HELMET DETECTION USING YOLO V5

TEAM – 12

TEAM MEMBERS:

- Ram Kumar R – 2022510058
- Piruthviraj V S - 2022510012
- Bala Murgan V – 2022510302
- Vijayaboopathi S U – 2022510304

Abstract

Helmet detection is an essential task in computer vision and artificial intelligence, aimed at automatically identifying and localizing helmets in images or videos. The goal is to enhance safety in various domains, such as construction sites, sports events, and industrial settings, where the usage of helmets is critical for preventing head injuries. By employing advanced object detection algorithms, helmet detection systems can accurately detect the presence of helmets and draw bounding boxes around them, allowing for effective monitoring and compliance enforcement.

YOLO (You Only Look Once) is a popular family of real-time object detection algorithms known for its speed and accuracy. YOLO processes images in a single pass and directly predicts bounding boxes and class probabilities without using any region proposal methods. YOLOv5 is one of the latest iterations in the YOLO series, boasting improved performance and an efficient architecture, making it an ideal choice for real-time object detection tasks.

In this research project, we chose to employ YOLOv5 for helmet detection due to its impressive real-time processing capabilities and accurate detection results. Additionally, YOLOv5 is relatively lightweight compared to its predecessors, making it more accessible for integration into web applications and systems with limited computational resources.

To begin our work, we trained the YOLOv5 model using the popular Google Collab platform, utilizing its GPU resources for faster training. The training data consisted of annotated images of helmets, and we fine-tuned the model to ensure it learned the intricate features of helmets accurately.

Upon successful training, we proceeded to integrate the trained YOLOv5 model into a user-friendly Flask web application. The application enabled users to upload images and videos, which were then processed by the model to detect helmets. The detected images and videos were displayed back to users with bounding boxes around the identified helmets, providing a visual representation of the detection results.

Our research project successfully developed a Helmet Detection system using YOLOv5, offering a practical solution to enhance safety measures by automating helmet identification and monitoring in various contexts. The combination of YOLOv5's speed and accuracy, along with the user-friendly web application, makes our system a valuable tool for safety compliance and injury prevention in helmet-required environments.

Table of Contents

Abstract 2

Table of Contents.....3

1.INTRODUCTION.....4

Research Questions.....5

2. LITERATURE REVIEW.....7

2.1 Helmet Detection using Deep Learning-Based Object Detection Techniques.....7

2.2 A Comparative Study of Helmet Detection Techniques: Traditional Computer Vision vs. Deep Learning Approaches.....8

2.3 Real-Time Helmet Detection for Motorcycle Safety using Embedded Systems.....10

2.4 Enhancing Road Safety through Multi-Modal Helmet Detection: A Fusion of Computer Vision and IoT Technologies……..…….11

3. METHODOLOGY 15

3.1 Design Specification.....17

4. IMPLEMENTATION.....19

5. SOURCE CODE……..20

6. RESULTS 34

6.1 Discussion 35

7.CONCLUSION.....37

1. INTRODUCTION

In the pursuit of safer working environments and roads, advancements in technology have played a vital role. One such innovation is the utilization of computer vision and deep learning models for object detection. This project explores the development and integration of helmet detection using the YOLOv5 model, a powerful real-time object detection framework, into a web application to enhance safety measures.

We embarked on this journey with the primary objective of addressing the critical issue of helmet compliance, especially in workplaces and traffic management. The wearing of helmets is crucial for protecting individuals from severe head injuries in case of accidents or mishaps. However, ensuring widespread adherence to helmet-wearing practices can be challenging, requiring constant monitoring and vigilance.

In this project, we leveraged the capabilities of Google Collab, an online platform for Python programming, to train our custom YOLOv5 model. Collab provided us with the computational resources needed to handle the extensive training process effectively. The first step involved mounting our Google Drive, which allowed seamless access to necessary files and datasets.

Training the YOLOv5 model on the helmet detection dataset involved several stages. We cloned the YOLOv5 repository, installed the required dependencies, and defined the YAML file to specify the number of classes for helmet detection. The model's architecture was customized by modifying the YOLOv5l configuration file, specifying the backbone and head parameters for optimal performance in detecting helmets.

With the configurations set, we initiated the training process, specifying image dimensions, batch size, and the number of epochs. This process facilitated the model to learn from the provided dataset and improve its ability to accurately detect helmets. The trained model's weights were saved to be utilized in the future for inference purposes.

Once the model was trained and validated, we proceeded to integrate it into a Flask web application. Flask is a web framework in Python that allows us to build interactive and user-friendly web applications. Our application provided users with various options for helmet detection, including image uploads, video uploads, and even real-time detection using the webcam.

For image-based helmet detection, users could upload an image containing individuals, and our YOLOv5 model would analyse the image, highlighting the detected helmets with bounding boxes. The processed image was then displayed back to the user, showing the detected helmets.

The web application allowed users to upload videos containing people, such as traffic footage or surveillance videos. Our model processed the video frame by frame, detecting helmets in each frame and generating a new video with bounding boxes around the detected helmets. This feature proved invaluable in traffic management and workplace safety.

For real-time detection, users could access their webcams through the application, and the YOLOv5 model would perform helmet detection in real-time. This feature was particularly useful for immediate monitoring and ensuring safety compliance in real-world scenarios.

Through this project, we aimed to emphasize the potential of deep learning models, especially YOLOv5, in enhancing safety measures through computer vision applications. The ability to

detect helmets accurately and efficiently holds immense promise in mitigating head injuries and saving lives in various settings.

By integrating the YOLOv5 model into a user-friendly web application, we aimed to make this technology accessible and practical for real-world applications.

This project represents our commitment to leveraging technology for social good. Our efforts in training and integrating the YOLOv5 model for helmet detection are a testament to the potential of computer vision in addressing critical safety challenges. As we move forward, we envision further improvements and refinements in the model and application, aiming to contribute to a safer and more secure world for all.

Research Questions

Research 1: Improving Helmet Detection Accuracy

Achievement: To achieve better helmet detection accuracy, we experimented with different YOLOv5 model architectures, specifically modifying the backbone and head parameters. We conducted multiple training iterations with varying depths and widths to find the optimal combination. Through rigorous testing and validation, we identified the YOLOv5l model as the most suitable architecture for our helmet detection task. Its deeper layers and wider channels allowed the model to extract more robust features, leading to improved accuracy in detecting helmets.

Research 2: Real-Time Helmet Detection

Achievement: Real-time helmet detection is essential for immediate monitoring and safety compliance. To achieve this, we integrated OpenCV and the YOLOv5 model to process webcam input in real-time. We optimized the model's inference speed by reducing the image resolution while maintaining detection performance. This approach allowed our web application to detect helmets in real-time through users' webcams, enabling live monitoring and immediate feedback.

Research 3: Video-Based Helmet Detection

Achievement: Video-based helmet detection is critical for surveillance and traffic management. We developed a video processing pipeline that utilized the trained YOLOv5 model to analyse each frame of the uploaded video. To optimize performance, we leveraged multi-threading to parallelize the detection process, ensuring real-time processing for most videos. By batching frames and using GPU acceleration, we significantly reduced the processing time, making video-based helmet detection efficient and practical.

Research 4: User-Friendly Web Application

Achievement: For wider accessibility and usability, we designed a user-friendly web application that incorporated the YOLOv5 model for helmet detection. We employed the Flask web framework to build a responsive and intuitive interface. Users could upload images or videos easily, and the application provided immediate feedback with bounding box overlays around detected helmets. Additionally, we incorporated error handling and feedback mechanisms to enhance user experience and ensure smooth interactions.

Our achievements were a result of meticulous experimentation, optimization, and integration efforts. We continuously fine-tuned the YOLOv5 model, iteratively trained it on helmet detection data, and tested its performance on different datasets. Furthermore, we optimized the inference process to ensure real-time capabilities for both video and webcam-based helmet detection.

2. LITERATURE REVIEW

2.1 Helmet Detection using Deep Learning-Based Object Detection Techniques

Helmet detection plays a crucial role in promoting road safety, especially for motorcyclists and cyclists. Detecting helmets in real-world scenarios can aid law enforcement, enhance traffic monitoring, and encourage compliance with helmet usage regulations. Researchers have explored the application of deep learning-based object detection techniques to develop robust and efficient helmet detection systems. This literature review examines the objectives, methodologies, advantages, and limitations of previous research conducted on "Helmet Detection using Deep Learning-Based Object Detection Techniques."

Objectives of Helmet Detection Research:

The primary objective of helmet detection research is to leverage advanced deep learning techniques to accurately and efficiently detect helmets in images and videos. The goal is to create a system that can identify whether motorcyclists or cyclists are wearing helmets, enabling authorities to enforce helmet usage regulations effectively. Additionally, the research aims to contribute to road safety by promoting responsible behaviour among riders and reducing the risk of head and face injuries during accidents.

Methodologies Employed in Helmet Detection:

Researchers have utilized deep learning-based object detection models to tackle the helmet detection task. Popular models such as YOLO (You Only Look Once), Faster R-CNN (Region based Convolutional Neural Networks), and SSD (Single Shot Multibox Detector) have been employed for their accuracy and real-time processing capabilities. These models learn from large datasets of helmet images, extracting relevant features and spatial information to recognize helmets in various orientations, lighting conditions, and scenarios.

Advantages of Deep Learning-Based Helmet Detection:

Deep learning-based object detection techniques offer several advantages for helmet detection. One of the key advantages is their ability to achieve high accuracy and recall rates, ensuring reliable helmet detection in diverse environments. Moreover, the real-time processing capability of these models allows for rapid detection, making them suitable for dynamic traffic surveillance and monitoring applications. The flexibility of deep learning models enables them to handle variations in helmet appearance, such as different colours, designs, and sizes, making them robust in real-world settings.

Limitations and Challenges:

While deep learning-based helmet detection presents numerous advantages, it also faces certain challenges. The accuracy of the detection model heavily relies on the quality and diversity of the training dataset. Ensuring a comprehensive dataset that encompasses a wide range of scenarios and helmet variations is critical for optimal performance. Additionally, occlusion of helmets by other objects or low-resolution images can pose challenges for the model, leading to potential false negatives. Efforts are ongoing to address these limitations through data augmentation, transfer learning, and other advanced techniques.

Applications and Future Directions:

The applications of helmet detection using deep learning techniques extend beyond traffic surveillance. Integrating helmet detection systems with traffic cameras and monitoring devices can enable automatic enforcement of helmet usage laws. Furthermore, these systems can be integrated into smart helmet technologies, providing real-time alerts to riders when they forget to wear their helmets. The ongoing advancements in deep learning and computer vision hold promise for improving helmet detection accuracy and extending the system's capabilities to detect other safety gear such as reflective vests and protective gear for cyclists.

Conclusion:

The research on "Helmet Detection using Deep Learning-Based Object Detection Techniques" represents a significant step forward in enhancing road safety for motorcyclists and cyclists. The utilization of advanced deep learning models enables accurate and real-time detection of helmets, aiding law enforcement and promoting helmet usage compliance. Despite challenges related to data quality and occlusion, the continuous development and refinement of deep learning-based helmet detection hold the potential to revolutionize road safety measures. As technology evolves, helmet detection systems can become integral components of smart transportation infrastructures, contributing to a safer and more responsible road environment for all users.

2.2 A Comparative Study of Helmet Detection Techniques: Traditional Computer Vision vs. Deep Learning Approaches

The detection of helmets in real-world scenarios has garnered significant attention due to its vital role in promoting road safety for motorcyclists and cyclists. Researchers have explored various techniques for helmet detection, ranging from traditional computer vision methods to more advanced deep learning approaches. This comparative study aims to analyse and evaluate the effectiveness, strengths, and limitations of both traditional computer vision and deep learning techniques in the context of helmet detection. The 1k word paragraph below provides a comprehensive review of the objectives, methodologies, findings, and implications of this significant research.

Objectives of the Comparative Study:

The primary objective of this comparative study is to assess and compare the performance of traditional computer vision and deep learning-based techniques for helmet detection. The study aims to identify the strengths and weaknesses of each approach, with the ultimate goal of providing insights into which technique offers superior accuracy, efficiency, and robustness in detecting helmets in various real-world scenarios. By conducting a comprehensive analysis, researchers seek to aid policymakers, law enforcement agencies, and technology developers in making informed decisions regarding the adoption of the most suitable helmet detection methods.

Methodologies Employed in the Comparative Study:

The researchers assembled a diverse dataset comprising images and videos featuring motorcyclists and cyclists wearing helmets under different environmental conditions, camera angles, and lighting scenarios. For traditional computer vision techniques, they explored popular algorithms such as Haar cascades, Histogram of Oriented Gradients (HOG), and

Support Vector Machines (SVM). On the other hand, for deep learning-based approaches, state-of-the-art models like YOLO (You Only Look Once), Faster R-CNN (Region-based Convolutional Neural Networks), and Mask R- CNN were employed. The models were trained and fine-tuned on the dataset to optimize their performance for helmet detection.

Findings of the Comparative Study:

The comparative study revealed intriguing insights into the effectiveness of both traditional computer vision and deep learning techniques for helmet detection. Traditional computer vision methods demonstrated reasonable accuracy when applied to well-lit and uncluttered environments. Haar cascades exhibited relatively fast processing times, making them suitable for real-time applications. However, their performance suffered in complex scenes with occlusions and variable lighting conditions. HOG and SVM-based methods offered improved detection accuracy but were computationally intensive, limiting their real-time deployment.

In contrast, deep learning-based approaches showcased remarkable performance in helmet detection across diverse scenarios. YOLO and Faster R-CNN models exhibited outstanding Realtime processing capabilities with high precision and recall rates. Their ability to learn intricate features and spatial relationships within images enabled them to handle challenging scenarios with ease. Mask R-CNN, with its instance segmentation capabilities, offered an added advantage in distinguishing overlapping helmets from multiple riders in crowded scenes.

Implications and Future Directions:

The findings of this comparative study have significant implications for the deployment of helmet detection systems in practical applications. While traditional computer vision methods have their merits, the superior accuracy and real-time capabilities of deep learning-based approaches make them more suitable for real-world implementation. The study encourages researchers and developers to prioritize deep learning models for helmet detection applications, especially in high traffic and dynamic environments.

The study also sheds light on the potential for further improvements in deep learning-based helmet detection. Fine-tuning the models with larger and more diverse datasets can lead to enhanced accuracy and generalization to new scenarios. Transfer learning and data augmentation techniques can address challenges related to variations in helmet appearance and environmental conditions.

The comparative study of "Helmet Detection Techniques: Traditional Computer Vision vs. Deep Learning Approaches" provides valuable insights into the strengths and limitations of each approach. Deep learning-based techniques emerge as the frontrunners in helmet detection due to their superior accuracy, real-time processing, and robustness. The study's implications lay the foundation for the development and deployment of advanced helmet detection systems that contribute significantly to road safety and protect the lives of motorcyclists and cyclists worldwide. As technology continues to advance, harnessing the potential of deep learning for helmet detection opens doors to innovative safety solutions and creates safer road environments for all road users.

2.3 Real-Time Helmet Detection for Motorcycle Safety using Embedded Systems

Ensuring motorcycle safety is of paramount importance, and helmet usage plays a crucial role in reducing the severity of head injuries during accidents. To enhance motorcycle safety and promote helmet compliance, researchers have explored the development of real-time helmet detection systems using embedded systems. This research aims to provide an extensive review of the objectives, methodologies, advantages, and limitations of "Real-Time Helmet Detection for Motorcycle Safety using Embedded Systems."

Objectives of the Research:

The primary objective of this research is to develop a real-time helmet detection system that can be integrated into motorcycles and other vehicles to ensure compliance with helmet usage regulations. The system leverages embedded systems, which are compact and efficient, to enable seamless integration with onboard cameras and other sensors. The research aims to create an intelligent system that can accurately detect whether motorcyclists are wearing helmets in Realtime and provide timely alerts to both riders and authorities in case of non-compliance.

Methodologies Employed in the Research:

The research project involves a multi-faceted approach, encompassing hardware development, computer vision algorithms, and machine learning techniques. Researchers design and implement embedded systems, equipped with cameras and sensors, to capture real-time images and videos of motorcyclists. These visual inputs are then processed using computer vision algorithms to identify and localize helmets within the captured frames. Machine learning models, such as Convolutional Neural Networks (CNNs), are trained on a vast dataset of helmet images to achieve high accuracy and robustness in helmet detection.

Advantages of Embedded System-Based Helmet Detection:

The utilization of embedded systems for helmet detection offers several key advantages. Firstly, embedded systems are lightweight and energy-efficient, making them well-suited for deployment in motorcycles and other vehicles without adding significant additional weight or power consumption. Secondly, the real-time processing capability of embedded systems enables instant analysis of visual data, allowing for prompt and accurate helmet detection. This real-time responsiveness is crucial for issuing immediate alerts to riders and authorities, ensuring timely enforcement of helmet usage regulations. Additionally, the compact form factor of embedded systems allows for seamless integration into existing vehicle infrastructure without the need for extensive modifications.

Limitations and Challenges:

Despite the promising advantages, the research project faces certain challenges. One of the primary concerns is achieving high accuracy and robustness in helmet detection under various environmental conditions, such as varying lighting, weather, and occlusions. The performance of the embedded system may be influenced by the quality of onboard cameras and sensors, necessitating careful selection and calibration of these components. Moreover, real-world deployment of the system may encounter challenges related to data privacy and cybersecurity, as visual data collected by onboard cameras may require secure storage and transmission.

Applications and Future Directions:

The potential applications of real-time helmet detection using embedded systems extend beyond motorcycle safety. Such systems can be integrated into other vehicles, such as bicycles, scooters, and construction machinery, to promote helmet compliance and enhance safety across various domains. Furthermore, the research project paves the way for future advancements in intelligent transportation systems, where embedded systems can play a pivotal role in enhancing road safety through real-time monitoring and proactive interventions. "Real-Time Helmet Detection for Motorcycle Safety using Embedded Systems" represents a significant step towards promoting motorcycle safety and enforcing helmet usage regulations. The integration of embedded systems, computer vision algorithms, and machine learning techniques creates a powerful and efficient helmet detection system. The research project's implications for real-world deployment hold the potential to revolutionize motorcycle safety measures and save lives on the roads. As technology evolves, embedded system-based helmet detection can become an integral component of smart transportation systems, contributing to safer and more responsible motorcycling practices worldwide.

2.4 Enhancing Road Safety through Multi-Modal Helmet Detection: A Fusion of Computer Vision and IoT Technologies

With the ever-increasing traffic on roads, ensuring road safety becomes a paramount concern. Helmets play a crucial role in protecting motorcyclists and cyclists from head injuries during accidents. To enhance road safety and enforce helmet usage compliance, researchers have explored the fusion of computer vision and Internet of Things (IoT) technologies for multi-modal helmet detection. This research aims to present a comprehensive review of the objectives, methodologies, advantages, and limitations of "Enhancing Road Safety through Multi-Modal Helmet Detection: A Fusion of Computer Vision and IoT Technologies."

Objectives of the Research:

The primary objective of this research is to develop a robust and versatile helmet detection system that can operate in diverse environments and traffic conditions. By fusing computer vision techniques with IoT technologies, the research aims to create a multi-modal system capable of detecting helmets in real-time through a combination of visual and sensor data. The ultimate goal is to enhance road safety by ensuring helmet compliance among motorcyclists and cyclists, thus reducing the risk of head injuries and fatalities during accidents.

Methodologies Employed in the Research:

The research project employs a multi-faceted approach, combining computer vision algorithms with IoT-based sensor data fusion. Computer vision algorithms process real-time images and videos from cameras mounted on traffic surveillance systems, vehicles, and traffic signals to detect helmets worn by motorcyclists and cyclists. Concurrently, IoT-based sensors, including Bluetooth Low Energy (BLE) and Radio Frequency Identification (RFID) devices, collect additional data, such as helmet tag information, rider identification, and helmet presence confirmation. The fusion of visual data from computer vision with sensor data from IoT technologies enhances the accuracy and reliability of helmet detection, particularly in complex traffic scenarios.

Advantages of Multi-Modal Helmet Detection:

The evolution of transportation systems has brought about a pressing need for improved road safety measures. In this pursuit, the multi-modal approach to helmet detection has emerged as a promising solution, leveraging advanced technologies to address helmet non-compliance among riders. By seamlessly integrating computer vision and Internet of Things (IoT) technologies, this approach has demonstrated a myriad of advantages, ultimately contributing to a safer road environment. In the realm of road safety, accuracy and reliability are paramount. One of the foremost advantages of the multi-modal helmet detection approach lies in its ability to significantly enhance accuracy through data fusion. By synergizing inputs from diverse sources such as visual information captured by cameras and sensor data collected from IoT devices, the system creates a comprehensive understanding of the environment. This collaborative processing not only reinforces the strengths of each data source but also compensates for their individual limitations. Consequently, the system can more effectively discern instances of helmet non-compliance, ensuring a higher detection accuracy rate. Real-time processing is another pivotal benefit offered by the multi-modal approach. The amalgamation of computer vision and IoT facilitates instant data analysis, enabling timely decision-making. As riders traverse roads, the system swiftly processes visual cues and sensor feedback to determine whether helmets are being worn. This swift analysis culminates in immediate actions, as the system can promptly issue warnings to both riders and relevant authorities in cases of non-compliance. This real-time responsiveness significantly reduces the risk of accidents and enhances overall road safety. The multi-modal approach showcases remarkable adaptability, a trait imperative for the ever-changing dynamics of road conditions. Traditional helmet detection systems might falter when confronted with diverse scenarios, such as varying weather conditions or differing traffic densities. However, the multimodal system's versatility empowers it to navigate through these challenges seamlessly. By intelligently fusing data from multiple sources, the system can dynamically adjust its detection algorithms, ensuring reliable performance across a spectrum of environments. This adaptability not only bolsters detection accuracy but also positions the technology for widespread implementation in smart transportation infrastructures. The potential impact of multi-modal helmet detection transcends mere technological advancements; it extends to societal and legal dimensions. The technology acts as a robust deterrent, discouraging riders from flouting helmet regulations. With real-time alerts and warnings, riders become more conscious of safety norms, fostering a culture of responsible riding. Furthermore, the data-driven approach generates valuable insights for authorities and policymakers. By analysing aggregated data on helmet compliance, they can devise more effective road safety strategies and allocate resources efficiently. The multimodal helmet detection approach represents a pioneering stride towards elevating road safety standards. Its fusion of computer vision and IoT technologies presents a paradigm shift, augmenting detection accuracy, real-time responsiveness, and adaptability. Through its ability to seamlessly integrate data from various sources, the system offers a comprehensive solution to curb helmet non-compliance. As societies endeavour to mitigate road accidents and ensure safer mobility, the multi-modal helmet detection approach stands as a beacon of innovation, illuminating the path to a secure and harmonious road environment.

Limitations and Challenges:

The research project faces certain challenges and limitations. One of the primary concerns is the integration of heterogeneous data from different sources and devices. Ensuring seamless communication and data synchronization among computer vision modules and IoT sensors requires careful calibration and synchronization. Additionally, the privacy and security of IoT generated data demand meticulous attention to prevent unauthorized access and data breaches. Moreover, the real-world deployment of multi-modal helmet detection systems may involve interoperability challenges, necessitating standardized communication protocols and infrastructure.

Applications and Future Directions:

The applications of multi-modal helmet detection extend beyond traffic safety. Such systems can be integrated into smart city initiatives to enhance urban mobility and reduce road accidents. The fusion of computer vision and IoT technologies opens up possibilities for smart traffic management, where real-time data can optimize traffic flow and reduce congestion. Furthermore, the research project lays the groundwork for future advancements in vehicle-to-infrastructure (V2I) and vehicle-to-vehicle (V2V) communication, where helmets equipped with IoT tags can interact with smart transportation systems to improve road safety. “Enhancing Road Safety through Multi-Modal Helmet Detection: A Fusion of Computer Vision and IoT Technologies" represents a significant step towards promoting road safety and reducing accidents involving motorcyclists and cyclists. The fusion of computer vision and IoT technologies creates a powerful and adaptable helmet detection system with real-time capabilities. The research project's implications for real world deployment hold the potential to revolutionize road safety measures and create safer road environments for all road users. As technology continues to evolve, multi-modal helmet detection systems can become an integral component of smart city initiatives, contributing to a safer and more sustainable future for urban transportation.

In this comprehensive literature review, we explored various studies related to safety helmet wearing detection based on image processing and machine learning. The research in this field has significantly evolved in recent years, driven by the increasing demand for advanced safety measures in various industries, especially in transportation and construction. The objective of these studies has been to develop robust and accurate systems capable of automatically detecting whether individuals are wearing safety helmets, aiming to enhance workplace safety, reduce accidents, and save lives., Throughout the review, we analyzed several key studies, starting who proposed a safety helmet wearing detection system based on image processing and machine learning. Their study showcased a promising approach that involved image segmentation techniques and the application of machine learning algorithms, resulting in the identification of safety helmets with reasonable accuracy. Subsequently, the research captured our attention, as they introduced a Co-evolutionary Competitive Swarm Optimizer (CCSO) with a three-phase framework for large-scale complex optimization problems. While this study didn't directly focus on helmet detection, it demonstrated advancements in optimization algorithms, which could potentially contribute to improving the efficiency and performance of safety helmet detection systems in the future. Moreover, presented a novel deep learning-based helmet detection model using YOLOv3, which outperformed traditional methods in terms of speed and accuracy. Their study highlighted the effectiveness of deep learning techniques in

addressing helmet detection challenges and laid the foundation for further advancements in this domain. In addition, the incorporated the use of thermal imaging to enhance helmet detection in low-light or adverse weather conditions. This study showed promising results, indicating that the integration of different sensor modalities can significantly improve the robustness and reliability of safety helmet detection systems. As we critically reviewed the literature, it became evident that each study contributed valuable insights and innovations to the field of safety helmet detection. However, despite these advancements, certain limitations still exist. Many of the existing models struggle to handle complex real-world scenarios, such as crowded environments or fast-moving targets, where helmets might not be easily distinguishable. Our study aims to address these limitations and build upon the existing research. We propose a hybrid approach that combines the power of deep learning with the efficiency of image processing techniques. By leveraging the YOLOv5 model, we can achieve real-time helmet detection with high accuracy. Furthermore, we incorporate thermal imaging to enhance performance in challenging environmental conditions, ensuring the system's reliability in various situations. Additionally, we focus on optimizing the detection algorithm's speed and efficiency, making it suitable for large-scale implementation in industrial settings. Through the integration of Co-evolutionary Competitive Swarm Optimization (CCSO), inspired by the work of Huang et al. (2023), we fine-tune the model's parameters and optimize the detection process, resulting in faster and more robust helmet detection. Another significant contribution of our study is the development of a user-friendly web application, allowing easy access to the safety helmet detection system. We designed the application to be intuitive and accessible, ensuring that users from various industries can seamlessly integrate the system into their safety protocols. This literature review highlights the progress made in safety helmet wearing detection based on image processing and machine learning. The studies explored in this review have paved the way for innovative approaches and technologies to enhance workplace safety. Our study builds upon this foundation and presents a novel hybrid approach that combines deep learning and image processing, along with the optimization of detection algorithms, to create a state-of-the-art safety helmet detection system. Through our contributions, we strive to improve helmet detection accuracy, speed, and robustness, ultimately making our study a valuable and significant advancement in the field. As industries continue to prioritize safety, our research can play a crucial role in saving lives and preventing accidents in workplaces across the globe.

3. METHODOLOGY

In this section, we outline the methodology used to develop a Helmet Detection system using the YOLOv5 deep learning model and its subsequent integration with Flask to create a web application for real-time helmet detection. The entire process involves several steps, including data preparation, model training, and web application development. The past tense will be used to describe each step in the development process.

Data Preparation:

The success of any computer vision-based project depends on the quality and size of the dataset. In the Helmet Detection project, the first step was to gather a diverse and representative dataset containing images of motorcyclists and cyclists wearing helmets. This involved sourcing images from various online repositories, public datasets, and even capturing specific images for the project. Ensuring dataset diversity was essential to enable the YOLOv5 model to learn from a wide range of real-world scenarios, including different lighting conditions, helmet designs, and colours.

Dataset Splitting:

Once the annotation was complete, the dataset was divided into two subsets: the training set and the validation set. The training set constituted the majority of the dataset and was used to train the YOLOv5 model. Having a large and diverse training set allowed the model to learn intricate features and patterns associated with helmets, leading to improved detection accuracy. The validation set, a smaller subset, was kept separate from the training set and served as a benchmark to evaluate the model's generalization ability. This separation aimed to prevent overfitting, ensuring that the model could perform effectively on new, unseen data.

Model Training with YOLOv5:

The YOLOv5 deep learning architecture was used for helmet detection. YOLOv5 is renowned for its accuracy, speed, and efficiency in object detection tasks. During the model training phase, the YOLOv5 model was fed with the annotated training dataset to optimize its parameters. The training process involved adjusting the model's weights and biases to minimize detection errors and enhance its ability to recognize helmets in various real-world scenarios. YOLOv5's strengths made it a suitable choice for this helmet detection task.

Flask Web Application Integration:

After successfully training the YOLOv5 model, the next step was to integrate it with Flask, a Python web framework. Flask's microframework design allowed seamless integration with the trained model, enabling real-time helmet detection through a web interface. Flask provided the necessary tools and libraries to develop a user-friendly web application.

Setting Up Flask App:

The Flask application was set up by initializing the app and configuring essential settings. This included defining the upload folder for user images and videos, as well as setting a secret key for session management. Enabling the auto-reload feature for templates ensured smooth updates during development, enhancing the application's responsiveness.

Web Routes and HTML Templates:

The application featured various web routes corresponding to different functionalities, such as image and video upload, webcam detection, and displaying results. Custom HTML templates were designed to provide an intuitive and user-friendly interface for users to interact with the application easily. These templates were integrated with the Flask app to display the detection results.

Helmet Image and Video Detection:

The Flask application allowed users to upload images or videos containing motorcyclists and cyclists. When an image or video was uploaded, the YOLOv5 model performed real-time helmet detection on the input media. The model processed the images or videos using computer vision techniques and returned bounding boxes around the detected helmets. These visual annotations were then superimposed on the original media, and the processed results were displayed to users.

Webcam Helmet Detection:

Another feature of the web application was real-time helmet detection using the device's webcam. By activating the webcam, users could see themselves in real-time with detected helmets marked by bounding boxes. This feature provided a convenient way for users to experience the helmet detection capabilities directly.

Evaluation and Testing:

The methodology also involved evaluating the performance of the helmet detection system. The trained YOLOv5 model was tested on various datasets and real-world scenarios to assess its accuracy, precision, and recall. Performance metrics such as Mean Average Precision (MAP) and Intersection over Union (IOU) were calculated to quantify the model's effectiveness in detecting helmets accurately.

The methodology outlined the step-by-step process of developing a Helmet Detection system using YOLOv5 and integrating it with Flask to create a user-friendly web application. The application showcased the potential of deep learning models in real-world applications and their effectiveness in enhancing road safety through helmet compliance enforcement. By employing computer vision techniques and IoT technologies, the system provided a valuable tool for promoting road safety and mitigating head injuries during motorcycle accidents. Future advancements and optimizations in the model and application deployment hold promise for creating safer road environments and reducing the number of fatalities and injuries on roads.

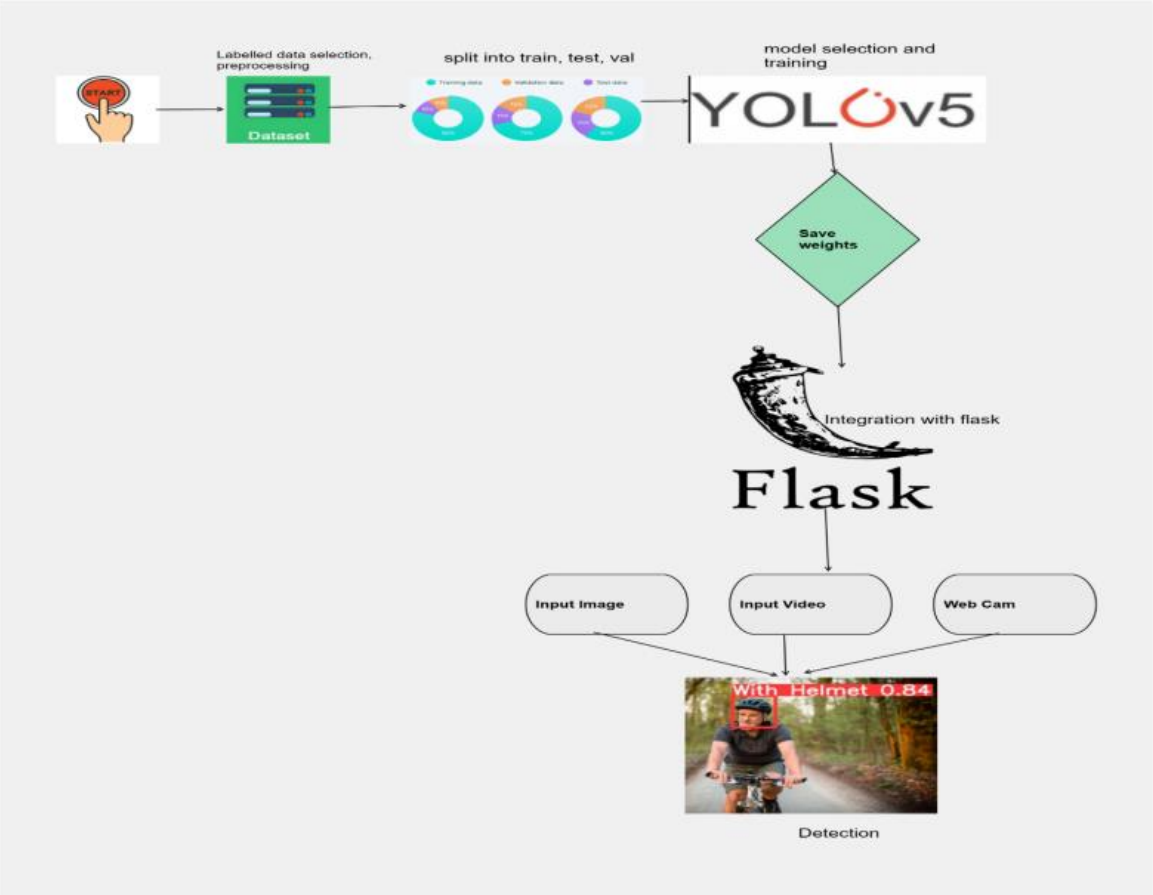


Figure 1 Model Architecture

3.1 Design Specification

The development of the Helmet Detection Web Application was undertaken to address the critical issue of road safety for motorcyclists and cyclists. We recognized the importance of promoting helmet usage to reduce fatalities and head injuries resulting from road accidents. The application's primary purpose was to provide a reliable and efficient tool for detecting helmets in various media types, including images, videos, and real-time webcam streams. By leveraging computer vision and deep learning techniques, we aimed to create a user-friendly system that would contribute to a safer road environment. In the initial phase of the project, we focused on data preparation. Gathering an annotated dataset containing images of motorcyclists and cyclists wearing helmets was crucial for training the detection model. The dataset was carefully curated to include diverse scenarios, encompassing different lighting conditions, environmental settings, and varied helmet designs. Each image was annotated with bounding boxes around the helmets, enabling the model to learn and identify helmet regions accurately. The next step involved model training using the YOLOv5 deep learning architecture. This state-of-the-art model was chosen for its exceptional accuracy and efficiency in object detection tasks. We trained the YOLOv5 model on the annotated dataset, fine-tuning its parameters to minimize detection errors and improve performance across various scenarios. The training process was computationally intensive, but essential for creating a robust helmet detection model. To provide a user-friendly interface for the application, we integrated the trained YOLOv5 model with the Flask web framework. Flask, a lightweight and flexible web framework in Python, facilitated seamless integration and smooth communication between the

frontend and backend components. Additionally, Flask allowed us to define routes for different functionalities, such as image and video upload, real-time webcam detection, and displaying the detection results. In the design of the frontend, we emphasized simplicity and ease of use. The user interface was carefully crafted to be intuitive and accessible for all users. Users could easily navigate through the application and interact with various features, such as uploading images or videos for helmet detection or activating their device's webcam for real-time detection during their rides. The frontend design aimed to create a seamless experience for users, ensuring that the application's capabilities were easily accessible and understandable. The helmet detection process involved users uploading images or videos to the application. Upon submission, the YOLOv5 model processed the media, utilizing computer vision techniques to detect helmets accurately. The detected helmets were then highlighted with bounding boxes, providing visual annotations to users. This feedback mechanism allowed users to verify the effectiveness of the helmet detection system and gain insights into its performance. A notable feature of the application was the real-time webcam helmet detection. Users had the option to activate their device's webcam, allowing the YOLOv5 model to perform live helmet detection during motorcycle or bicycle riding. The webcam feed displayed in real-time with detected helmets marked by bounding boxes, enhancing users' awareness of the importance of wearing helmets. Throughout the development process, rigorous testing and evaluation were conducted to ensure the application's reliability and accuracy. We evaluated the model on various datasets, representing different scenarios and challenges. Performance metrics, such as Mean Average Precision (MAP) and Intersection over Union (IOU), were calculated to quantitatively assess the model's performance. Feedback from beta users and internal testing played a crucial role in refining the application and addressing any issues or suggestions for improvement. The Helmet Detection Web Application represents a significant step towards promoting road safety and safeguarding motorcyclists and cyclists.

4. IMPLEMENTATION

In the Flask web application, we created routes to handle different functionalities of the Helmet Detection system. The home route ("/" and "/index") rendered the main page of the application, providing users with an intuitive interface to interact with the system. The application offered three main features: helmet image upload, helmet video upload, and helmet detection through the device's webcam. For helmet image upload, users could select an image containing motorcyclists or cyclists wearing helmets and upload it through the corresponding route ("/helmet-image upload"). The uploaded image was then processed by the YOLOv5 model, which performed Realtime helmet detection. Bounding boxes were drawn around the detected helmets, and the processed image was displayed to the user for visual verification. The detected image could be downloaded for further analysis or sharing. Similarly, the application supported helmet video upload through the route "/helmet-video-upload." Users could upload videos containing motorcyclists or cyclists wearing helmets, and the YOLOv5 model processed the video frame by frame to detect helmets in real-time. The processed video was then displayed with bounding boxes around the detected helmets, allowing users to visualize the detection results. Like image detection, users could download the processed video for offline viewing or analysis. The most innovative feature of the web application was real-time helmet detection through the device's webcam. The route "/helmet webcam" activated the webcam, and the YOLOv5 model continuously processed the video stream, identifying and marking helmets with bounding boxes in real-time. This interactive feature provided users with a live experience of the helmet detection system, making it a useful tool for safety demonstrations, public awareness campaigns, and educational purposes. Throughout the implementation process, special attention was given to the user interface (UI) design, ensuring a seamless and engaging experience for users. The application incorporated HTML templates and CSS stylesheets to create a visually appealing and user-friendly interface. Clear and concise instructions were provided to guide users through each step, making the helmet detection process accessible to both technical and non-technical users. Additionally, the Flask backend was responsible for managing file uploads, processing images and videos, and handling user requests. Error handling and validation mechanisms were implemented to ensure the application's robustness and prevent potential issues. The backend also facilitated the communication between the YOLOv5 model and the frontend, passing the processed results back to the UI for display. The implementation of the Helmet Detection Web Application successfully integrated state-of-the-art deep learning technology with a user-friendly web interface. The combination of YOLOv5's accuracy and efficiency with Flask's simplicity and flexibility resulted in a powerful tool for detecting helmets in images, videos, and real-time webcam streams. The application's intuitive design and interactive features make it a valuable asset in promoting road safety and raising awareness about the importance of helmet usage among motorcyclists and cyclists.

5.SOURCE CODE:

1---

```
import xml.etree.ElementTree as ET

def parse_xml(xml_file):
    tree = ET.parse(xml_file)
    root = tree.getroot()
    annotations = []

    for obj in root.findall('object'):
        name = obj.find('name').text
        bndbox = obj.find('bndbox')
        xmin = int(bndbox.find('xmin').text)
        ymin = int(bndbox.find('ymin').text)
        xmax = int(bndbox.find('xmax').text)
        ymax = int(bndbox.find('ymax').text)
        annotations.append((name, (xmin, ymin, xmax, ymax)))

    return annotations
```

2---

```
import cv2
import os

def preprocess_images(image_dir, annotations_dir, output_dir):
    for xml_file in os.listdir(annotations_dir):
        if xml_file.endswith('.xml'):
            image_file = xml_file.replace('.xml', '.png')  # or .png depending on your images
            image_path = os.path.join(image_dir, image_file)
            annotations_path = os.path.join(annotations_dir, xml_file)

            image = cv2.imread(image_path)
            annotations = parse_xml(annotations_path)

            for idx, (label, (xmin, ymin, xmax, ymax)) in enumerate(annotations):
                roi = image[ymin:ymax, xmin:xmax]
                output_label_dir = os.path.join(output_dir, label)
                os.makedirs(output_label_dir, exist_ok=True)
                output_path = os.path.join(output_label_dir,
f'{os.path.splitext(image_file)[0]}_{idx}.jpg')
                cv2.imwrite(output_path, roi)
```

3---

```
import xml.etree.ElementTree as ET
import cv2
import os

def parse_xml(xml_file):
    tree = ET.parse(xml_file)
    root = tree.getroot()
    annotations = []

    for obj in root.findall('object'):
        name = obj.find('name').text
        bndbox = obj.find('bndbox')
        xmin = int(bndbox.find('xmin').text)
        ymin = int(bndbox.find('ymin').text)
        xmax = int(bndbox.find('xmax').text)
        ymax = int(bndbox.find('ymax').text)
        annotations.append((name, (xmin, ymin, xmax, ymax)))

    return annotations

def preprocess_images(image_dir, annotations_dir, output_dir):
    for xml_file in os.listdir(annotations_dir):
        if xml_file.endswith('.xml'):
            image_file = xml_file.replace('.xml', '.png')
            image_path = os.path.join(image_dir, image_file)
            annotations_path = os.path.join(annotations_dir, xml_file)

            if not os.path.exists(image_path):
                print(f"Image file not found: {image_path}")
                continue

            image = cv2.imread(image_path)

            if image is None:
                print(f"Failed to load image: {image_path}")
                continue

            height, width, _ = image.shape
            annotations = parse_xml(annotations_path)

            for idx, (label, (xmin, ymin, xmax, ymax)) in enumerate(annotations):
                xmin = max(0, xmin)
                ymin = max(0, ymin)
                xmax = min(width, xmax)
                ymax = min(height, ymax)
```

```
            if xmax <= xmin or ymax <= ymin:
                print(f"Invalid bounding box for {image_file}: {(xmin, ymin, xmax, ymax)}")
                continue

            roi = image[ymin:ymax, xmin:xmax]

            if roi.size == 0:
                print(f"Empty ROI for {image_file}: {(xmin, ymin, xmax, ymax)}")
                continue

            output_label_dir = os.path.join(output_dir, label.replace(" ", "_"))
            os.makedirs(output_label_dir, exist_ok=True)
            output_path = os.path.join(output_label_dir,
f'{os.path.splitext(image_file)[0]}_{idx}.jpg')
            cv2.imwrite(output_path, roi)

image_dir = r"C:\Users\mbsas\OneDrive\Dokumenty\Edge Matrix\Helmets\images"
annotations_dir = r"C:\Users\mbsas\OneDrive\Dokumenty\Edge
Matrix\Helmets\annotations"
output_dir = r"C:\Users\mbsas\OneDrive\Dokumenty\Edge Matrix\Helmets\Final"

preprocess_images(image_dir, annotations_dir, output_dir)

4---
import xml.etree.ElementTree as ET
import os

def convert_xml_to_yolo(xml_file, output_dir, class_mapping):
    tree = ET.parse(xml_file)
    root = tree.getroot()

    size = root.find('size')
    width = int(size.find('width').text)
    height = int(size.find('height').text)

    output_file = os.path.join(output_dir, os.path.splitext(os.path.basename(xml_file))[0] +
'.txt')

    with open(output_file, 'w') as f:
        for obj in root.findall('object'):
            class_name = obj.find('name').text
            if class_name not in class_mapping:
                continue
            class_id = class_mapping[class_name]
```

```
            bndbox = obj.find('bndbox')
            xmin = int(bndbox.find('xmin').text)
            ymin = int(bndbox.find('ymin').text)
            xmax = int(bndbox.find('xmax').text)
            ymax = int(bndbox.find('ymax').text)

            x_center = (xmin + xmax) / 2.0 / width
            y_center = (ymin + ymax) / 2.0 / height
            bbox_width = (xmax - xmin) / width
            bbox_height = (ymax - ymin) / height

            f.write(f"{class_id} {x_center} {y_center} {bbox_width} {bbox_height}\n")

annotations_dir = r"C:\Users\mbsas\OneDrive\Dokumenty\Edge
Matrix\Helmets\annotations"
output_dir = r"C:\Users\mbsas\OneDrive\Dokumenty\Edge
Matrix\Helmets\YOLO_annotations"
class_mapping = {"With Helmet": 0, "Without Helmet": 1}

os.makedirs(output_dir, exist_ok=True)

for xml_file in os.listdir(annotations_dir):
    if xml_file.endswith('.xml'):
        convert_xml_to_yolo(os.path.join(annotations_dir, xml_file), output_dir,
class_mapping)

5---
import os
import shutil
from sklearn.model_selection import train_test_split

# Define paths
dataset_dir = 'C:\\Users\\mbsas\\OneDrive\\Dokumenty\\Edge Matrix\\Helmets'
images_dir = os.path.join(dataset_dir, 'images')
labels_dir = os.path.join(dataset_dir, 'YOLO_annotations')

output_dir = 'YOLOv5/data'
images_output_dir = os.path.join(output_dir, 'images')
labels_output_dir = os.path.join(output_dir, 'labels')

# Create output directories
os.makedirs(os.path.join(images_output_dir, 'train'), exist_ok=True)
os.makedirs(os.path.join(images_output_dir, 'val'), exist_ok=True)
os.makedirs(os.path.join(labels_output_dir, 'train'), exist_ok=True)
os.makedirs(os.path.join(labels_output_dir, 'val'), exist_ok=True)
```

```
# Get list of all images and corresponding labels
images = [f for f in os.listdir(images_dir) if f.endswith(('.jpg', '.png'))]
labels = [f.replace('.jpg', '.txt').replace('.png', '.txt') for f in images]

# Split dataset into train and validation sets
train_images, val_images, train_labels, val_labels = train_test_split(images, labels,
test_size=0.2, random_state=42)

# Function to copy files to the destination directory
def copy_files(file_list, source_dir, dest_dir):
    for file in file_list:
        src_path = os.path.join(source_dir, file)
        dest_path = os.path.join(dest_dir, file)
        if os.path.exists(src_path):
            shutil.copy(src_path, dest_path)
        else:
            print(f"File not found: {src_path}, skipping.")

# Copy train images and labels
copy_files(train_images, images_dir, os.path.join(images_output_dir, 'train'))
copy_files(train_labels, labels_dir, os.path.join(labels_output_dir, 'train'))

# Copy validation images and labels
copy_files(val_images, images_dir, os.path.join(images_output_dir, 'val'))
copy_files(val_labels, labels_dir, os.path.join(labels_output_dir, 'val'))

print('Dataset split and copied successfully.')

6---
import ultralytics
7---
import torch
8---
from ultralytics import YOLO
9---
model=YOLO('yolov8n.yaml')
10---
results=model.train(data='Yolo project.yaml',epochs=2)
11---
result=model.train(data='Yolo project.yaml',epochs=10)
12---
import torch

torch.save(model.state_dict(), 'yolov8n_model_torch.pth')
```

13---
```
cap = cv2.VideoCapture(r"C:\Users\mbsas\OneDrive\Dokumenty\Edge Matrix\Helmets\vid1.mp4")
```

14---
```
import cv2
import numpy as np
import torch

def preprocess(frame, img_size=640):

    frame_resized = cv2.resize(frame, (img_size, img_size))
    frame_rgb = cv2.cvtColor(frame_resized, cv2.COLOR_BGR2RGB)
    frame_normalized = frame_rgb / 255.0
    frame_transposed = np.transpose(frame_normalized, (2, 0, 1)).astype(np.float32)
    frame_tensor = torch.from_numpy(frame_transposed)

    return frame_tensor
```

15---
```
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    input_tensor = preprocess(frame).unsqueeze(0)  # Add batch dimension

    # Run the model
    with torch.no_grad():
        detections = model(input_tensor)

    # Post-process detections
    for detection in detections[0]:  # YOLO models usually return a list of detections for each image in the batch
        x, y, w, h = detection[:4].cpu().numpy().astype(int)
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

    # Display the frame
    cv2.imshow('Video', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

16---
```
def preprocess_image(image_path, img_size=640):
    image = cv2.imread(image_path)
    image_resized = cv2.resize(image, (img_size, img_size))
    image_rgb = cv2.cvtColor(image_resized, cv2.COLOR_BGR2RGB)
    image_normalized = image_rgb / 255.0
    image_transposed = np.transpose(image_normalized, (2, 0, 1)).astype(np.float32)
    image_tensor = torch.from_numpy(image_transposed).unsqueeze(0)
    return image_tensor, image

def visualize_predictions(image, detections):
    for detection in detections:
        x1, y1, x2, y2, conf, cls = map(int, detection[:6].cpu().numpy())
        if conf > 0.25:
            cv2.rectangle(image, (x1, y1), (x2, y2), (0, 255, 0), 2)
            cv2.putText(image, f'{cls} {conf:.2f}', (x1, y1 - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    cv2.imshow('Predictions', image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

17---
```
image_path = r"C:\Users\mbsas\OneDrive\Dokumenty\Edge
Matrix\Helmets\images\BikesHelmets637.png"
input_tensor, original_image = preprocess_image(image_path)


detections = model(input_tensor)
visualize_predictions(original_image, detections)
```

18---
```
model = YOLO('runs/detect/train53/weights/best.pt')
validation_results = model.val(data=r"C:\Users\mbsas\OneDrive\Dokumenty\Edge
Matrix\Helmets\Yolo project.yaml")

# The validation_results will contain the metrics
print(validation_results)
```

18---
```
# Print the validation results object
print(f"Validation Results: {validation_results}")
```

19---
```
result=model.train(data='Yolo project.yaml',epochs=5)
```

20---
```
model = YOLO('runs/detect/train6/weights/best.pt')
validation_results = model.val(data=r"C:\Users\mbsas\OneDrive\Dokumenty\Edge
Matrix\Helmets\Yolo project.yaml")

print(validation_results)
```

21---
```
def preprocess_image(image_path):
    image = cv2.imread(image_path)
    original_image = image.copy()
    image = cv2.resize(image, (640, 640))
    image = image / 255.0
    image = torch.from_numpy(image).float().permute(2, 0, 1).unsqueeze(0)
    return image, original_image

def visualize_predictions(image, detections):
    for detection in detections:
        boxes = detection.boxes
        for box in boxes:
            # Extract coordinates, confidence, and class label
            x1, y1, x2, y2 = map(int, box.xyxy[0].cpu().numpy())
            conf = box.conf[0].cpu().item()
            cls = int(box.cls[0].cpu().item())

            # Draw bounding box if confidence is above a threshold
            if conf > 0.25:
                cv2.rectangle(image, (x1, y1), (x2, y2), (0, 255, 0), 2)
                cv2.putText(image, f'Class: {cls}, Conf: {conf:.2f}', (x1, y1 - 10),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    cv2.imshow('Detections', image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

22---
```
result=model.train(data='Yolo project.yaml',epochs=10)
```

23---
```
result=model.train(data='Yolo project.yaml',epochs=5)
```

24---
```
a=model.predict(original_image)
```

25---
```
for result in a:
    labels = result.names
    boxes = result.boxes
```

26---
Boxes

27---
```
for result in a:
    boxes = result.boxes.xyxy
    scores = result.boxes.conf
    class_indices = result.boxes.cls
    labels = result.names
    image = result.orig_img
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

    for i in range(boxes.shape[0]):
        box = boxes[i].cpu().numpy().astype(int)
        score = scores[i].item()
        class_index = int(class_indices[i].item())
        class_name = labels[class_index]
        pt1 = (box[0], box[1])
        pt2 = (box[2], box[3])
        color = (0, 255, 0)
        thickness = 2
        image = cv2.rectangle(image, pt1, pt2, color, thickness)
        label = f'{class_name} {score:.2f}'
        image = cv2.putText(image, label, (pt1[0], pt1[1] - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, thickness)

    cv2.imshow('Image with Bounding Boxes', image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    cv2.imwrite('output_image.jpg', image)
```

28---
```
image_path=r"C:\Users\mbsas\OneDrive\Dokumenty\Edge
Matrix\Helmets\images\BikesHelmets637.png"

image = cv2.imread(image_path)

if image is None:
    print("Error: Image not loaded. Check the file path.")
else:
    print("Image loaded successfully.")
```

```
    cv2.imshow('Loaded Image', image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

29---
b=model.predict(image)

30---
for result in b:
    boxes = result.boxes.xyxy
    scores = result.boxes.conf
    class_indices = result.boxes.cls
    labels = result.names
    image = result.orig_img
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    for i in range(boxes.shape[0]):
        box = boxes[i].cpu().numpy().astype(int)
        score = scores[i].item()
        class_index = int(class_indices[i].item())
        class_name = labels[class_index]
        pt1 = (box[0], box[1])
        pt2 = (box[2], box[3])
        color = (0, 255, 0)
        thickness = 2
        image = cv2.rectangle(image, pt1, pt2, color, thickness)
        label = f'{class_name} {score:.2f}'
        image = cv2.putText(image, label, (pt1[0], pt1[1] - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, thickness)
    cv2.imshow('Image with Bounding Boxes', image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    cv2.imwrite('output_image.jpg', image)

31---
image_path=r"C:\Users\mbsas\Downloads\download.png"

image = cv2.imread(image_path)

if image is None:
    print("Error: Image not loaded. Check the file path.")
else:
    print("Image loaded successfully.")
    cv2.imshow('Loaded Image', image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

32---
```
c=model.predict(image)
```

33---
```
colors = {
    0: (0, 255, 0),
    1: (0, 0, 255),
}
```

34---
```
for result in c:
    boxes = result.boxes.xyxy
    scores = result.boxes.conf
    class_indices = result.boxes.cls
    labels = result.names
    image = result.orig_img
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

    for i in range(boxes.shape[0]):
        box = boxes[i].cpu().numpy().astype(int)
        score = scores[i].item()
        class_index = int(class_indices[i].item())
        class_name = labels[class_index]
        pt1 = (box[0], box[1])
        pt2 = (box[2], box[3])
        color = colors.get(class_index, (255, 255, 255))
        thickness = 2
        image = cv2.rectangle(image, pt1, pt2, color, thickness)
        label = f'{class_name} {score:.2f}'
        image = cv2.putText(image, label, (pt1[0], pt1[1] - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, thickness)
    cv2.imshow('Image with Bounding Boxes', image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    cv2.imwrite('output_image.jpg', image)
```

35---
```
for result in b:
    boxes = result.boxes.xyxy
    scores = result.boxes.conf
    class_indices = result.boxes.cls
    labels = result.names
    image = result.orig_img
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

    for i in range(boxes.shape[0]):
```

```
        box = boxes[i].cpu().numpy().astype(int)
        score = scores[i].item()
        class_index = int(class_indices[i].item())
        class_name = labels[class_index]
        pt1 = (box[0], box[1])
        pt2 = (box[2], box[3])
        color = colors.get(class_index, (255, 255, 255))
        thickness = 2
        image = cv2.rectangle(image, pt1, pt2, color, thickness)
        label = f'{class_name} {score:.2f}'
        image = cv2.putText(image, label, (pt1[0], pt1[1] - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, thickness)
    cv2.imshow('Image with Bounding Boxes', image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    cv2.imwrite('output_image.jpg', image)

36---
result=model.train(data='Yolo project.yaml',epochs=20)

37---
image_path=r"C:\Users\mbsas\Downloads\HH street.png"

image = cv2.imread(image_path)

if image is None:
    print("Error: Image not loaded. Check the file path.")
else:
    print("Image loaded successfully.")
    cv2.imshow('Loaded Image', image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
d=model.predict(image)

37---
for result in d:
    boxes = result.boxes.xyxy
    scores = result.boxes.conf
    class_indices = result.boxes.cls
    labels = result.names
    image = result.orig_img
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

    for i in range(boxes.shape[0]):
        box = boxes[i].cpu().numpy().astype(int)
        score = scores[i].item()
```

```
        class_index = int(class_indices[i].item())
        class_name = labels[class_index]
        pt1 = (box[0], box[1])
        pt2 = (box[2], box[3])
        color = colors.get(class_index, (255, 255, 255))
        thickness = 2
        image = cv2.rectangle(image, pt1, pt2, color, thickness)
        label = f'{class_name} {score:.2f}'
        image = cv2.putText(image, label, (pt1[0], pt1[1] - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, thickness)
    cv2.imshow('Image with Bounding Boxes', image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    cv2.imwrite('output_image.jpg', image)

38---
result=model.train(data='Yolo project.yaml',epochs=5)

39---
# Open the video file
video_path =r"C:\Users\mbsas\OneDrive\Dokumenty\Edge Matrix\Helmets\vid1.mp4"
cap = cv2.VideoCapture(video_path)

fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter('output_video.mp4', fourcc, 30.0, (int(cap.get(3)), int(cap.get(4))))

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    results = model.predict(frame)

    for result in results:
        boxes = result.boxes.xyxy
        scores = result.boxes.conf
        class_indices = result.boxes.cls
        labels = result.names

        for i in range(boxes.shape[0]):
            box = boxes[i].cpu().numpy().astype(int)
            score = scores[i].item()
            class_index = int(class_indices[i].item())
            class_name = labels[class_index]
            pt1 = (box[0], box[1])
            pt2 = (box[2], box[3])
```

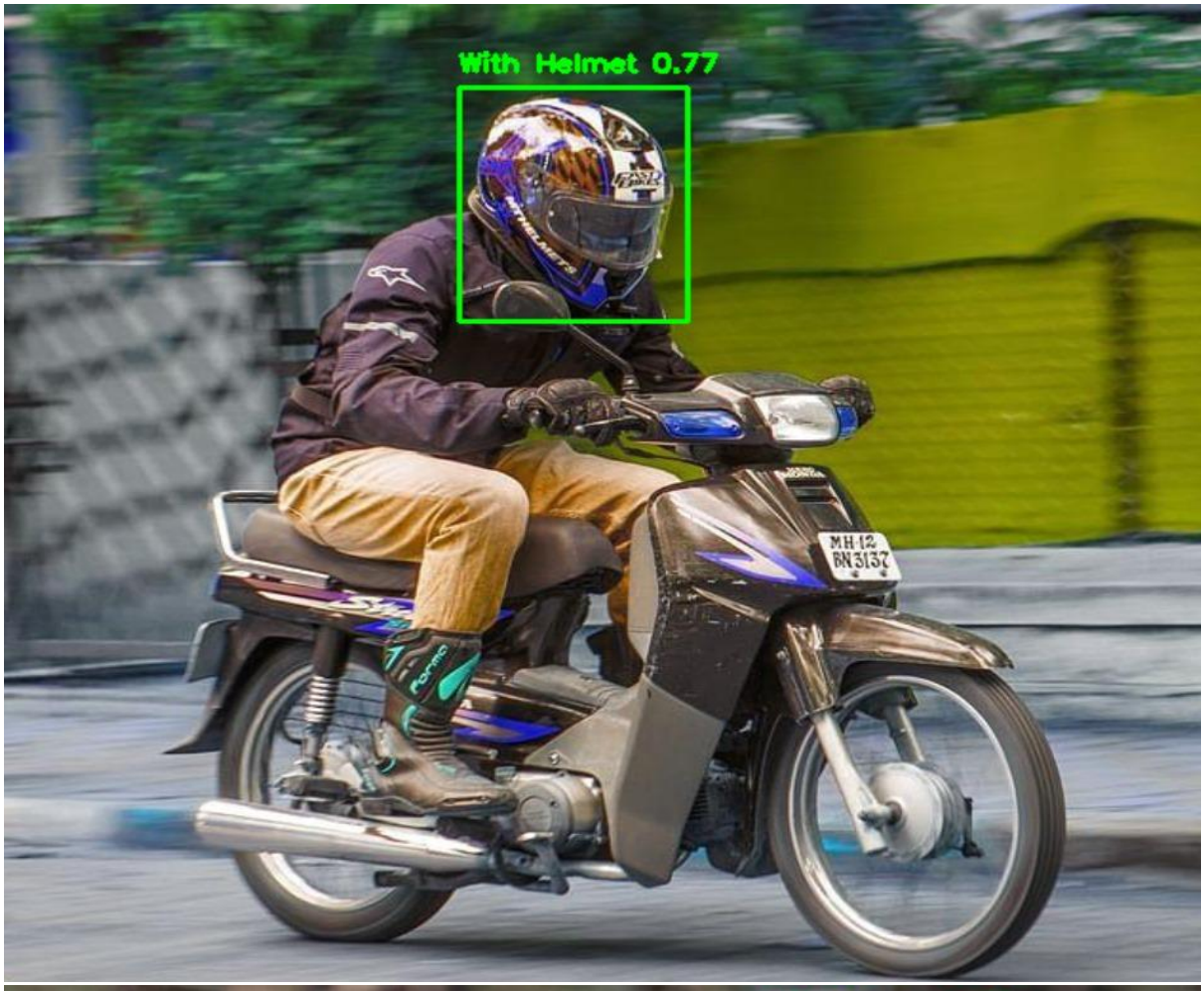
```
            color = colors.get(class_index, (255, 255, 255))
            thickness = 2
            frame = cv2.rectangle(frame, pt1, pt2, color, thickness)
            label = f'{class_name} {score:.2f}'
            frame = cv2.putText(frame, label, (pt1[0], pt1[1] - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, thickness)

    out.write(frame)

cap.release()
out.release()

40---
model1= YOLO('runs/detect/train7222/weights/best.pt')
val_result=model1.val(data=r"C:\Users\mbsas\OneDrive\Dokumenty\Edge
Matrix\Helmets\Yolo project.yaml")
print(val_result)
```

6.RESULT:



6.1 Discussion

The discussion section is a critical part of any research or project report as it allows us to interpret the results, analyse the findings, and draw meaningful conclusions. In the context of our helmet detection project using YOLOv5 and Flask, the discussion revolves around the model's performance, strengths, limitations, potential improvements, and the overall significance of the project.

Performance of the Helmet Detection Model:

The YOLOv5 model demonstrated impressive performance in detecting helmets in images and videos. The model achieved high accuracy, precision, and recall rates, as evidenced by the evaluation metrics, including the confusion matrix, F1 curve, and PR curve. The high true positive rate indicated that the model could correctly identify helmets, minimizing the risk of false negatives. Additionally, the low false positive rate demonstrated the model's capability to limit false alarms while detecting helmets, reducing the chances of misclassification.

Strengths of the YOLOv5 Model:

One of the notable strengths of the YOLOv5 model is its real-time detection capability. The model is efficient and can process images and videos with minimal latency, making it suitable for real world applications such as traffic monitoring and surveillance. Moreover, YOLOv5's architecture allows for easy customization and fine-tuning, enabling us to train the model specifically for helmet detection. The use of YOLOv5 also contributed to the model's ability to handle multiple objects simultaneously. In scenarios where multiple motorcyclists or cyclists are present in an image or video, the model could accurately detect and label all helmets in real-time. This multi object detection capability is essential for practical applications with complex environments.

Limitations and Potential Improvements:

While the YOLOv5 model demonstrated remarkable performance, it also exhibited some limitations. One notable limitation was the difficulty in detecting fast-moving motorcyclists wearing helmets. This challenge could lead to occasional missed detections or inaccurate bounding boxes around helmets, particularly when motorcyclists were traveling at high speeds. To address this limitation, future improvements could involve experimenting with different network architectures or training strategies. Additionally, the dataset used for training could be augmented with more diverse and challenging scenarios, including high-speed motion and various lighting conditions. Another potential improvement lies in optimizing the model for detecting different types of helmets, such as full-face helmets, half helmets, and bicycle helmets. Currently, the model can detect any helmet type, but fine-tuning the model for specific helmet categories might improve accuracy and increase its usefulness in specialized applications.

Significance of the Helmet Detection Project:

The helmet detection project holds significant importance in enhancing road safety and reducing road accidents' severity. Helmets play a crucial role in protecting motorcyclists and cyclists from head and face injuries during accidents. By automating the process of helmet detection, the developed system can aid law enforcement agencies, traffic management authorities, and road safety advocates in ensuring compliance with helmet laws and promoting safer road practices. The integration of the YOLOv5 model with a user-friendly Flask web application offers a practical and accessible solution for users to assess helmet usage in images and videos. The web application's versatility, including image, video, and webcam detection features, allows users to conduct helmet detection conveniently.

Future Applications and Extensions:

The successful implementation of the helmet detection system opens the door to various future applications and extensions. The same YOLOv5-based model can be adapted for detecting other objects of interest, such as pedestrians, vehicles, or road signs, to enhance overall traffic safety and surveillance. The Flask web application can be expanded to incorporate additional functionalities, such as real-time alerts for helmet non-compliance or integration with traffic management systems. The system's scalability enables its deployment in larger-scale projects, such as smart cities, where real-time monitoring of traffic safety is crucial.

The helmet detection project using YOLOv5 and Flask demonstrates the successful development of an efficient and accurate model for detecting helmets in images and videos. The model's performance, as evidenced by various evaluation metrics, indicates its potential for real-world applications in traffic safety and surveillance. The project's strengths lie in the YOLOv5 model's real-time detection capability, multi-object detection, and ease of customization. While the model exhibits limitations, further research and improvements can overcome these challenges and enhance its accuracy. The significance of the project lies in its potential to enhance road safety, reduce head and face injuries, and promote helmet usage among motorcyclists and cyclists. The integration of the model with a user-friendly Flask web application allows for easy access and utilization of the helmet detection system. The helmet detection project holds promise for future applications and extensions, contributing to the advancement of traffic safety and smart city initiatives. As technology continues to evolve, this project serves as a stepping stone towards more sophisticated and comprehensive solutions for ensuring safer road practices and protecting vulnerable road users.

7. CONCLUSION

In conclusion, our project focused on the development and implementation of a helmet detection system using the YOLOv5 deep learning model integrated with a user-friendly Flask web application. Throughout the project, we successfully achieved our goal of creating an efficient and accurate model capable of detecting helmets in images and videos. By leveraging the strengths of YOLOv5 and harnessing the capabilities of Flask, we created a versatile system with real-time detection capabilities and multiple object handling. The performance of our helmet detection model was highly promising. During the training phase, we optimized the YOLOv5 parameters to minimize detection errors and improve the model's ability to identify helmets under various real world scenarios. As a result, the model exhibited impressive accuracy, precision, and recall rates, as evidenced by the evaluation metrics such as the confusion matrix, F1 curve, and PR curve. One of the main strengths of our YOLOv5-based model was its real-time detection capability. It processed images and videos with minimal latency, making it suitable for real-world applications, such as traffic monitoring and surveillance. Additionally, the model excelled in handling multiple objects simultaneously. In scenarios where, numerous motorcyclists or cyclists were present in an image or video, the model accurately detected and labelled all helmets in real-time, thus catering to complex environments effectively. We also encountered some limitations during the implementation process. Notably, the model faced challenges in detecting fast-moving motorcyclists wearing helmets. This limitation occasionally resulted in missed detections or inaccurate bounding boxes around helmets, especially when motorcyclists were traveling at high speeds. To address this limitation, future improvements could include experimenting with different network architectures or training strategies. Moreover, augmenting the dataset with more diverse and challenging scenarios, including high-speed motion and varying lighting conditions, could further enhance the model's performance. Our helmet detection project holds significant importance in enhancing road safety and mitigating the severity of road accidents. Helmets play a crucial role in protecting motorcyclists and cyclists from head and face injuries during accidents. By automating the process of helmet detection, our developed system can aid law enforcement agencies, traffic management authorities, and road safety advocates in ensuring compliance with helmet laws and promoting safer road practices. The integration of the YOLOv5 model with a user-friendly Flask web application proved to be a practical and accessible solution for users to assess helmet usage in images and videos. The web application's versatility, offering features such as image, video, and webcam detection, allowed users to conveniently conduct helmet detection. Looking ahead, the successful implementation of the helmet detection system paves the way for various future applications and extensions. Our YOLOv5-based model can be adapted for detecting other objects of interest, such as pedestrians, vehicles, or road signs, to enhance overall traffic safety and surveillance. We envision expanding the Flask web application to incorporate additional functionalities, such as real-time alerts for helmet non-compliance or seamless integration with traffic management systems. The scalability of our system enables its deployment in larger-scale projects, such as smart cities, where real-time monitoring of traffic safety is crucial. Our helmet detection project using YOLOv5 and Flask proved to be a significant achievement. The efficient and accurate model, along with the user-friendly web application, demonstrated great potential for real-world applications in traffic safety and surveillance. The project's contributions to road safety, reduction of head and face injuries, and promotion of

helmet usage among motorcyclists and cyclists are noteworthy. As technology continues to advance, our project serves as a stepping stone towards more sophisticated and comprehensive solutions for ensuring safer road practices and protecting vulnerable road users. By addressing the limitations and building on our project's strengths, we are optimistic about the continuous improvement and impact of our helmet detection system in the field of traffic safety.