Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Курсова робота

освітній ступінь – бакалавр

на тему: «**Real-time motion detection and alert management system using OpenCV and Deep Learning**»

Виконав: студент 3-го року навчання,

освітньо-наукової програми «Інженерія програмного забезпечення», 121

Цимбал Ілля Володимирович

Керівник ст. викладач Кузьменко Д.О.

Старший викладач кафедри мультимедійних систем

Курсова робота захищена

з оцінкою _____

«____» _____ 20____ р.

Київ – 2024

Table of Contents

1 INTRODUCTION

In the rapidly evolving domain of computer vision, the detection and analysis of human motion in real-time stands as a critical challenge with profound usages across many different applications, ranging from security surveillance to interactive systems and healthcare monitoring. As digital environments become increasingly integrated into daily life, the ability to accurately and efficiently recognize human activities in various contexts not only enhances the functionality of interactive systems but also ensures safety and provides critical data for decision-making processes.

The purpose of this research is to design and implement a real-time person detection in video streams and alert management system using OpenCV [4] and Deep Learning detectors. To achieve this overarching goal, the following tasks are identified:

1. Analyze existing detectors and evaluate their performance in real-time motion detection scenarios.
2. Develop a Python application using OpenCV for real-time video processing and integration with Deep Learning detectors.
3. Implement algorithms for recording and alerting based on detected motion.
4. Evaluate the system's performance in terms of accuracy, speed, and reliability under various conditions.
5. Compare the results with existing solutions and identify areas for improvement and future research directions.

The object of this study is a motion detection security system designed to identify and alert on the instances of human movement in real-time through the integration of OpenCV and deep learning-based single-shot detectors.

The results of this research hold practical significance in various domains, including but not limited to security surveillance, traffic monitoring, and human-computer interaction. The developed system can be utilized for proactive threat

detection, automated surveillance, and enhancing overall situational awareness in real-time scenarios. These findings offer valuable insights for further improving motion detection systems and their practical applications.

This research is structured into extensive sections, each delving into a crucial element of implementing a real-time person detection application.

Section 2 of this work focuses on a comprehensive examination of existing object detection methodologies and choosing the most suitable approach for our specific needs.

Section 3 is dedicated to the implementation of the application, detailing its architecture, the technologies used, and the integration processes.

Section 4 describes the experiments conducted to evaluate the performance of our implemented security system under various conditions. This section outlines the testing scenarios used to assess the system's efficiency overall.

2 OBJECT DETECION METHODS

Understanding different object detection methods, their strengths and limitations is crucial when building such an application. This section aims to analyse and identify the most suitable approach that aligns with the specific needs and requirements of our task.

2.1 Detectors

In the realm of deep learning based object detection, three primary methods stand out:

1. Faster R-CNNs [1]
2. You Only Look Once (YOLO) [2]
3. Single Shot Detectors (SSDs) [3]

Faster R-CNNs are often recognized as the most widely known method for object detection utilizing deep learning. However, comprehending this technique, especially for novices in deep learning, can pose challenges in terms of understanding, implementation, and training complexities. Additionally, despite the use of the "faster" R-CNN implementation (with "R" representing "Region Proposal"), the algorithm can still be relatively slow, operating at around 7 frames per second.

For scenarios prioritizing speed, YOLO emerges as a preferred choice due to its significantly faster processing capabilities. In its fastest variant, YOLO can even reach an impressive 155 FPS. However, YOLO's drawback lies in its compromised accuracy.

Single shot detectors, initially pioneered by Google, strike a balance between the aforementioned methods. This algorithm is comparatively more straightforward and, better described in its original paper than Faster R-CNNs. It

offers a middle ground, combining acceptable speed with a reasonable level of accuracy.

2.2 MobileNets

When building object detection networks, a common practice is to use existing network architectures like VGG or ResNet within the object detection pipeline. However, a significant issue with such an approach lies in a considerable size of these architectures, often ranging from 200 to 500 megabytes.

These network architectures are unsuitable for resource constrained devices because of their large size and resulting number of computations. And this factor is crucial within the realm of security systems, specifically due to the compact size of the devices utilized in this domain.

Instead MobileNets can be used to solve this problem, as they were created for mobile and embedded vision applications. Such networks are built upon an efficient architecture employing depth-wise separable convolutions to construct lightweight deep neural networks.

The general idea behind depthwise separable convolution is to split convolution into two stages: a 3x3 depthwise convolution then followed by a 1x1 pointwise convolution.

By combining the MobileNet architecture and the Single Shot Detector (SSD) framework, we recieve a fast and efficient deep learning method for object detection, suitable for usage on resource constrained devices, which is an ideal set of characteristics for our purposes.

The model of choice is a Caffe implementation of Google's MobileNets. Which was first trained on the COCO dataset [5] (Common Objects in Context) and was then tuned on PASCAL VOC [6] reaching 72.7 mAP.

3 IMPLEMENTING SECURITY SYSTEM

3.1 Tech stack and high level architecture

For computer vision tasks a Python & OpenCV stack was chosen. Python's distribution of OpenCV offers a streamlined and pythonic approach for utilizing OpenCV's features, making it widely used by developers when working on computer vision projects. This choice goes with some important benefits:

1. Python's simplicity and human-like syntax make it easy to access OpenCV functionalities, allowing one to write complex computer vision logic for a relatively short period of time.
2. It has an extensive documentation that is well-maintained and provides detailed explanations of functions, modules, and techniques.
3. Python & OpenCV is one of the most popular solutions for computer vision tasks, which means it has large and active communities of developers. This indicates that there are a lot of resources, tutorials, forums, and libraries available to help you solve problems.

To manage our computer vision logic through an API solution, we employ the FastAPI library [7] due to its capability to create simple and lightweight APIs very quickly, aligning perfectly with our requirements.

For the implementation of the frontend of our application, Streamlit [8] was chosen as a solution. It allows to create presentable user interfaces with minimum time and efforts, thanks to predefined Streamlit components.

In the context of using the Streamlit Python library for user interface development, a reasonable question may arise: why to create an additional API to be interfaced with Streamlit, when the necessary logic can simply be executed directly within the Streamlit code, given that everything is scripted in Python?

The idea lying behind this approach is to decouple the core detection logic from the specific user interface implementation, thereby facilitating seamless UI modifications if required.

By segregating the detection logic into an independent API, we gain the flexibility to alter or update the UI implementation without changing the core functionality, ensuring a more modular and adaptable software architecture.

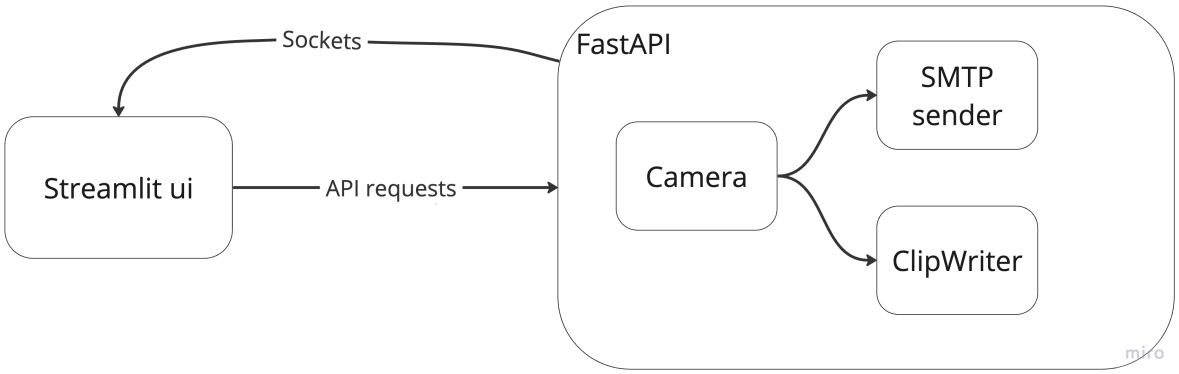

Figure 1: High level architecture diagram

3.2 Core logic implementation

This subsection details the implementation of the primary application logic, covering different aspects of the development process, encountered challenges, and their respective solutions.

3.2.1 Camera input handling

At the core of our system is the `Camera` class, which interacts directly with a video streaming source. This class is responsible for sequentially retrieving video frames and processing them in real-time. The continuous stream of frames forms the primary input for our motion detection algorithm.

3.2.2 Person detection logic

We use a selected deep learning model to perform object detection on each frame. When processing a frame, the model outputs a list of detected objects, each with an associated confidence score. Our system specifically looks for detections categorized under the person class, that meet a predefined confidence threshold.

When a person is detected with sufficient confidence, two key operations are initiated: the start of the clip writer (if not yet started) and the sending of a notification signal about the detection.

3.2.3 Detection clip recording

For managing the recording of clips with a detected person, a separate specific class is implemented (DetectedClipWriter). This class features a configurable buffer that temporally stores a sequence of frames. The size of this buffer is set to accommodate several seconds of video both prior to and following a detection event, thus preserving the context around the motion which is often vital for comprehensive analysis in security applications.

The clip writer works in a separate thread from the main video processing loop. This architectural decision allows the system to maintain high performance by offloading the intensive input/output operations associated with writing video data to disk away from the main thread that processes incoming video frames. This separation ensures that the frame capture rate remains consistent and unaffected by disk write speeds.

When activation of recording is triggered by a person detection, the clip writer first dumps all pre-buffered frames to a video file. It then continues to record the stream while a person is present in the frame and for the same buffer size duration after the detection event, capturing the aftermath, which may provide additional valuable insights.

3.3 Frontend implementation

In order to provide a user-friendly interface for our computer vision application, we used the Python library Streamlit to create simple and intuitive user experience. The user interface was designed to provide several key features essential for effective interaction with the application. These features included real-time video streaming, toggling the person motion detection on and off, accessing a list of recorded clips and managing notifications such as enabling or disabling them and modifying the email address for notification delivery.

From a UI/UX perspective, the design approach aimed to be intuitive for the users. To achieve this, the idea was to structure the user interface into distinct pages, with the help of a relatively new feature of Streamlit known as Multipage App.

This feature allows to decompose the Streamlit code into separate files organized under the 'pages' folder, each representing a separate page. Which differs from an initial Streamlit approach, where even the complex multipage apps were implemented in a single file, making it hard to maintain and extend such code base.

The new Multipage App code design solution not only allows to create multiple user interface pages themselves, but also gives an opportunity to decompose logic into multiple files which enhances code readability, maintainability, and scalability, avoiding the pitfalls of a monolithic codebase that could become unwieldy over time.

The main UI controls, including functionalities such as enabling/disabling motion detection and managing notifications, were implemented using toggle switches. These switches served as intuitive user input elements, with their corresponding handlers executing the necessary requests to our API when triggered. This approach not only improved the user experience by providing clear and concise controls but also streamlined the backend functionality by ensuring that each UI element was associated with specific actions and API calls, enhancing the overall system's responsiveness and reliability.

On the main page of our user interface, the video stream functionality was implemented through the insertion of a single-element container. These containers are designed to hold a singular element, offering flexibility in managing UI elements dynamically. Utilizing this approach allowed us to manipulate elements effectively, such as removing elements or replacing multiple elements simultaneously. In our implementation, we continuously update this container with an image representing the current frame received through sockets, providing the real time stream of the video to our frontend application. More detailed information regarding the socket usage will be expounded upon in the subsequent subsection.

3.4 Sockets

Often, software applications that integrate computer vision functionalities such as motion detection and video recording are implemented as monolithic entities. However, such an approach often results in tightly coupled components, leading to challenges in scalability, maintenance, and flexibility. As discussed in previous sections, separating the frontend (e.g., Streamlit-based user interface) from the backend (e.g., Python OpenCV-based computer vision logic) mitigates these issues by promoting modularity and abstraction.

One of the challenges encountered in this decoupled architecture is the efficient transfer of image data (frames) from the backend to the frontend for real-time display. Unlike a monolithic setup where passing frames as images within the application is straightforward, the decoupled nature necessitates a communication strategy that ensures seamless data exchange while maintaining loose coupling between components.

To address this challenge, we use sockets to establish a communication channel between Streamlit and Python OpenCV components. Sockets provide a

versatile and efficient solution for transmitting data over a network, making them well-suited for real-time applications such as computer vision systems.

In our implementation, each frame captured by the backend's camera feed is encoded into bytes and transmitted over the socket connection to the frontend. On the frontend side, the received bytes are decoded to reconstruct the corresponding image, which is then displayed to the user via the Streamlit interface.

While it may appear complex, this approach offers more benefits than challenges. One of the key advantages of using sockets in this context is the flexibility it offers in terms of frontend implementations. Whether the frontend is developed using Streamlit with Python or a JavaScript client utilizing frameworks like React, the socket-based communication remains agnostic to the frontend technology stack. This decoupling of communication logic from frontend specifics simplifies future enhancements or transitions to alternative frontend frameworks, as the core functionality relies on a standardized socket interface.

3.5 Email sending

The implementation of email alerts is achieved through Python's SMTP (Simple Mail Transfer Protocol) client library. This choice was made due to the simplicity and robustness of SMTP for sending emails programmatically. By using Python's built-in SMTP library, we ensure seamless communication with an SMTP server for email delivery.

For our implementation, we have configured an SMTP server using the services provided by Ukr.net [9]. Ukr.net offers reliable SMTP server capabilities, making it suitable for handling the email delivery aspect of our application. The SMTP server setup includes configuring the IMAP access to a created Ukr.net email address.

To ensure the security of email communications, we have implemented secure transmission protocols such as SSL (Secure Sockets Layer) when connecting to the SMTP server. This encryption layer adds a level of protection to sensitive information, including email contents and authentication credentials, during transmission over the network.

4 EXPERIMENTS

To assess the performance and robustness of our real-time motion detection and alert management system, a series of experiments under various scenarios was conducted. Our evaluation aimed to test the system's detection accuracy, processing speed, notification latency and reliability across different environmental conditions and setups.

4.1 Experimental Setup

The experiments were carried out using a hardware setup consisting of an Apple M1 MacBook Pro with 32 GB RAM and its built-in 1080p FaceTime HD camera. The software environment included Python 3.9 and an opencv-contrib-python library of version 4.9.0.80.

4.2 Scenarios

For real-world applicability, the experiments were designed to simulate conditions that vary in terms of lighting, background activity, and moving object speeds. The testing scenarios included:

1. Indoor environment (Normal light): This scenario tested the system's performance in typical indoor lighting conditions, such as those found in office spaces during daylight hours. This setup aimed to evaluate the detection accuracy and processing efficiency under optimal light conditions, which are expected in everyday indoor environments.
2. Indoor environment (Low light): This scenario tested the detection capability in dimly lit conditions inside a building during evening hours.
3. Disguised person: To test the system's ability to detect individuals wearing disguises such as masks, hoods, or hats, which could be common in

security-sensitive environments, this scenario introduced subjects wearing various headgear that partially hides their facial features. This test was designed to evaluate the detector's robustness in identifying human motion despite visual obstacles caused by clothing or accessories.

4. High-speed movements: Simulated by recording a person running or moving quickly or jumping through the frame to ensure that the system can accurately detect fast-moving objects.

4.3 Methodology

Each scenario was recorded using a high-definition camera at 1080p resolution. The system processed the video feeds in real time.

Performance metrics collected included:

1. Detection Accuracy: Measured as the percentage of correctly identified person movements versus missed detections.
2. Notification latency: The average time taken to receive an email notification, after a detected motion.

For a comprehensive analysis, each test was repeated five times. The results were averaged to minimize variability in the performance metrics.

4.4 Data collection

The data collection and analysis process involved an examination of the recordings of detections made after the conducted experiments. Each video contained a timestamp, enabling a precise comparison between the time of detection and the subsequent notification. This timestamp-based analysis facilitated the identification of any latency in the system's response, providing all the necessary data for calculating the defined metrics.

4.5 Results

4.5.1 Indoor environment (Normal light) scenario

In this scenario, our motion detection system achieved a 100% success rate in identifying the presence of a person. Across all test runs, the system consistently recognized and correctly identified human motion without any failures. This high level of performance proves the system's reliability and effectiveness in optimal lighting conditions, which are typical of many indoor settings such as offices, homes, and public buildings.

4.5.2 Indoor environment (Low light) scenario

In this scenario, the motion detection system successfully detected the presence of a person in three out of the five tests conducted, resulting in a 60% detection success rate. This indicates a noticeable decrease in performance compared to the 100% success rate observed in optimal lighting conditions.

The speed of detection in low light conditions was observed to be slightly slower compared to the normal light scenario. Additionally, there were two instances where the system registered false detections alongside the correct ones. These false positives show that the system can sometimes make mistakes caused by shadows or other low-light artifacts.

The performance in this scenario is significantly influenced by the camera's ability to capture clear images under poor lighting conditions. The current setup, while capable of functioning in low light, does not match the efficiency and accuracy seen in better-lit environments. The results show that by using a night vision camera we could improve detection accuracy and reduce false positives.

4.5.3 Disguised person scenario

In this scenario all the test cases were successfully passed. Despite the person wearing disguises that covered their face, such as masks, hoods, or hats, the system effectively identified and recognized the person's silhouette. This resulted in a 100% detection success rate across all the conducted tests.

4.5.4 High-speed movements

In this scenario our system successfully detected the person in four out of the five tests conducted. This corresponds to an 80% detection success rate.

It's important to note that the failed test involved extreme conditions that are unlikely to occur in practical surveillance settings.

4.6 Experiments conclusions

The main observation from our experiments is the system's reduced performance in low light conditions, as demonstrated by a 60% detection success rate, compared to a 100% success rate in normal lighting.

However, it's essential to note that the system maintained a high success rate even in scenarios involving high-speed movements and successfully detected people wearing masks and other obstacles in all the cases. These results demonstrate the system's effectiveness in handling dynamic and challenging situations.

A key takeaway from these experiments is a significant area of improvement for performance in low light conditions by using a night vision camera. By addressing this limitation, we can improve the system's reliability across a wider range of possible scenarios.

5 CONCLUSIONS

This research work set out to address the challenges of real-time human motion detection in video streams through designing and implementing a security system application using OpenCV and deep learning detectors. The results achieved from this study not only advance the domain of computer vision but also offer contributions to practical applications such as security surveillance and human-computer interaction.

The initial task involved a thorough analysis of existing detectors, where several primary models were evaluated for their performance in object detection. Our findings indicate that single-shot detectors, due to their speed and accuracy, are particularly effective for real-time analysis. Analyzing these detectors gave us a solid foundation to start implementing the application itself.

We then devoloped the complete system by carefully considering the technological stack, followed by a detailed description of implementing the key components such as the person detection mechanism, the detected clip writer etc.

Finally, we conducted a series of experiments to test our system under various scenarios, revealing its capabilities and areas for improvement. These experiments provided valuable insights into the performance of our application, showcasing its strengths and highlighting areas where enhancements could be made.

Having outlined our development and research process, we can now discuss the strengths of our solution, as well as the areas to improve.

The architecture of our application stands out as its strongest asset. Unlike conventional machine learning and computer vision applications, which often adopt a monolithic style, our application stands out for its architecturally competent approach. By separating Streamlit UI and Python OpenCV core logic into two independent parts communicating via API, we have created a scalable and adaptable architecture.

The decision to implement live video streaming from the backend to the frontend using sockets further enhances the application's robustness and real-time capabilities. This architecture not only improves the performance and reliability of our application but also makes it easier to modify and extend in the future. Allowing to seamlessly switch between different frontend implementations or create multiple UI versions, such as web and mobile interfaces.

Overall, the strength of our application lies in its forward-thinking architectural design, which sets it apart from traditional approaches and ensures the ease of its support and development in the future.

While our solution showcases notable strengths, there are still areas where further optimizations could be done to enhance its overall performance and versatility.

One crucial enhancement is to transition from using local file storage for saving recordings of detections, to a cloud storage solution, offering scalability, accessibility, and enhanced data management capabilities.

Additionally, integrating email providers instead of using plain SMTP can improve emailing logics. It is also a reasonable idea to expand notification channels by adding SMS messages, giving the users some choice.

Furthermore, conducting experiments with resource-constrained devices and with different cameras like CCTV or night vision ones would validate the system's adaptability and performance in more different environments.

Moreover, switching from Streamlit to a more flexible JavaScript client solution could provide greater customization and adaptability in developing a user interface of our system.

Lastly, replacing sockets with WebRTC for streaming frames from backend to UI offers advantages such as improved real-time communication and image quality, reduced latency and enhanced security, making it a better choice for live video streaming.

REFERENCES

[1] Ren et al., “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, arXiv:1506.01497, Jan. 2016.

[2] Redmon et al., “You Only Look Once: Unified, Real-Time Object Detection”, arXiv:1506.02640, May 2016.

[3] Liu et al., “SSD: Single Shot MultiBox Detector”, arXiv:1512.02325, Dec. 2016.

[4] OpenCV - Open Computer Vision Library, https://opencv.org/

[5] COCO - Common Objects in context, https://cocodataset.org/

[6] PASCAL Visual Object Classes, http://host.robots.ox.ac.uk/pascal/VOC/

[7] FastAPI, https://fastapi.tiangolo.com

[8] Streamlit, https://streamlit.io/

[9] ukr.net – Ukrainian emails, https://mail.ukr.net/