare object category frequencies
- View optional hardware usage plots

4.2 User Interaction

- Dropdowns to filter classes, devices, timeframes
- Hover to inspect exact values
- Live update of charts with new data
- Upload new JSON outputs for reanalysis

4.3 Chart Design

Technology: `Chart.js`

Chart Types:

- Line Chart: prediction confidence over time
- Bar Chart: object class frequency
- Pie Chart: object class distribution
- Optional time-series plots: CPU/memory usage

Charts are styled within HTML card containers for a clean and organized layout.

4.4 Code Structure

- `index.html`: layout, canvas, and external links
- `main.js`: data parsing, chart setup, interactivity
- `style.css`: layout styling and UX behavior

5 Conclusion

We have implemented and tested real-time visual perception modules for edge devices and provided an interactive frontend for monitoring their output. These tools contribute to interpretable federated learning.

References

- YOLOv8: `https://github.com/ultralytics/yolov8`
- Chart.js: `https://www.chartjs.org/`
- XKCD Color List: `https://xkcd.com/color/rgb/`