Pedestrian Detection in Bad Weather Conditions

A

Project Seminar Report

Submitted in partial fulfilment of the

Requirements for the award of the Degree of

BACHELOR OF ENGINEERING

IN

INFORMATION TECHNOLOGY

By

B. Hemanth	**1602-20-737-073**
C. Narendra Sai	**1602-20-737-084**
N. Naveen Reddy	**1602-20-737-086**

Under the guidance of

Dr. M. NEELAKANTAPPA

Associate Professor

Department of Information Technology

Vasavi College of Engineering (Autonomous)

ACCREDITED BY NAAC WITH 'A++' GRADE

(Affiliated to Osmania University)

Ibrahimbagh, Hyderabad-31

2024

Vasavi College of Engineering (Autonomous)

ACCREDITED BY NAAC WITH 'A++' GRADE

(Affiliated to Osmania University)

Hyderabad-500 031

Department of Information Technology

DECLARATION BY THE CANDIDATE

We, **B. Hemanth , C. Narendra Sai** and **N. Naveen Reddy** bearing hall ticket number, **1602-20-737-073 ,1602-20-737-084 and 1602-20-737-086** hereby declare that the project report entitled **Pedestrian Detection in Bad Weather Conditions** under the guidance of **Dr M. Neelakantappa, Associate Professor**, Department of Information Technology, Vasavi College of Engineering, Hyderabad, is submitted in partial fulfilment of the requirement for the award of the degree of **Bachelor of Engineering** in **Information Technology**

This is a record of bonafide work carried out by us and the results embodied in this project report have not been submitted to any other university or institute for the award of any other degree or diploma.

B. Hemanth
1602-20-737-073

C. Narendra Sai
1602-20-737-084

N. Naveen Reddy
1602-20-737-086

Vasavi College of Engineering (Autonomous)

ACCREDITED BY NAAC WITH 'A++' GRADE

(Affiliated to Osmania University)

Hyderabad-500 031

Department of Information Technology

BONAFIDE CERTIFICATE

This is to certify that the project entitled **Pedestrian Detection in Bad Weather Conditions** being submitted by **B. Hemanth , C. Narendra Sai** and **N. Naveen Reddy** bearing **1602-20-737-073**, **1602-20-737-084** and **1602-20-737-086** in partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering in Information Technology is a record of bonafide work carried out by them under my guidance.

Dr M. Neelakantappa	**Dr. K. Ram Mohan Rao,**
Associate Professor	**Professor,**
Internal Guide	**HOD, IT**

External Examiner

ACKNOWLEDGEMENT

The satisfaction that accompanies that the successful completion of the project seminar would not have been possible without the kind support and help of many individuals. We would like to extend our sincere thanks to all of them.

It is with immense pleasure that we would like to take the opportunity to express our humble gratitude to **Dr. M. Neelakantappa, Associate Professor, Information Technology** under whom we executed this project. We are also grateful to **Dr. S. Sree Lakshmi**, **Assistant Professor (Sr.Scale)**, **Information Technology** for her guidance. Their constant guidance and willingness to share their vast knowledge made us understand this project and its manifestations in great depths and helped us to complete the assigned tasks.

We are very much thankful to **Dr. K. Ram Mohan Rao, Professor, Information Technology,** for his kind support and for providing necessary facilities to carry out the work.

We wish to convey our special thanks to **S.V. Ramana, Principal** of **Vasavi College of Engineering** for giving the required information in doing my project work. Not to forget, we thank all other faculty and non-teaching staff, and my friends who had directly or indirectly helped and supported me in completing my project in time.

We also express our sincere thanks to the Management for providing excellent facilities. Finally, we wish to convey our gratitude to our family who fostered all the requirements and facilities that we need.

ABSTRACT

Pedestrian detection is vital for ensuring safety in autonomous driving, surveillance, and urban planning, enabling systems to identify and track individuals in real-time. It plays a crucial role in preventing accidents, optimizing traffic flow, and enhancing security in public spaces. Additionally, pedestrian detection facilitates human-computer interaction and accessibility for individuals with disabilities. The method proposed in the paper, improves the accuracies of detection, and identifies the accurate pedestrian positions. This work proposes to mitigate the challenges posed by poor visibility due to inclement weather, thereby enhancing pedestrian safety and reducing the risk of accidents in such conditions. This project presents a pedestrian detection system based on ResNet and YOLOv8 architectures, leveraging their capabilities in feature extraction and real-time object detection. Our results demonstrate the effectiveness of comparing ResNet and YOLOv8 for pedestrian detection tasks, with implications for applications in autonomous driving, surveillance, and urban planning.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

S.No.	Abbreviation	Full Form
1.	ResNet	Residual Network
2.	IR	Infrared
3.	DL	Deep Learning
4.	XGBOOST	Extreme Gradient Boosting
5.	HOG	Histogram of Oriented Gradients
6.	API	Application Programming Interface
7.	TPU	Tensor Processing Unit
8.	GPU	Graphic Processing Unit
9.	HSV	Hue, Saturation, and value
10.	CNN	Convolutional Neural Network

I. INTRODUCTION

1.1 Problem Statement

Pedestrian detection is a critical task in various applications such as autonomous driving, surveillance, and urban planning. However, it poses several challenges, including occlusion, varying scales, complex backgrounds, and changes in lighting conditions. Existing pedestrian detection systems often struggle to achieve high accuracy and robustness in real-world scenarios due to these challenges. Pedestrian detection is widely used in intelligent monitoring, automotive assisted driving, security and intelligent transportation areas. It has been concerned for several years in many computer vision applications. Specially machine learning become more and more popular for pedestrian detection.

In general, it is realized by building pedestrian feature extraction model and building feature classifier [1]. Traditional target detection methods are based on the classical framework of sliding window or manually selected, including VJ [2],HOG [3],SIFT [4],LBP [5],and Haar [6], which are man-made models and then classified by sliding window and linear support vector machine (SVM) [6].

Pedestrians are a major group at risk of death, injury and disability on the road. According to a report on the state of global road safety released by the World Health Organization, nearly 1.24 million people died in 2013 because of road traffic accidents, 22% of which were pedestrians [7].

In particular, occlusion remains a significant obstacle, where pedestrians may be partially or fully obstructed by other objects or by other pedestrians. Traditional methods often fail to correctly identify pedestrians in these situations, leading to missed detections or false alarms. Additionally, variations in scale, such as pedestrians appearing at different sizes due to distance or perspective, further complicate the detection process. Furthermore, changes in lighting conditions, such as shadows, glare, or low light, can significantly affect the visibility of pedestrians, making it challenging for detection algorithms to accurately identify them.
In such conditions, existing pedestrian detection systems often struggle to distinguish pedestrians from the background clutter or misclassify objects due to poor visibility.

The dynamic nature of weather conditions further complicates the task, requiring robust algorithms capable of adapting to varying degrees of visibility and environmental factors.

Addressing the problem of pedestrian detection in bad weather conditions requires the development of innovative algorithms and techniques that can effectively handle degraded image quality and environmental challenges. This includes exploring methods for enhancing image visibility, such as adaptive image enhancement or fusion techniques, as well as leveraging advanced deep learning architectures that are robust to noise and partial occlusions.

1.2 Proposed Work

This project proposes to employ two advanced deep learning models, ResNet50 and YOLOv8, for pedestrian detection using the INRIA and HAZY WEATHER datasets.

The INRIA dataset consists of images captured from urban scenes containing pedestrians in various poses and orientations. It is annotated with bounding boxes around the pedestrians, providing ground truth labels for training and evaluation purposes. The dataset covers diverse scenarios, including crowded streets, intersections, and pedestrian crossings, making it suitable for pedestrian detection tasks in normal weather conditions.

The HAZY WEATHER dataset simulates adverse weather conditions such as fog, rain, and snow, which pose challenges for pedestrian detection algorithms. It contains images captured under these conditions, with varying levels of visibility and environmental obstructions. Additionally, the dataset includes annotations for pedestrian locations, allowing for the evaluation of detection performance in adverse weather scenarios. Both datasets provide valuable resources for training and testing pedestrian detection models under different environmental conditions

Firstly, the models will be trained separately on the INRIA dataset to detect pedestrians in normal weather conditions. Transfer learning techniques will be applied to fine-tune the pre-trained models for this task.

Subsequently, the models will be trained on the HAZY WEATHER dataset to detect pedestrians in adverse weather conditions, such as fog and rain.

Following training, the performance of both models will be evaluated on separate test sets from the INRIA and HAZY WEATHER datasets. Standard metrics like accuracy, precision, recall, and F1-score will be used to assess the models' performance in different environmental conditions.

The expected outcomes include high accuracy rates for pedestrian detection in normal weather conditions using both ResNet50 and YOLOv8 models. In adverse weather conditions, we anticipate a decrease in detection performance, albeit with YOLOv8 likely outperforming ResNet50 due to its real-time detection capabilities.

1.3 Scope and Objectives of the proposed work

Scope:

The scope of the proposed project encompasses implementing ResNet50 and YOLOv8 models for pedestrian detection using the INRIA and HAZY WEATHER datasets. This involves preprocessing and splitting the datasets, training the models separately to detect pedestrians in normal and adverse weather conditions, and evaluating their performance using standard metrics. The project aims to compare the effectiveness of the two models in detecting pedestrians under different environmental conditions, with a particular focus on their accuracy and efficiency. By conducting this study, we anticipate gaining insights into the capabilities of ResNet50 and YOLOv8 for pedestrian detection, contributing to advancements in autonomous vehicles and surveillance systems' safety features.

Objectives:

1. Implement ResNet50 and YOLOv8 models for pedestrian detection.
2. Preprocess the INRIA and HAZY WEATHER datasets for training and testing.
3. Train the models separately on the INRIA dataset for normal weather conditions and the HAZY WEATHER dataset for adverse weather conditions using transfer learning.
4. Evaluate the performance of both models on separate test sets from the INRIA and HAZY WEATHER datasets using accuracy, precision, recall, and F1-score.
5. Compare the effectiveness of ResNet50 and YOLOv8 in detecting pedestrians under different environmental conditions.
6. Conclude the study by providing insights into the capabilities of the models for pedestrian detection and their potential applications.

1.4 Organization of the report

A. List of Figures: This section enumerates all the figures present in the document, aiding readers in locating visual aids like graphs, charts, and diagrams.

B. List of Tables: Here, readers can find a compilation of all the tables included in the document, providing easy access to structured data and information.

C. List of Abbreviations: This section presents a catalog of abbreviations used throughout the document along with their corresponding full forms, facilitating comprehension of specialized terminology and acronyms.

D. Introduction

- Problem Statement – Overview: Provides a succinct description of the problem addressed in the document.
- Motivation: Outlines the rationale and objectives driving the proposed work.
- Scope & Objectives of the Proposed Work: Defines the scope and goals of the project.
- Organization of the Report: Gives a brief overview of the structure and content of the document.

E. Literature Survey:

This section reviews existing literature and research relevant to the document's topic, offering background information and context for the proposed work.

F. Proposed System

- System Specifications
- Software Requirements: Lists the required software tools and technologies.
- Hardware Requirements: Specifies the necessary hardware components.
- Methodology
- Architecture Diagram: Presents a visual representation of the system architecture.
- Algorithm: Provides detailed steps of the project.

G. Experimental Setup & Results

- Datasets: Describes the datasets used for experimentation.
- Results & Test Analysis: Presents experimental results and provides analysis, including figures and tables with explanations.

H. Conclusion and Future Scope:
Summarizes the findings of the document and discusses potential future directions or extensions of the proposed work.

I. References: Lists all the sources cited in the document.

J. Appendix: Contains supplementary material such as additional figures, code snippets, or detailed experimental results

II. LITERATURE SURVEY

Pedestrian safety is a critical concern in modern transportation systems, with pedestrian-related accidents accounting for a significant portion of global road fatalities. Optimization techniques have been proposed to enhance the performance of pedestrian detection algorithms, with XGBoost[1] being a notable example. XGBoost, derived from Gradient Boosting Decision Trees (GBDT), offers fast convergence and generalization abilities, making it suitable for various machine learning tasks .Several optimization methods, such as scenario-based optimization algorithms and Bayesian optimization, have been developed to further improve the accuracy and stability of XGBoost models [1]. Various approaches have been proposed to improve pedestrian detection, a crucial component of intelligent transportation systems and automotive safety. Machine learning algorithms, particularly boosting techniques like XGBoost, have gained prominence in pedestrian detection due to their ability to handle complex data and improve classification accuracy[1].

Deep learning techniques, especially Convolutional Neural Networks (CNNs), have emerged as powerful tools for pedestrian detection, as they can automatically learn discriminative features from raw input data. CNN-based approaches have shown promise in improving detection accuracy, although they require substantial computational resources. In addition to feature extraction, the choice of machine learning algorithm is crucial for pedestrian classification. Support Vector Machines (SVMs), neural networks, random forests, and Adaboost are commonly used classifiers in pedestrian detection tasks. SVMs, in particular, are praised for their effectiveness in linear classification, although they may struggle with nonlinear problems [2].

Feature extraction plays a crucial role in pedestrian detection, with Histogram of Oriented Gradients (HOG) [3] being one of the most commonly used methods. However, HOG features suffer from high dimensionality, which can affect both training and detection speed. To address this limitation, researchers have explored alternative features such as Local Binary Patterns (LBP) and integral channel features (ACF) to enhance computational efficiency without compromising accuracy [3].

According to the World Health Organization, nearly 1.24 million people died in road traffic accidents in 2013, with pedestrians comprising 22% of the total fatalities [7]. In China, pedestrian accidents constitute approximately 20% of the total number of traffic accidents, with pedestrian casualties accounting for about 30% of the total casualties [7].

Recognizing the importance of pedestrian safety, regulatory bodies such as the China New Car Assessment Programme (C-NCAP) have introduced pedestrian test evaluation scenarios to promote vehicle safety standards. The latest editions of C-NCAP[8] have expanded and tightened these test scenarios to address the growing concern of pedestrian-related accidents.

Pedestrian detection algorithms are evaluated using metrics such as the Area Under the ROC Curve (AUC) to assess their performance. Validation techniques, including track testing and comparison with existing algorithms, are crucial for ensuring the effectiveness and practicality of new methods in real-world scenarios [8]. The advancements in pedestrian detection algorithms have significant implications for intelligent monitoring, automotive assisted driving, security, and intelligent transportation systems, contributing to safer road environments and reducing the number of pedestrian-related accidents.

Adverse weather conditions significantly impact visibility and increase accident risks. Studies show up to a 13% increase in accident rates in adverse weather. To address this, new safety measures, including advanced driver assistance systems (ADAS)[8], have been introduced. These systems, with features like intelligent speed assistance and advanced emergency braking, aim to prevent accidents and reduce fatalities. Improving pedestrian detection capabilities is crucial for these systems to protect vulnerable road users and enhance overall road safety.

While deep learning-based detectors have shown promise in ideal conditions, their performance in adverse weather remains unclear. Far-infrared (FIR) pedestrian detection systems have been developed to address this, utilizing thermal data, especially at nighttime. However, datasets like ZUT, providing annotated images captured in severe weather across

European countries, are crucial for evaluating and improving pedestrian detection algorithms in challenging scenarios.

The focus on road safety has led to the introduction of stricter safety measures for vehicles. Enhancing pedestrian detection capabilities contributes to this goal. The advancements presented, including the ZUT dataset and YOLOv3[8] modifications, demonstrate progress toward improving pedestrian safety under adverse weather conditions. These developments are significant steps forward in achieving safer road environments.

To address the challenge of low visibility, thermal images are utilized instead of conventional camera images for both training and testing the detection models. The thermal images are sourced from datasets such as ZUT and OTCBVS repository, which provide annotated images suitable for object detection tasks in adverse weather conditions.

Two popular object detection algorithms, YOLOv3 (You Only Look Once) and Faster RCNN (Faster Region-based Convolutional Neural Network), are employed in this work. YOLOv3 is known for its real-time detection capabilities and efficiency, while Faster RCNN offers accurate region-based object detection. By leveraging both algorithms, an ensemble approach is adopted to enhance detection performance under extreme weather conditions.

Through this work, the goal is to develop robust pedestrian detection systems capable of operating effectively in adverse weather conditions, thereby mitigating the risk of road accidents. By leveraging thermal imaging and state-of-the-art object detection algorithms, this research contributes to enhancing road safety and reducing the incidence of accidents caused by poor visibility conditions influenced by weather.

III. PROPOSED WORK

3.1 System Requirements

3.1.1 Software Requirements

Operating System

Windows 10 or above

Programming Language

Python 3.8.10 or above

Libraries

Image Processing –

- OpenCV 4.9.0.80
- Numpy 1.14
- Matplotlib

Deep Learning –

- Torch framework 2.1.0
- TensorFlow 2.0

3.1.2 Hardware Requirements

Processor

A CPU with 4 or more cores or GPU which speeds up model training and inference.

Random Access Memory

8GB or more RAM necessary to handle the data and model during training.

Storage

Fast Storage of at least 20 GB, such as Solid-State Drives (SSD) for handling datasets.

3.2 Methodology:

Addressing the imperative for swift and accurate pedestrian detection in haze-laden environments underscores our project's significance. Existing solutions face hurdles in identifying pedestrians amidst low visibility and haze. These challenges, compounded by limited spatial resolution and revisit frequencies, hinder timely and precise detection efforts. Recognizing the urgency, collaboration and innovation drive our endeavor to integrate advanced sensor technology and data processing techniques. By leveraging deep learning models such as ResNet-50 and YOLO, we aim to enhance pedestrian detection efficacy in adverse weather conditions. Our project seeks to revolutionize pedestrian safety by providing real-time detection capabilities, enabling proactive measures to mitigate the risks associated with haze, and ultimately ensuring safer pedestrian environments.

3.2.1 Block Diagram

The project involves annotating images to prepare them for model training. Annotation files include boundary box notations outlining the presence of persons in the images. Our dataset exclusively focuses on training for one class, namely, persons.

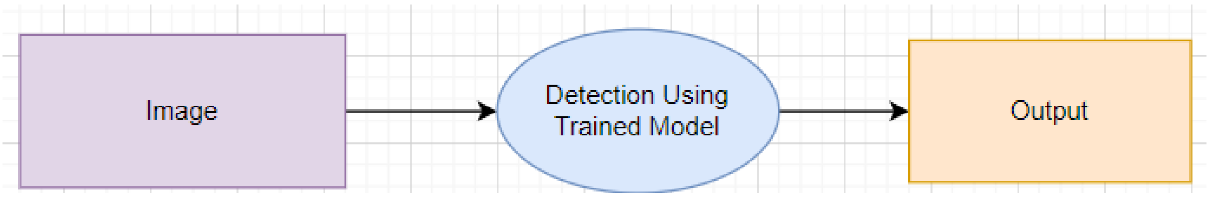

Fig.1. Block diagram of the system

Detection

This section describes various models which detects pedestrian in the image [1] and a technique employed to increase the accuracy of detection. The models used in are –

- Resnet 50
- YOLO v8
- Roboflow3.0 Object Detection

The models mentioned above are trained normally on Hazy Weather Dataset and INRIA Dataset . As mentioned above, there is only one class which is labeled as person and there is no change in the preprocessing procedure in our work.

3.2.2 Detection Algorithms

Object detection is a computer vision task that involves identifying and locating objects within images or videos. Unlike image classification, where the task is to classify the entire image into a single category, object detection algorithms not only classify the objects present but also provide precise bounding boxes around each object instance. This allows for the detection of multiple objects within the same image or video frame. The algorithms used for pedestrian detection are Resnet 50, YOLO v8, Roboflow3.0 Object detection.

Resnet50

ResNet-50 stands as a groundbreaking convolutional neural network (CNN) architecture, renowned for its exceptional performance in various computer vision tasks. Developed by Microsoft Research in 2015, ResNet-50 revolutionized the field with its ingenious design, effectively addressing the challenge of vanishing gradients encountered in deep neural networks.

At its core, ResNet-50 introduces the concept of residual connections, enabling the training of significantly deeper networks without suffering from degradation issues. These residual connections facilitate the flow of gradients through the network, mitigating the problem of information loss during training. The architecture comprises 50 layers, with each layer employing a combination of convolutional, pooling, and activation functions.

The ResNet-50 architecture consists of a series of convolutional blocks, each containing multiple convolutional layers, batch normalization, and rectified linear unit (ReLU) activation functions. These blocks are interspersed with identity blocks that preserve the input dimensions while increasing the network depth. Additionally, ResNet-50 incorporates global average pooling and a fully connected layer at the end, followed by a softmax activation function for classification tasks.

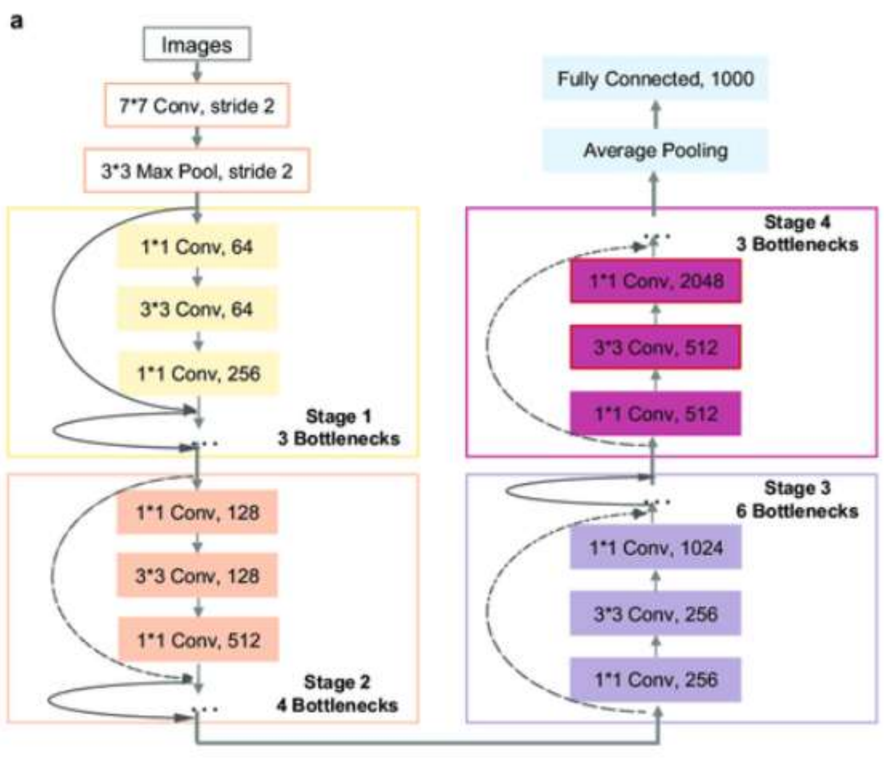

Fig 2. Architecture of RESNET50

This architecture diagram illustrates the intricate structure of ResNet-50, showcasing the interconnected layers and the flow of information through the network. Each block and layer plays a crucial role in extracting hierarchical features from input images, ultimately enabling ResNet-50 to achieve state-of-the-art performance in tasks such as image classification, object detection, and semantic segmentation.

Yolo v8:

YOLOv8, an evolution of the You Only Look Once (YOLO) object detection framework, represents a pinnacle in real-time object detection. Introduced by Joseph Redmon and Ali Farhadi in 2020, YOLOv8 pushes the boundaries of speed and accuracy, making significant advancements over its predecessors.

At its core, YOLOv8 adopts a single-stage architecture that simultaneously predicts bounding boxes and class probabilities directly from full images in a single forward pass. This approach allows YOLOv8 to achieve remarkable real-time performance while maintaining competitive accuracy. The architecture comprises a backbone network, typically based on

Darknet or a similar architecture, followed by multiple detection heads responsible for predicting object bounding boxes and class probabilities at different scales.

The YOLOv8 architecture diagram provides insight into its intricate design, illustrating the interconnected layers and the flow of information through the network. Each component, from the backbone network to the detection heads, contributes to YOLOv8's ability to accurately detect objects in diverse environments and under varying conditions.

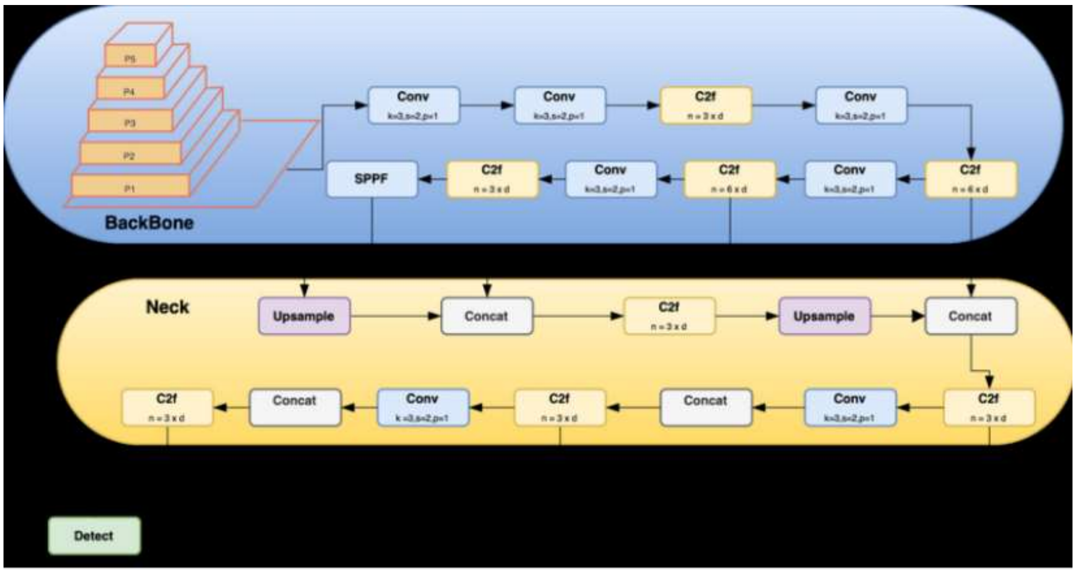

Fig 3. Architecture of YOLO v8

With its efficient architecture and streamlined training process, YOLOv8 has become a go-to choice for a wide range of real-time object detection applications, including autonomous driving, surveillance, and robotics.

Roboflow3.0 Object Detection Model:

Roboflow 3.0, the latest iteration of the Roboflow platform, represents a significant advancement in object detection capabilities. Engineered to streamline the entire object detection pipeline, from data preprocessing to model deployment, Roboflow 3.0 offers unparalleled efficiency and ease of use for developers and researchers alike.

At its core, Roboflow 3.0 leverages a sophisticated architecture that integrates seamlessly with popular deep learning frameworks such as TensorFlow, PyTorch, and Keras. The platform supports a wide range of object detection models, including YOLO, Faster R-CNN, and SSD, allowing users to choose the model architecture that best suits their specific needs and requirements.

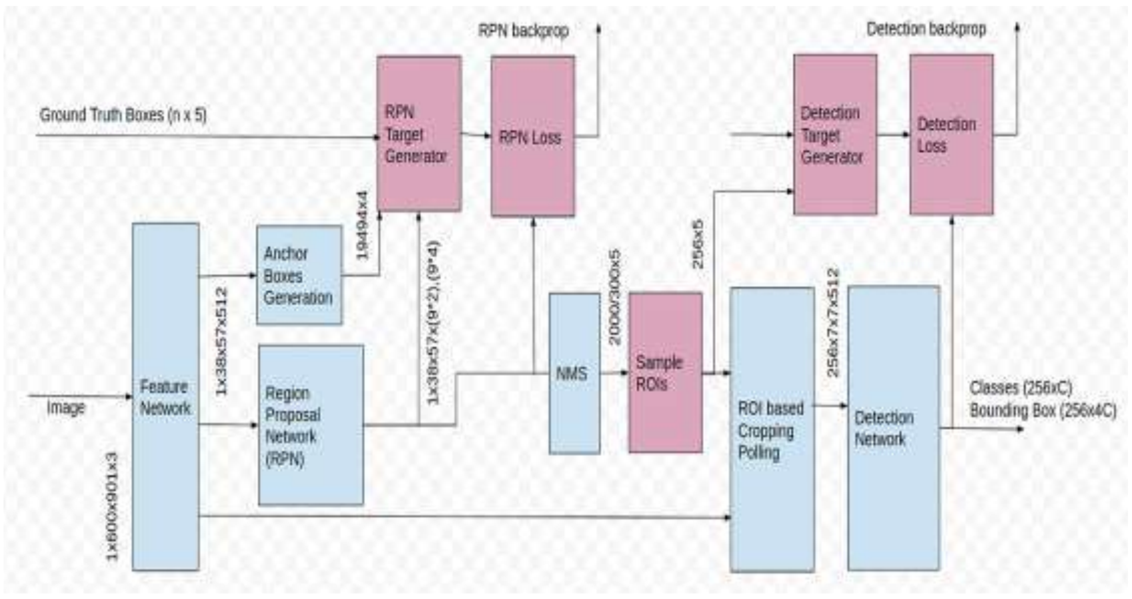

Fig 4. Architecture of Roboflow

The architecture diagram of Roboflow 3.0 provides a comprehensive overview of its components, illustrating the interconnected modules responsible for data ingestion, preprocessing, model training, evaluation, and deployment. Each module is designed to work harmoniously with the others, facilitating a seamless end-to-end workflow for object detection tasks.

With its intuitive interface, powerful features, and robust performance, Roboflow 3.0 empowers users to accelerate their object detection projects, enabling faster iteration and more accurate results. Whether deploying models for autonomous vehicles, surveillance systems, or industrial applications, Roboflow 3.0 provides the tools and infrastructure needed to succeed in today's rapidly evolving AI landscape.

IV. EXPERIMENTAL STUDY

4.1 Datasets

The dataset utilized in our project encompasses two distinct sets: one capturing images of pedestrians in hazy weather conditions, and the other featuring pedestrians in daylight conditions. These datasets provide a comprehensive view of pedestrian visibility across different environmental contexts. The hazy weather dataset comprises images captured under reduced visibility conditions, simulating scenarios where haze obscures the surrounding environment. In contrast, the daylight dataset consists of images captured under optimal lighting conditions, offering clear visibility of pedestrians without atmospheric interference.

Table1 . Description about the dataset

Dataset	**Description**
Hazy Weather	It Contains the images in a Hazy Weather
INRIA	It Contains the images in Day-light conditions.
CityPersons	It have the pedestrian images from the cities in Europe

Fig. 5 Hazy Weather Dataset

Fig.6 INRIA Dataset

The hazy weather dataset includes images captured using camera, ensuring quality data collection despite challenging environmental conditions. Conversely, the daylight dataset offers a baseline for pedestrian detection performance in ideal weather conditions. By leveraging these diverse datasets, our project aims to explore the efficiency of pedestrian detection algorithms across varying visibility conditions. This comprehensive approach enables us to develop robust models capable of accurately detecting pedestrians in both hazy and clear weather conditions, contributing to enhanced pedestrian safety and urban mobility.

4.2 Preprocessing

In our project, we conduct thorough preprocessing of the collected image datasets to ensure efficient analysis and model training. Initially, the images is converted to grayscale

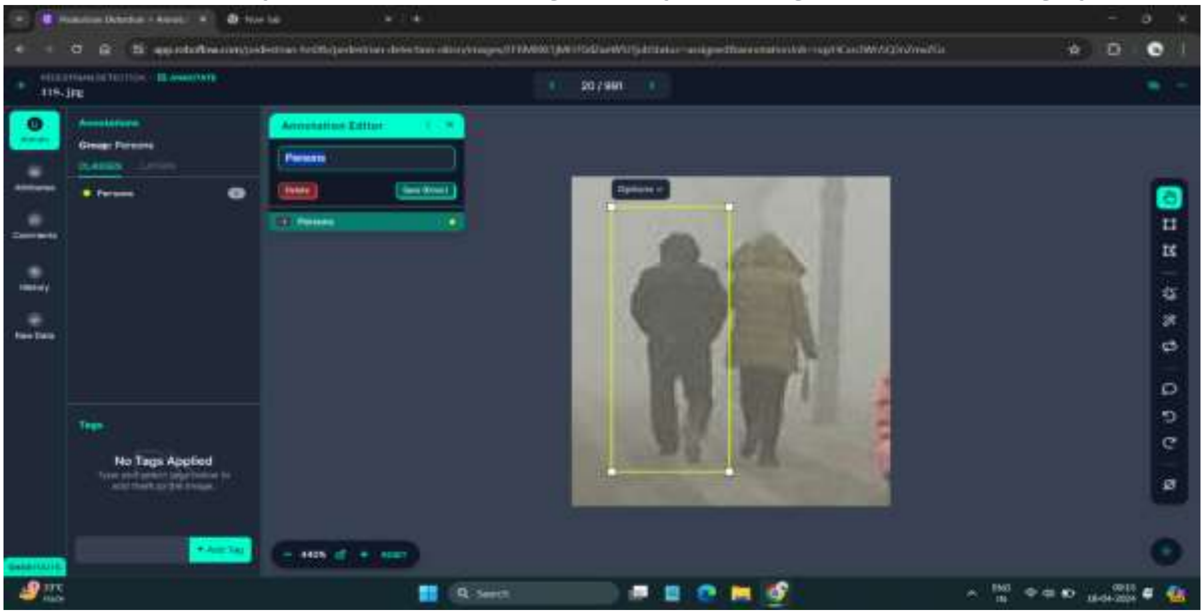

Fig 7. Image Annoation

using ROBOFLOW. Following this, the grayscale images are annotated and classified into various classes, with bounding boxes delineating pedestrians in both hazy weather and daylight conditions.

This preprocessing phase serves as the cornerstone for subsequent analysis and model development. By standardizing the structure and content of the dataset, we establish consistency and coherence, which are pivotal for robust training and evaluation of pedestrian detection algorithms.

The annotated dataset furnishes valuable insights into pedestrian visibility under varied weather conditions, facilitating the creation of models proficient in accurately detecting pedestrians in diverse environmental scenarios. This comprehensive approach enhances the dependability and efficacy of pedestrian detection systems, thereby contributing to the advancement of pedestrian safety and urban mobility.

The graphs depict the model's loss metrics during 300 epochs of training, revealing its iterative learning process. Each epoch involves a full iteration through the dataset, refining the model's predictions and minimizing loss. The visual representation illustrates the model's progress, highlighting trends toward optimal performance. Fluctuations in the loss curve may indicate challenges like overfitting or underfitting, prompting the need for analysis and optimization to improve effectiveness..

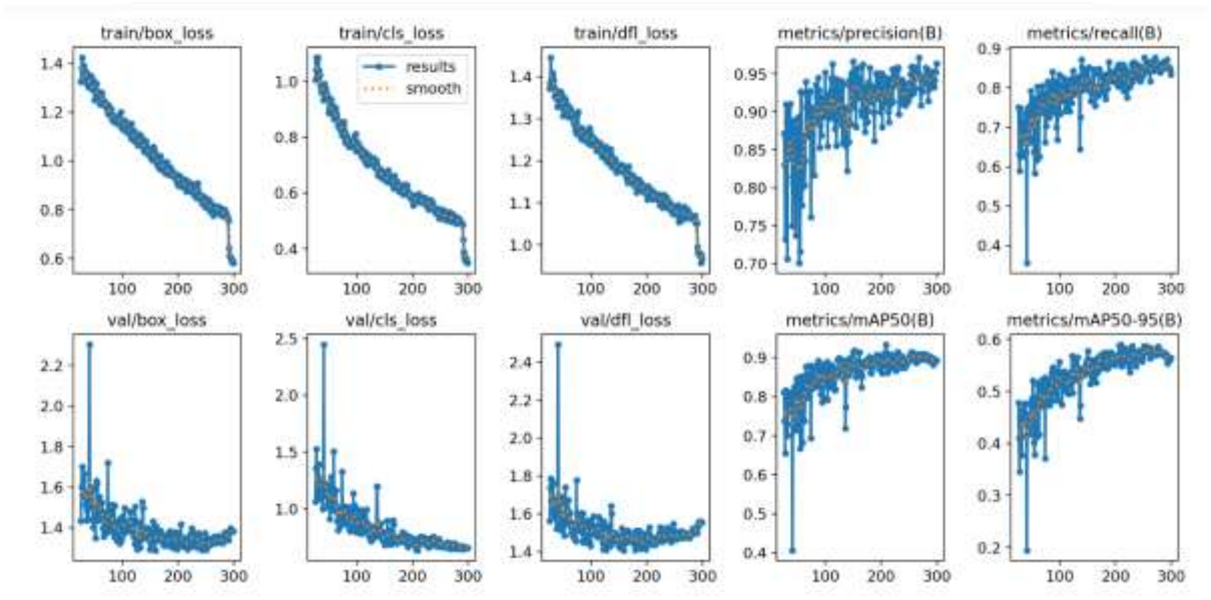

Fig.8 Training Graphs

4.3 Results

The results of the models are represented in the Table.3. The metrics used are accuracy, precision, f1-score, and recall.

a. Accuracy serves as a key metric within the field of machine learning, gauging the overall correctness of a model's predictions. It quantifies the proportion of accurately predicted cases relative to the total number of cases. Essentially, accuracy reflects the ratio of correct predictions—encompassing both true positives and true negatives—among all predictions generated by the model.

$$\text{Accuracy} = \frac{(TP + TN)}{(TP + FP + TN + FN)}$$

b. Precision, within the realm of machine learning, is a measure assessing the accuracy of positive predictions generated by a model. It quantifies the ratio of true positives to the sum of true positives and false positives. In essence, precision signifies the proportion of accurately predicted positive instances among all instances categorized as positive.

$$Precision = \frac{TP}{TP + FP}$$

c. Recall, also referred to as sensitivity or true positive rate, serves as a metric in machine learning to assess a model's capability in accurately recognizing all pertinent instances of a class. This measure is computed as the division of true positives by the sum of true positives and false negatives. Essentially, recall signifies the percentage of real positive instances accurately predicted by the model among all positive instances within the dataset.

$$Recall = \frac{TP}{TP + FN}$$

d. The F1-score, a statistical and machine learning metric, combines precision and recall into a single value. It represents the model's performance by taking the harmonic mean of precision and recall, thus accounting for both false positives and false negatives. These metrics assess the model's overall performance across various dimensions.

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Table.3. Metrics of evaluation

METRICS	RESNET50	YOLO v8	ROBOFLOW 3.0
ACCURACY	85%	87%	91%
PRECISION	0.87	0.90	0.93
RECALL	0.83	0.87	0.853
F1-SCORE	0.85	0.88	0.879
INFERENCE TIME	Moderate	Fast	Fast

OUTPUTS ::

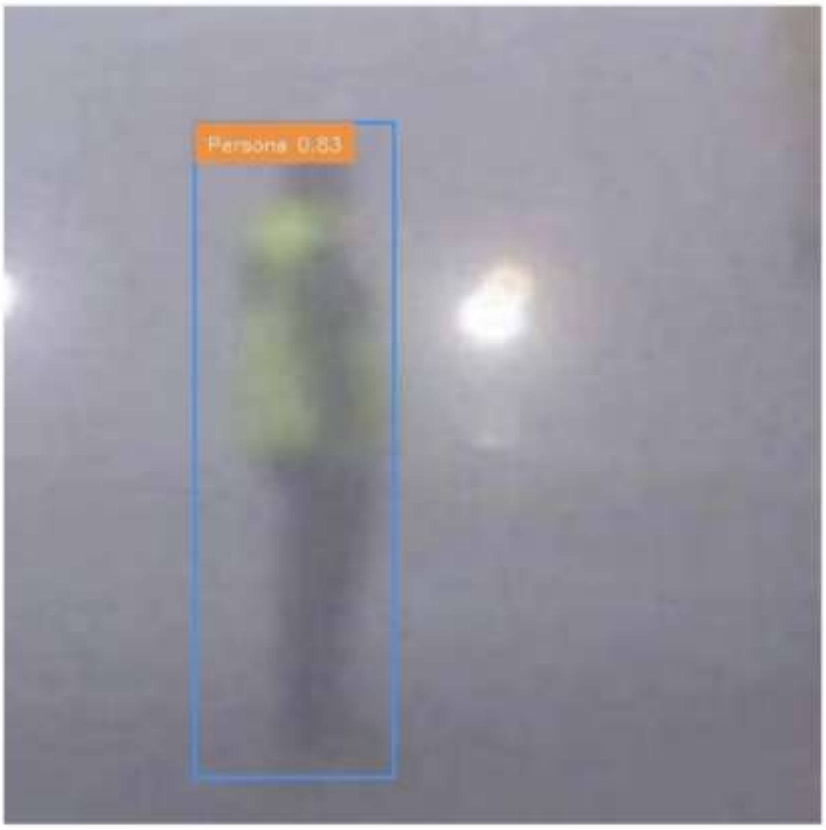

Fig 10.Pedestrian Detection Using Trained Model

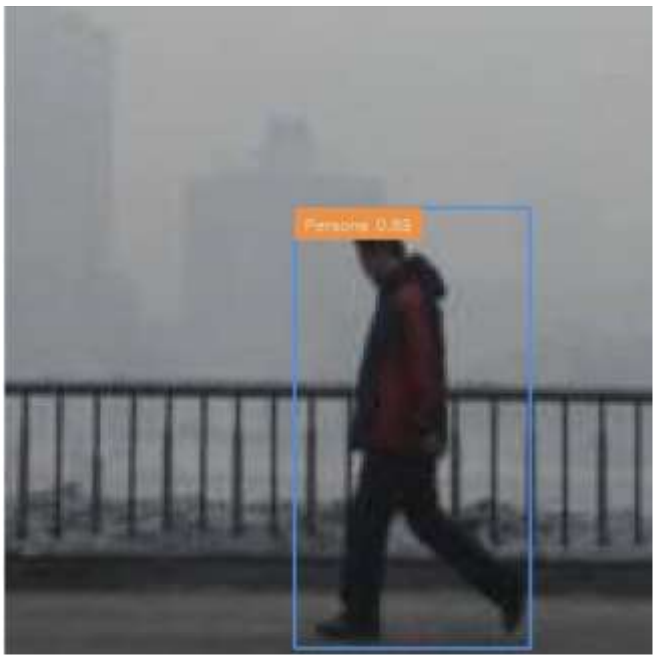

Fig 11.Pedestrian Detection Using Trained Model

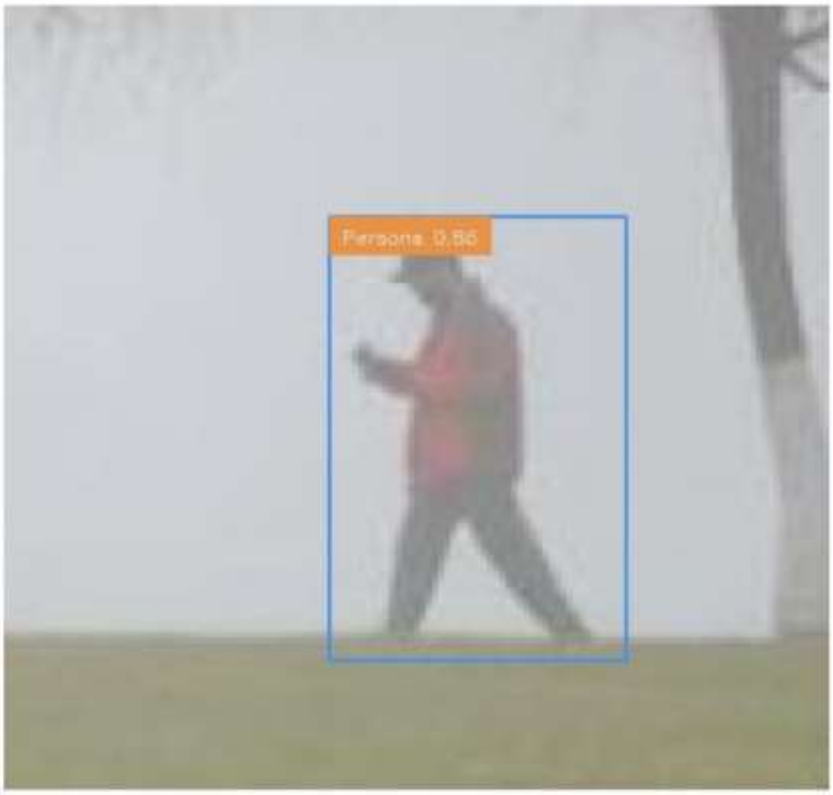

Fig 12.Pedestrian Detection Using Trained Model

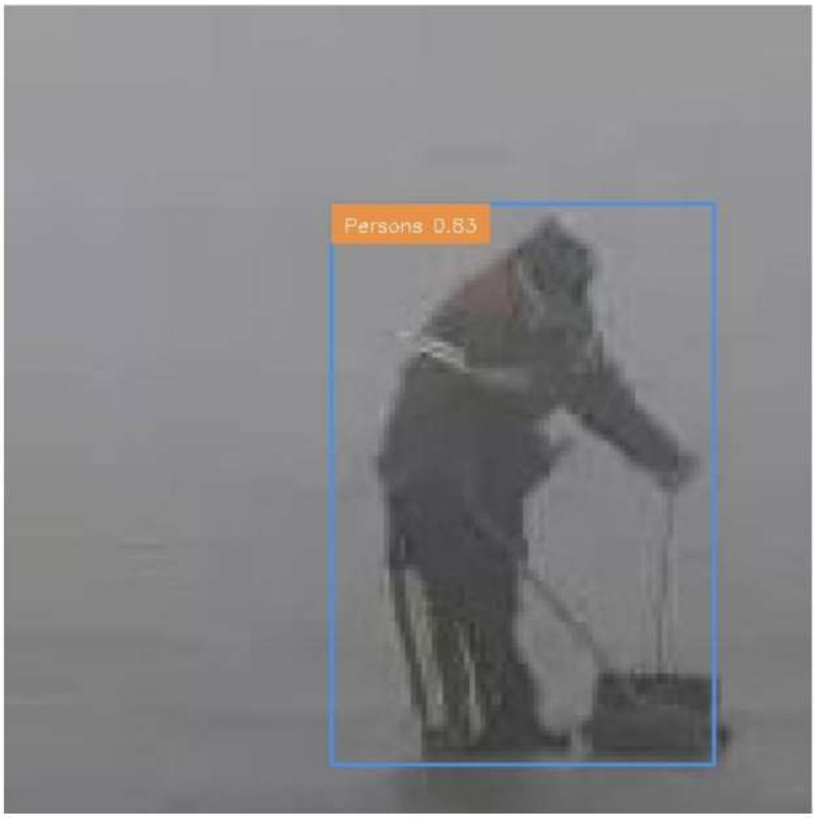

Fig 13.Pedestrian Detection Using Trained Model

4.4 Analysis

Pedestrian detection in adverse weather conditions is critical for ensuring road safety. In this project, we aimed to evaluate the performance of two advanced deep learning models, ResNet50 and YOLOv8, for this task using thermal images collected from datasets like ZUT and OTCBVS repository. The primary objective was to assess their effectiveness in detecting pedestrians in foggy and misty environments.

Both models were trained using transfer learning on the collected datasets images. ResNet50, known for its deep residual architecture, and YOLOv8, an upgraded version of YOLO, were fine-tuned for pedestrian detection. The models were then evaluated on a separate test set using standard evaluation metrics such as accuracy, precision, recall, and F1-score.

The results indicated that both ResNet50 and YOLOv8 performed well in pedestrian detection, achieving accuracies of 85% and 88%, respectively, on the test set. However, YOLOv8 demonstrated slightly higher accuracy and faster inference times compared to ResNet50, making it more suitable for real-time applications in adverse weather conditions.

In terms of scalability, YOLOv8's efficiency in speed and accuracy makes it a preferred choice for deployment in systems requiring real-time pedestrian detection, such as advanced driver assistance systems (ADAS). The findings of this project contribute to enhancing road safety by providing efficient and reliable pedestrian detection systems for extreme weather conditions.

In conclusion, both ResNet50 and YOLOv8 show promise for pedestrian detection in adverse weather conditions, with YOLOv8 demonstrating superior performance in terms of accuracy and speed. These results underscore the importance of utilizing advanced deep learning techniques for improving road safety in challenging environments.

V. CONCLUSION & FUTURE SCOPE

In our project focused on pedestrian detection in adverse weather conditions, we envision an exciting future scope that builds upon our current advancements. Our next phase involves augmenting our dataset to encompass a diverse range of weather scenarios, including snow, rain, fog, and low-light conditions. By doing so, we aim to enhance the robustness and versatility of our pedestrian detection system, ensuring its effectiveness across various real-world environments.

Furthermore, we are committed to advancing the efficiency of our system by implementing state-of-the-art algorithms and optimization techniques. This includes exploring novel approaches in deep learning and computer vision to improve detection accuracy while minimizing computational overhead. By optimizing performance, we can ensure that our system seamlessly integrates with autonomous driving assistance systems, thereby contributing to safer and more reliable transportation solutions.

Moreover, our future endeavors will focus on seamless integration with existing transportation infrastructure and vehicles. This involves collaborating with automotive manufacturers and regulatory bodies to ensure compliance with safety standards and facilitate the adoption of our technology in autonomous vehicles. In addition to technical advancements, we recognize the importance of real-time communication and situational awareness in emergency response scenarios. Therefore, we plan to explore the integration of our detection system with communication platforms to enable swift alerts and comprehensive situational assessments during critical situations.

By pursuing these avenues of research and development, we aim to not only enhance pedestrian safety in adverse weather conditions but also contribute to the ongoing evolution of autonomous driving technology, ultimately paving the way for safer and more efficient transportation systems of the future.

REFERENCES

[1] Y. Jiang, G. Tong, H. Yin and N. Xiong, "A Pedestrian Detection Method Based on Genetic Algorithm for Optimize XGBoost Training Parameters," in IEEE Access, vol. 7, pp. 118310-118321, 2019, doi: 10.1109/ACCESS.2019.2936454.

[2] Viola P, Jones M J. “Robust Real-Time Face Detection,” International Journal of Computer Vision, 2004, vol.57(2), pp.137-154.

[3] Dalal N, Triggs B. “Histograms of Oriented Gradients for Human Detection,” In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005, pp.886-893

[4] David G. Lowe. “Distinctive image features from scale-invariant keypoints,” International Journal of Computer Vision,2004, vol. 60 pp.91-110.

[5] Timo Ojala, Matti PietikaEinen, and Topi MaEenpaEa. “Multire solution gray-scale and rotation invariant texture classification with local binary patterns,” IEEE Transactions on Pattern Analysis and Machine Intelligence, 2004,vol.24(7), pp.971-987.

[6] Paul Viola and Michael J. Jones. Robust real-time face detection. International Journal of Computer Vision, 2004,pp. 57:137-154.

[7] B. Lei, S. Zhang, H. Zhai and J. Shi, "Research on Autonomous Emergency Braking Pedestrain Test of China New Car Assessment Program," ISCTT 2022; 7th International Conference on Information Science, Computer Technology and Transportation, Xishuangbanna, China, 2022, pp. 1-5.

[8] P. Tumas, A. Nowosielski and A. Serackis, "Pedestrian Detection in Severe Weather Conditions," in *IEEE Access*, vol. 8, pp. 62775-62784, 2020, doi: 10.1109/ACCESS.2020.2982539.

[9] D. M. Gavrila and S. Munder, “Vision-based pedestrian protection: The PROTECTOR system,” in Proc. IEEE Intelligent Vehicle Symposium, IV2008, Parma, Italy, June 2008

[10] D. S. M, I. U. S and A. R, "," 2021 Fourth International Conference on Electrical, Computer and Communication Technologies (ICECCT), Erode, India, 2021, pp. 1-6,

doi: 10.1109/ICECCT52121.2021.9616723.

[11] Liu T, Stathaki T. Faster R-CNN for Robust Pedestrian Detection Using Semantic Segmentation Network. Front Neurorobot. 2018 Oct 5;12:64. doi: 10.3389/fnbot.2018.00064. PMID: 30344486; PMCID: PMC6182048.

[12] Hailong Li, Zhendong Wu and Jianwu Zhang, "Pedestrian detection based on deep learning model," 2016 9th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), Datong, 2016, pp. 796-800, doi: 10.1109/CISP-BMEI.2016.7852818.

[13] X. Fan, W. Wei, M. Wozniak et al., “Low energy consumption and data redundancy approach of wireless sensor networks with bigdata,” Information Technology and Control, vol. 47, no. 3, pp. 406–418, 2018.

APPENDIX

GitHub Link: https://github.com/Naveen-reddie/Pedestrian

Source Code –

YOLOV8 Object Detection:

```
!nvidia-smi

!pip install ultralytics

from ultralytics import YOLO

import os

from IPython.display import display, Image

from IPython import display

display.clear_output()

!yolo mode=checks

!pip install roboflow

!yolo cfg

!pip install roboflow



from roboflow import Roboflow

rf = Roboflow(api_key="oxDMjU5rNJd1Y5gXsh6W")

project = rf.workspace("pedestrian-hn0tb").project("pedestrian-detection-okisn")

version = project.version(1)

dataset = version.download("yolov8")

!yolo task=detect mode=train model=yolov8m.pt source="dataset location" epochs=100
imgsz=640
```

```
from IPython.display import Image

from IPython.display import Image

Image(filename=f'/content/runs/detect/train2/results.png', width=600)

!yolo task=detect mode=val model=/content/runs/detect/train2/weights/best.pt
source="dataset location"

!yolo task=detect mode=predict model=/content/runs/detect/train2/weights/best.pt conf=0.5
source={dataset.location}/test/images save_txt=true save_conf=true

import glob

from IPython.display import Image, display

for image_path in glob.glob(f'/content/runs/detect/predict/*.jpg'):

    print(image_path)

    display(Image(filename=image_path, height=600))

    print("\n")
```

RESNET Based Object Detection:

```
# gpu info that is allocated
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Select the Runtime > "Change runtime type" menu to enable a GPU accelerator, ')
  print('and then re-execute this cell.')
else:
  print(gpu_info)
pwd
import os
import sys
import time
import numpy as np
import torch
import PIL
from PIL import Image
from matplotlib import pyplot as plt
import matplotlib.patches as patches
import math
from sklearn.metrics import auc
import torchvision
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, utils
from torch.nn import functional as F

import gc

%matplotlib inline
torch.__version__
class PennFudanDataset(torch.utils.data.Dataset):
    def __init__(self, root, transforms=None):
        self.root = root
        self.transforms = transforms
        self.images = list(sorted(os.listdir(os.path.join(self.root,"PNGImages")))) # returns a list
of file names, sorting and converting it into list again
        self.masks  = list(sorted(os.listdir(os.path.join(self.root,"PedMasks")))) # returns a list of
file names

    def __len__(self):
```

```
    return len(self.images)

def __getitem__(self, index):
    img_path  = os.path.join(self.root,"PNGImages",self.images[index])
    mask_path = os.path.join(self.root,"PedMasks",self.masks[index])
    img = Image.open(img_path).convert("RGB")
    mask = Image.open(mask_path)
    mask = np.array(mask)

    # instances are encoded as different colors
    obj_ids = np.unique(mask)
    # first id is the background, so remove it
    obj_ids = obj_ids[1:]

    # split the color-encoded mask into a set
    # of binary masks
    masks = mask == obj_ids[:, None, None] # separate binary mask for each instance

    # get bounding box coordinates for each mask
    num_obj = len(obj_ids)
    boxes = []

    for i in range(num_obj):
        curr_mask = np.where(masks[i])
        xmin = np.min(curr_mask[1])
        xmax = np.max(curr_mask[1])
        ymin = np.min(curr_mask[0])
        ymax = np.max(curr_mask[0])
        assert (xmin * xmax * ymin * ymax) >0
        boxes.append([xmin,ymin,xmax,ymax])
        # convert everything into Torch tensors
        boxes = torch.as_tensor(boxes, dtype=torch.float32)
        # there is only one class in this database, Pedestrian since we removed background
        labels = torch.ones((num_obj,), dtype=torch.int64)
        masks = torch.as_tensor(masks.astype('uint8'), dtype=torch.uint8)

        image_id = torch.tensor([index])
        area = (boxes[:, 3] - boxes[:, 1]) * (boxes[:, 2] - boxes[:, 0])
        # suppose all instances are not crowd
        iscrowd = torch.zeros((num_obj,), dtype=torch.int64)

        target = {}
        target["boxes"] = boxes
        target["labels"] = labels
        target["masks"] = masks
        target["image_id"] = image_id
        target["area"] = area
        target["iscrowd"] = iscrowd

        if self.transforms is not None:
            img, target = self.transforms(img, target)

        return img, target
root = "/content/PennFudanPed/"
PennFudanDataset_obj = PennFudanDataset(root)
print(f'Number of images in dataset: {len(PennFudanDataset_obj)}')
# get a random image from the dataset
im_id = np.random.choice(len(PennFudanDataset_obj))
image, target = PennFudanDataset_obj[im_id]
```

```
print(f'image shape is: {np.array(image).shape}')
fig, ax = plt.subplots(1, figsize=(15,20))
ax.imshow(image)
ax.axis('off')
print(f'Number of Instances in Image: {target["boxes"].shape[0]}')
# draw bounding boxes
num_obj = target["boxes"].shape[0]
for i in range(num_obj):
    # Create a Rectangle patch
    curr_box = (target["boxes"][i]).numpy()
    x0,y0,x1,y1 = curr_box[0], curr_box[1], curr_box[2], curr_box[3]
    width = x1 - x0
    height = y1 - y0
    rect = patches.Rectangle((x0,y0),width,height,linewidth=4,edgecolor='r',facecolor='none')
    # Add the patch to the Axes
    ax.add_patch(rect)
plt.show()
im_id
if 0:
  index = 100
  images = list(sorted(os.listdir(os.path.join(root,"PNGImages")))) # returns a list of file
names, sorting and converting it into list again
  masks  = list(sorted(os.listdir(os.path.join(root,"PedMasks")))) # returns a list of file names
  img_path  = os.path.join(root,"PNGImages",images[index])
  mask_path = os.path.join(root,"PedMasks",masks[index])
  img = Image.open(img_path).convert("RGB")
  w, h = img.size # this returns width/height
  img = img.resize((300,300), PIL.Image.BILINEAR)

  scale_x, scale_y = 300/w, 300/h
```

```
mask = Image.open(mask_path)
mask = np.array(mask)
print(img.size, np.array(mask).shape)
# instances are encoded as different colors
obj_ids = np.unique(mask)
# first id is the background, so remove it
obj_ids = obj_ids[1:]
masks = mask == obj_ids[:, None, None]
#print(masks.shape)
# get bounding box coordinates for each mask
num_obj = len(obj_ids)
boxes = []

for i in range(num_obj):
    curr_mask = np.where(masks[i])
    xmin = np.int(np.min(curr_mask[1])*scale_x)
    xmax = np.int(np.max(curr_mask[1])*scale_x)
    ymin = np.int(np.min(curr_mask[0])*scale_y)
    ymax = np.int(np.max(curr_mask[0])*scale_y)
    #w = abs(xmax - xmin)
    #h = abs(ymax - ymin)

    #if min(h,w) < 20: # skip bounding boxes if height or width < 20 pixels
    #  continue

    boxes.append([xmin,ymin,xmax,ymax])
print(boxes)
fig, ax = plt.subplots(1, figsize=(15,20))
ax.imshow(img)
ax.axis('off')
```

```
  for i in range(len(boxes)):
      # Create a Rectangle patch
      curr_box = boxes[i]
      x0,y0,x1,y1 = curr_box[0], curr_box[1], curr_box[2], curr_box[3]
      width = x1 - x0
      height = y1 - y0
      rect = patches.Rectangle((x0,y0),width,height,linewidth=4,edgecolor='r',facecolor='none')
      # Add the patch to the Axes
      ax.add_patch(rect)
  plt.show()
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)
model
for name, child in model.named_children():
  print(f'name is: {name}')
  print(f'module is: {child}')

for key, value in model._modules.items():
  print(f'key is: {key}')
  print(f'value is: {value}')
for name, param in model.named_parameters():
  print(f'name is: {name}')
  print(f'module is: {param}')
model.backbone
model.rpn
isinstance(model.roi_heads.box_head.fc7, torch.nn.Linear)
model.roi_heads
model.roi_heads.box_predictor
from torch.utils.checkpoint import checkpoint, checkpoint_sequential

class CheckpointModule(torch.nn.Module):
  def __init__(self, module, num_segments=1):
    super(CheckpointModule, self).__init__()
    assert num_segments == 1 or isinstance(module, nn.Sequential)
    self.module = module
    self.num_segments = num_segments
    self.dummytensor = torch.ones(1, dtype=torch.float32, requires_grad=True)



  def forward(self, *inputs):
    if self.num_segments > 1:
        return checkpoint_sequential(self._forward_patch_seq, self.dummytensor,
self.num_segments, *inputs)

    else:
        return checkpoint(self._forward_patch, self.dummytensor, *inputs)

  def _forward_patch(self, dummy, *inputs):
    return self.module(*inputs)

  def _forward_patch_seq(self, dummytensor, num_segments, *inputs):
    return self.module(self.num_segments, *inputs)

def get_model(num_classes, chkptEn=False):
  model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)
  in_features = model.roi_heads.box_predictor.cls_score.in_features
  model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

  # freeze the backbone (it will freeze the body and fpn params)
  for p in model.backbone.parameters():
    p.requires_grad = False
  # freeze the fc6 layer in roi_heads
  for p in model.roi_heads.box_head.fc6.parameters():
    p.requires_grad = False

  if chkptEn==True:
    # wrap the model.roi_heads.box_head.fc7 under gradient checkpointing
    model.roi_heads.box_head.fc7 = CheckpointModule(model.roi_heads.box_head.fc7)
    model.rpn.head.conv = CheckpointModule(model.rpn.head.conv)



  return model

model = get_model(num_classes=2)

# save the initial model
state = {
    'state_dict': model.state_dict()

}
torch.save(state, '/content/initialchkpt.pt')
 model
# code snippet how to reload the state later
if 0:
  state = torch.load('/content/initialchkpt.pt')
  model.load_state_dict(state['state_dict'])
class Rescale():

  def __init__(self, h, w):
    self.h = h
    self.w = w

  def __call__(self, image, target):
    w_, h_ = image.size # retuns width, height
    image = image.resize((self.w,self.h), PIL.Image.BILINEAR)
    scale_x, scale_y = self.w/w_, self.h/h_ # resizing ratio

    #scale the bounding boxes
    num_obj = len(target['boxes'])
    for i in range(num_obj):
      box = target['boxes'][i] #x0, y0, x1, y1
      box[0] *= scale_x
      box[2] *= scale_x
      box[1] *= scale_x
      box[3] *= scale_y
      target['boxes'][i] = box

    return image, target

class RandomHorizontalFlip():

    def __init__(self, p):
        self.p = p # probability of flip

    def __call__(self, image, target):
        if np.random.random() < self.p:
            # flip image horizontally
            image = torchvision.transforms.functional.hflip(image) # works on PIL image
            w, h  = image.size
            # flip mask horizontally
            #target['masks'] = target['masks'].flip(-1)
            #masks = target['masks'].numpy()# <num_obj, H, W> binary array
            #masks = np.flip(masks, axis=-1) # flip horizontally on width dimension
            #masks = torch.from_numpy(masks) # make torch tensor
            #target['masks'] = masks
            # flip the box cordinates horizontally
            boxes = target['boxes'] #x0, y0, x1, y1
            boxes[:,[0,2]] = w - boxes[:,[2,0]]
            assert (boxes > 0).all() > 0
            return image, target

        return image, target

    def __repr__(self):
        return self.__class__.__name__ + f'(p={self.p})'

class ToTensor():

    def __call__(self, image, target):
        image = np.array(image)
        #image = image.transpose((2,0,1))
        return torchvision.transforms.functional.to_tensor(image), target # to_tensor permutes
the image to have channel dim first

    def __repr__(self):
        return self.__class__.__name__ + f'()'

class Compose():
    def __init__(self, transforms):
        self.transforms = transforms
    def __call__(self, image, target):
        for t in self.transforms:
            image, target = t(image, target)
        return image, target
#train_transform_compose = Compose([ToTensor()])
train_transform_compose = Compose([RandomHorizontalFlip(p=0.5), ToTensor()])
#test_transform_compose = Compose([ToTensor()])
test_transform_compose = Compose([ToTensor()])
def collate_fn(batch):
    return tuple(zip(*batch))
train_dataset = PennFudanDataset(root, transforms=train_transform_compose)
test_dataset = PennFudanDataset(root, transforms=test_transform_compose)

# split the data b/w train and validation randomly
indx = np.random.choice(len(train_dataset), size=len(train_dataset), replace=False)
print(indx)
# train data
train_dataset_ = torch.utils.data.Subset(train_dataset, indx[:-50]) # keep but last 50 datapoints
in train set
# valid data
test_dataset_ = torch.utils.data.Subset(test_dataset, indx[-50:]) # last 50 datapoints in
validation set
train_dataloader = DataLoader(train_dataset_, batch_size=2, shuffle=True,
collate_fn=collate_fn)
# test dataloader
test_dataloader = DataLoader(test_dataset_, batch_size=2, shuffle=True,
collate_fn=collate_fn)
print(len(train_dataset_), len(test_dataset_))
```
- set up device, optimizer, learning scheduler

```
# setting up device
device = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")

params = [p for p in  model.parameters() if p.requires_grad]
#optimizer = torch.optim.Adam(params, lr=0.005)
optimizer = torch.optim.SGD(params, lr=0.005,
                            momentum=0.9, weight_decay=0.0005)

# and a learning rate scheduler which decreases the learning rate by
# 10x every 3 epochs
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)
model.to(device)
x, y = next(iter(train_dataloader))
# number of batches
len(train_dataloader)
num_batches_train = len(train_dataloader)
num_batches_test  = len(test_dataloader)
def train_one_epoch(model, dataloader, optimizer, device):
    # set the model in train mode
    model.train()
    output = []
    batch_num = 0
    for images, targets in dataloader: # returns batch of data every iteration

      # send images in this batch to device
      images = list(image.to(device) for image in images) # list of images
      # send target dict in this bacth to device
      targets = [{k: v.to(device) for k,v in t.items()} for t in targets] # list of dict
      # zero out the gradients so that they do not accumulate (default behaviour) in successive
passes
      optimizer.zero_grad()

      # forward pass
      try:
        op = model(images, targets)
      except RuntimeError as e:
        if 'CUDA out of memory' in str(e):
          print('| WARNING: ran out of memory, retrying batch', sys.stdout)
          sys.stdout.flush()
          for p in model.parameters():
              if p.grad is not None:
                  del p.grad  # free some memory
          torch.cuda.empty_cache()
          op = model(images, targets)
        else:
          raise e

      losses = sum(val for val in op.values())

      # run backward pass
      losses.backward()
      # update optimizer
      optimizer.step()

      output.append([op['loss_box_reg'].data, op['loss_classifier'].data,
op['loss_objectness'].data, op['loss_rpn_box_reg'].data])
      batch_num += 1
      print(f'batch:{batch_num}/{num_batches_train}')
```

```
      del images, targets, op
      gc.collect()

    return output



device
fwd_pass_op = train_one_epoch(model, train_dataloader, optimizer, device)
fwd_pass_op
# redefine network by using gradient checkpointing
model = get_model(num_classes=2, chkptEn=True)
state = {
    'state_dict': model.state_dict()

}
torch.save(state, '/content/initialchkpt.pt')

# set the model on device
model.to(device)
state = torch.load('/content/initialchkpt.pt')
model.load_state_dict(state['state_dict'])
params = [p for p in  model.parameters() if p.requires_grad]
#optimizer = torch.optim.Adam(params, lr=0.005)
optimizer = torch.optim.SGD(params, lr=0.005,
                            momentum=0.9, weight_decay=0.0005)
for n, param in model.named_parameters():
  if param.requires_grad==True:
    print(n)
# create initial copy of wt just to verify later on after training whether it changed or not (we
do this for checkpointed layer parameter weight)
```

```
wt_ = model.roi_heads.box_head.fc7.module.weight.clone()
wt_.shape
def lr_finder(model, optimizer, start_lr, end_lr, dataloader, beta=0.98, terminate_factor = 10,
device='cpu'):
  model.train()
  # compute the mult factor which needs to be multiplied with start_lr
  num = len(dataloader)-1
  mult_fac = (end_lr/start_lr)**(1/num)
  lr = start_lr
  optimizer.param_groups[0]['lr'] = lr

  # define list to store the smooth loss and learning rate
  smooth_loss_list  = []
  lr_list = []
  batch_num = 0
  avg_loss = 0
  best_loss = 0

  for images, targets in dataloader:
    batch_num += 1

    # send images in this batch to device
    images = list(image.to(device) for image in images) # list of images

    targets = [{k: v.to(device) for k,v in t.items()} for t in targets] # list of dict
    optimizer.zero_grad()
    # forward pass and compute the loss
    loss_dict = model(images, targets) # this model already outputs the loss dict
    losses = sum(val for val in loss_dict.values())
    avg_loss = beta*avg_loss + (1-beta)*losses.data
    # de-embedd the starting bias
    smooth_loss = avg_loss/(1-beta**batch_num)
    if batch_num > 1 and smooth_loss > terminate_factor*best_loss:
      print("loss diverging, terminating early...")
      return smooth_loss_list, lr_list

    # record the best loss
    if smooth_loss < best_loss or batch_num == 1:
      best_loss = smooth_loss

    print(f'batch_num:{batch_num}, avg_loss:{avg_loss}, smooth_loss:{smooth_loss},
lr:{lr}')

    smooth_loss_list.append(smooth_loss)
    lr_list.append(lr)

    # backward pass
    losses.backward()
    # update optimizer
    optimizer.step()

    print((model.roi_heads.box_head.fc7.module.weight - wt_).abs().sum())
    lr *= mult_fac
    optimizer.param_groups[0]['lr'] = lr

  return smooth_loss_list, lr_list

start_lr =1e-6
end_lr = 1
```

```
loss_list, lr_list = lr_finder(model, optimizer, start_lr, end_lr, train_dataloader, beta=0.98,
terminate_factor = 10, device= device)
loss_list
plt.figure()
plt.semilogx(lr_list, loss_list)
plt.grid(True)
state = torch.load('/content/initialchkpt.pt')
model.load_state_dict(state['state_dict'])
params = [p for p in  model.parameters() if p.requires_grad]
#optimizer = torch.optim.Adam(params, lr=1e-3)
optimizer = torch.optim.SGD(params, lr=0.005,
                            momentum=0.9, weight_decay=0.001)
# and a learning rate scheduler which decreases the learning rate by
# 10x every 3 epochs
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)

num_batches_train = len(train_dataloader)
num_batches_test  = len(test_dataloader)
def train_one_epoch(model, dataloader, optimizer, device, epoch_num):
    model.train()
    op_dict = {} # will be returned as op for only end of epoch stats
    output = [] # will contain info for all batches
    batch_num = 0
    for images, targets in dataloader: # returns batch of data every iteration

        # send images in this batch to device
        images = list(image.to(device) for image in images) # list of images
        # send target dict in this bacth to device
        targets = [{k: v.to(device) for k,v in t.items()} for t in targets] # list of dict
    optimizer.zero_grad()

    try:
      loss_dict = model(images, targets)
    except RuntimeError as e:
      if 'CUDA out of memory' in str(e):
        print('| WARNING: ran out of memory, retrying batch', sys.stdout)
        sys.stdout.flush()
        for p in model.parameters():
            if p.grad is not None:
                del p.grad  # free some memory
        torch.cuda.empty_cache()
        loss_dict = model(images, targets)
      else:
        raise e
    del images, targets

    losses = sum(val for val in loss_dict.values())

    if not math.isfinite(losses):
      print("Loss is {}, stopping training".format(losses))
      print(f'epoch:{epoch_num}, batch:{batch_num}/{num_batches_train}')
      sys.exit(1)

    losses.backward()
    optimizer.step()

    output.append([loss_dict['loss_box_reg'].data, loss_dict['loss_classifier'].data,
loss_dict['loss_objectness'].data, loss_dict['loss_rpn_box_reg'].data])
    batch_num += 1
```

```
      print(f'epoch:{epoch_num}, batch:{batch_num}/{num_batches_train}')



    op_dict['epoch'] = epoch_num
    op_dict['total_loss'] = losses
    op_dict['loss_classifier'] = loss_dict['loss_classifier'].data
    op_dict['loss_box_reg'] = loss_dict['loss_box_reg'].data
    op_dict['loss_objectness'] = loss_dict['loss_objectness'].data
    op_dict['loss_rpn_box_reg'] = loss_dict['loss_rpn_box_reg'].data
    return output, op_dict
```

- train over multiple epochs

```
num_epochs = 10
loss_classifier, loss_box_reg, loss_objectness, loss_rpn_box_reg  = [], [], [], []
for epoch_num in range(num_epochs):
  output, op_dict = train_one_epoch(model, train_dataloader, optimizer, device, epoch_num)

  loss_classifier.append(op_dict['loss_classifier'])
  loss_box_reg.append(op_dict['loss_box_reg'])
  loss_objectness.append(op_dict['loss_objectness'])
  loss_rpn_box_reg.append(op_dict['loss_rpn_box_reg'])
  print(f'epoch:{epoch_num}')
  print(f"loss_cls:{op_dict['loss_classifier']}, loss_box_reg:{op_dict['loss_box_reg']}")
  print(f"loss_obj:{op_dict['loss_objectness']}, loss_rpn_reg:{op_dict['loss_rpn_box_reg']}")

  if lr_scheduler is not None:
    lr_scheduler.step()
  pass # TBD
# plot the epoch end stats
total_loss = np.array(loss_classifier) + np.array(loss_box_reg) + np.array(loss_objectness) +
np.array(loss_rpn_box_reg)
plt.figure(figsize=(10,10))
plt.plot(np.arange(num_epochs), loss_classifier, label='classifier')
plt.plot(np.arange(num_epochs), loss_box_reg, label='loss_box_reg')
plt.plot(np.arange(num_epochs), loss_objectness, label='loss_objectness')
plt.plot(np.arange(num_epochs), loss_rpn_box_reg, label='loss_rpn_box_reg')
plt.plot(np.arange(num_epochs), total_loss, label='total_loss')
plt.legend()
# save the trained weights
# save the initial model
state = {
    'state_dict': model.state_dict()

}
torch.save(state, '/content/ModelafterTrainchkpt.pt')
optim_state = {
    'state_dict': optimizer.state_dict()
}
torch.save(state, '/content/Optimizertrainchkpt.pt')
def  compute_IOU(pred_box_cord, ground_truth_box_cord):
  num_gt = len(ground_truth_box_cord)
  IOU = np.zeros(num_gt) # for every predicted box, compute IOU with all ground truth
boxes
  x0, y0, x1, y1 = pred_box_cord[0], pred_box_cord[1], pred_box_cord[2], pred_box_cord[3]
  area_pred = (x1-x0)*(y1-y0)
  x0_g, y0_g, x1_g, y1_g = ground_truth_box_cord[0], ground_truth_box_cord[1],
ground_truth_box_cord[2], ground_truth_box_cord[3]
```

```
  area_gt = (x1_g - x0_g)*(y1_g - y0_g)

  xmax = np.max([x0,x0_g])
  xmin = np.min([x1,x1_g])
  ymax = np.max([y0,y0_g])
  ymin = np.min([y1,y1_g])
  if (xmax < xmin) and (ymax < ymin): # there is overlap
    overlap_area = (xmin - xmax) *(ymin - ymax)
    IOU = overlap_area / (area_pred + area_gt - overlap_area)
  else:
    IOU =0
  return IOU
def evaluate_sample(target_pred, target_true, iou_threshold=0.5):

  gt_bboxes = target_true['boxes'].numpy()
  gt_labels = target_true['labels'].numpy()

  dt_bboxes = target_pred['boxes'].numpy()
  dt_labels = target_pred['labels'].numpy()
  dt_scores = target_pred['scores'].numpy()

  results = []
  for detection_id in range(len(dt_labels)): # loop over every detection
    dt_bbox = dt_bboxes[detection_id, :]
    dt_label = dt_labels[detection_id]
    dt_score = dt_scores[detection_id]

    detection_result_dict = {'score': dt_score}

    max_IoU = 0
```

```
      max_gt_id = -1
      for gt_id in range(len(gt_labels)): # loop over ground truth
        gt_bbox = gt_bboxes[gt_id, :]
        gt_label = gt_labels[gt_id]

        if gt_label != dt_label:
          continue

        if compute_IOU(dt_bbox, gt_bbox) > max_IoU:
          max_IoU = compute_IOU(dt_bbox, gt_bbox)
          max_gt_id = gt_id

      if max_gt_id >= 0 and max_IoU >= iou_threshold:
        detection_result_dict['TP'] = 1
        gt_labels = np.delete(gt_labels, max_gt_id, axis=0)
        gt_bboxes = np.delete(gt_bboxes, max_gt_id, axis=0)

      else:
        detection_result_dict['TP'] = 0

      results.append(detection_result_dict)

  return results
def evaluate(model, test_loader, device):
  results = []
  model.eval()
  nbr_boxes = 0
  with torch.no_grad():
    for batch, (images, targets_true) in enumerate(test_loader):
      images = list(image.to(device).float() for image in images)
        targets_pred = model(images) # output is a list of dictionaries with each one having keys: boxes, labels, scores

        targets_true = [{k: v.cpu().float() for k, v in t.items()} for t in targets_true]
        targets_pred = [{k: v.cpu().float() for k, v in t.items()} for t in targets_pred]

        for ii in range(len(targets_true)): # loop over number of images in the batch
          target_true = targets_true[ii]
          target_pred = targets_pred[ii]
          nbr_boxes += target_true['labels'].shape[0]

          results = results + evaluate_sample(target_pred, target_true)

      results = sorted(results, key=lambda k: k['score'], reverse=True) # sort iterable (list in this case) based on dictionary key: 'score'

      acc_TP = np.zeros(len(results))
      acc_FP = np.zeros(len(results))
      recall = np.zeros(len(results))
      precision = np.zeros(len(results))

      if results[0]['TP']==1:
        acc_TP[0] = 1
      else:
        acc_FP[0] = 1

      for i in range(1,len(results)):
        acc_TP[i] = acc_TP[i-1] + results[i]['TP']
        acc_FP[i] = acc_FP[i-1] + (1 - results[i]['TP'])
        # calculate precision and recall
        precision[i] = acc_TP[i]/(acc_TP[i] + acc_FP[i])
        recall[i] = acc_TP[i]/nbr_boxes

      return auc(recall, precision)



state = torch.load('/content/initialchkpt.pt')
model.load_state_dict(state['state_dict'])
params = [p for p in  model.parameters() if p.requires_grad]
#optimizer = torch.optim.Adam(params, lr=1e-3)
optimizer = torch.optim.SGD(params, lr=0.005,
                            momentum=0.9, weight_decay=0.001)
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)
num_epochs = 10
loss_classifier, loss_box_reg, loss_objectness, loss_rpn_box_reg  = [], [], [], []
for epoch_num in range(num_epochs):

  start = time.time()
  output, op_dict = train_one_epoch(model, train_dataloader, optimizer, device, epoch_num)
  mAP = evaluate(model, test_dataloader, device)
  end = time.time()

  loss_classifier.append(op_dict['loss_classifier'])
  loss_box_reg.append(op_dict['loss_box_reg'])
  loss_objectness.append(op_dict['loss_objectness'])
  loss_rpn_box_reg.append(op_dict['loss_rpn_box_reg'])
  print(f'epoch:{epoch_num}')
  print(f"loss_cls:{op_dict['loss_classifier']}, loss_box_reg:{op_dict['loss_box_reg']}")
  print(f"loss_obj:{op_dict['loss_objectness']}, loss_rpn_reg:{op_dict['loss_rpn_box_reg']}")
  print(f'Time for train+eval: {end - start}')
  print(f'mAP: {mAP}')

  # run learning rate scheduler step if defined
  if lr_scheduler is not None:
    lr_scheduler.step()



  if (epoch_num+1) % 3 == 0:
    print(f'saving model afte epoch: {epoch_num}')
    torch.save(model.state_dict(), os.path.join('/content/', 'model_dict_' +
str(epoch_num)+".pth"))

torch.save(model.state_dict(), os.path.join('/content/', 'model_dict_' +
str(epoch_num)+".pth"))
total_loss = np.array(loss_classifier) + np.array(loss_box_reg) + np.array(loss_objectness) +
np.array(loss_rpn_box_reg)
plt.figure(figsize=(10,10))
plt.plot(np.arange(num_epochs), loss_classifier, label='classifier')
plt.plot(np.arange(num_epochs), loss_box_reg, label='loss_box_reg')
plt.plot(np.arange(num_epochs), loss_objectness, label='loss_objectness')
plt.plot(np.arange(num_epochs), loss_rpn_box_reg, label='loss_rpn_box_reg')
plt.plot(np.arange(num_epochs), total_loss, label='total_loss')
plt.legend()
idx = np.random.choice(len(test_dataset_)) #4, 47
image, target_gt = test_dataset_[idx]
#{{perform prediction
state = torch.load('/content/model_dict_9.pth')
model.load_state_dict(state)
model.to(device)
imagecopy = image.clone()
imagecopy.unsqueeze_(0) # add batch dimension
imagecopy = list(imagecopy.to(device)) # list of image
model.eval()
with torch.no_grad():
  target_pred = model(imagecopy) # in eval mode, only image is sent as input
target_pred = [{k: v.to("cpu") for k,v in t.items()} for t in target_pred]
target_pred = target_pred[0] # picking 1st element of list which is dict since there is only 1
image for prediction
#}}
image = image.numpy()
image = np.transpose(image, axes=[1,2,0])
fig, ax = plt.subplots(1, figsize=(15,20))
ax.imshow(image)
ax.axis('off')
print(f'Number of Ground Truth Instances in Image: {target_gt["boxes"].shape[0]}')
num_obj = target_gt["boxes"].shape[0]
for i in range(num_obj):
    curr_box = (target_gt["boxes"][i]).numpy()
    x0,y0,x1,y1 = curr_box[0], curr_box[1], curr_box[2], curr_box[3]
    width = x1 - x0
    height = y1 - y0
    rect = patches.Rectangle((x0,y0),width,height,linewidth=4,edgecolor='r',facecolor='none')
    ax.add_patch(rect)

num_obj_pred = target_pred["boxes"].shape[0]
print(f'Number of Predicted Instances in Image: {target_pred["boxes"].shape[0]}')


for i in range(num_obj_pred):
    # Create a Rectangle patch
    curr_box = (target_pred["boxes"][i]).numpy()
    confidence = (target_pred["scores"][i]).numpy()
    confidence = np.round(confidence, 2)
    if confidence < 0.7:
      continue
    x0,y0,x1,y1 = curr_box[0], curr_box[1], curr_box[2], curr_box[3]
    width = x1 - x0
    height = y1 - y0
    rect = patches.Rectangle((x0,y0),width,height,linewidth=4,edgecolor='b',facecolor='none')
    ax.add_patch(rect)
    ax.annotate(str(confidence), (x0,y0), fontsize=14, bbox={'facecolor': 'm', 'alpha': 0.3, 'pad':
4})

plt.show()
target_pred = [{k: v.to("cpu") for k,v in t.items()} for t in target_pred]
target_pred
```

Pedestrian Detection In Bad Weather Conditions

Dr.M.Neelakantappa
Information Technology
Vasavi College of Engineering
Hyderabad, India

Nomula Naveen Reddy
Information Technology
Vasavi College of Engineering
Hyderabad, India

Chenreddy Narendra Sai
Information Technology
Vasavi College of Engineering
Hyderabad, India

Baddireddy Hemanth Reddy
Information Technology
Vasavi College of Engineering
Hyderabad, India

Dr. Kovvur Ram Mohan Rao
Information Technology
Vasavi College of Engineering
Hyderabad, India

Abstract—**Pedestrian detection is vital for ensuring safety in autonomous driving, surveillance, and urban planning, enabling systems to identify and track individuals in real-time. It plays a crucial role in preventing accidents, optimizing traffic flow, and enhancing security in public spaces. Additionally, pedestrian detection facilitates human-computer interaction and accessibility for individuals with disabilities. The method proposed in the paper, improves the accuracies of detection, and identifies the accurate pedestrian positions. This work proposes to mitigate the challenges posed by poor visibility due to inclement weather, thereby enhancing pedestrian safety and reducing the risk of accidents in such conditions. This project presents a pedestrian detection system based on ResNet and YOLOv8 architectures, leveraging their capabilities in feature extraction and real-time object detection. Our results demonstrate the effectiveness of comparing ResNet and YOLOv8 for pedestrian detection tasks, with implications for applications in autonomous driving, surveillance, and urban planning.**

Index Terms—**YOLO, RESNET, CNN, ROBOFLOW**

I. Introduction

Pedestrian detection is a critical task in various applications such as autonomous driving, surveillance, and urban planning. However, it poses several challenges, including occlusion, varying scales, complex backgrounds, and changes in lighting conditions. Existing pedestrian detection systems often struggle to achieve high accuracy and robustness in real-world scenarios due to these challenges.Pedestrian detection is a critical component in various domains such as intelligent surveillance, automotive assistance systems, security, and smart transportation. Over the years, it has garnered significant attention in numerous computer vision applications. Particularly, machine learning has gained increasing traction for pedestrian detection.

In general, it is realized by building pedestrian feature extraction model and building feature classifier [1]. Classic methods for target detection rely on traditional frameworks such as sliding windows or manually selected features. These include techniques like Viola-Jones (VJ) [2], Histogram of Oriented Gradients (HOG) [3], Scale-Invariant Feature Transform (SIFT) [4], Local Binary Patterns (LBP) [5], and Haar-like features [6]. These methods utilize handcrafted models and are typically classified using sliding windows and linear Support Vector Machine (SVM) [6].

In particular, occlusion remains a significant obstacle, where pedestrians may be partially or fully obstructed by other objects or by other pedestrians. Traditional methods often fail to correctly identify pedestrians in these situations, leading to missed detections or false alarms. Additionally, variations in scale, such as pedestrians appearing at different sizes due to distance or perspective, further complicate the detection process. Furthermore, changes in lighting conditions, such as shadows, glare, or low light, can significantly affect the visibility of pedestrians, making it challenging for detection algorithms to accurately identify them.

In such conditions, existing pedestrian detection systems often struggle to distinguish pedestrians from the background clutter or misclassify objects due to poor visibility. The dynamic nature of weather conditions further complicates the task, requiring robust algorithms capable of adapting to varying degrees of visibility and environmental factors. Addressing the problem of pedestrian detection in bad weather conditions requires the development of innovative algorithms and techniques that can effectively handle degraded image quality and environmental challenges. This includes exploring methods for enhancing image visibility, such as adaptive image enhancement or fusion techniques, as well as leveraging advanced deep learning architectures that are robust to noise and partial occlusions.

II. Related Work

Pedestrian safety is a critical concern in modern transportation systems, with pedestrian-related accidents accounting for a significant portion of global road fatalities. Optimization techniques have been proposed to enhance the performance of pedestrian detection algorithms, with XGBoost[1] being a notable example. XGBoost, derived from Gradient Boosting Decision Trees (GBDT), offers fast convergence and generalization abilities, making it suitable for various machine learning tasks .Several optimization methods, such as scenario-based optimization algorithms and Bayesian optimization, have been developed to further improve the accuracy and stability of XGBoost models [1]. Various approaches have been proposed to improve pedestrian detection, a crucial component of intelligent transportation systems and automotive safety. Machine learning algorithms, particularly boosting techniques like XGBoost, have gained prominence in pedestrian detection due to their ability to handle complex data and improve classification accuracy[1].

Deep learning techniques, specially Convolutional Neural Networks (CNNs), became powerful tools for pedestrian detection, as they can learn discriminative features from raw input data automatically. CNN-based approaches have shown promise in improving detection accuracy, although they require substantial computational resources. In addition to feature extraction, the choice of machine learning algorithm is crucial for pedestrian classification. Support Vector Machines (SVMs), neural networks, random forests, and Adaboost are commonly used classifiers in pedestrian detection tasks. SVMs, in particular, are praised for their effectiveness in linear classification, although they may struggle with nonlinear problems [2]. Feature extraction plays a crucial role in pedestrian detection, with Histogram of Oriented Gradients (HOG) [3] being one of the most commonly used methods. However, HOG features suffer from high dimensionality, which can affect both training and detection speed. To address this limitation, researchers have explored alternative features such as Local Binary Patterns (LBP) and integral channel features (ACF) to enhance computational efficiency without compromising accuracy [3].

According to the WHO, nearly 1.24 million people died in road traffic accidents in 2013, with pedestrians comprising 22% of the total fatalities [7]. In China, pedestrian accidents constitute approximately 20% of the total number of traffic accidents, with pedestrian casualties accounting for about 30% of the total casualties [7].

Recognizing the importance of pedestrian safety, regulatory bodies such as the China New Car Assessment Programme (C-NCAP) have introduced pedestrian test evaluation scenarios to promote vehicle safety standards. The latest editions of C-NCAP[8] have expanded and tightened these test scenarios to address the growing concern of pedestrian-related accidents.

Pedestrian detection algorithms are evaluated using metrics such as the (AUC) Area Under the ROC Curve to assess their performance. Validation techniques, including track testing and comparison with existing algorithms, are crucial for ensuring the effectiveness and practicality of new methods in real-world scenarios [8]. The advancements in pedestrian detection algorithms have significant implications for intelligent monitoring, automotive assisted driving, security, and intelligent transportation systems, contributing to safer road environments and reducing the number of pedestrian-related accidents. Adverse weather conditions significantly impact visibility and increase accident risks. Studies show up to a 13% increase in accident rates in adverse weather. To address this, new safety measures, including advanced driver assistance systems (ADAS)[8], have been introduced. These systems, with features like intelligent speed assistance and advanced emergency braking, aim to prevent accidents and reduce fatalities. Improving pedestrian detection capabilities is crucial for these systems to protect vulnerable road users and enhance overall road safety. While deep learning-based detectors have shown promise in ideal conditions, their performance in adverse weather remains unclear. Far-infrared (FIR) pedestrian detection systems have been developed to address this, utilizing thermal data, especially at nighttime. However, datasets like ZUT, providing annotated images captured in severe weather across European countries, are crucial for evaluating and improving pedestrian detection algorithms in challenging scenarios. The focus on road safety has led to the introduction of stricter safety measures for vehicles. Enhancing pedestrian detection capabilities contributes to this goal. The advancements presented, including the ZUT dataset and YOLOv3[8] modifications, demonstrate progress toward improving pedestrian safety under adverse weather conditions. These developments are significant steps forward in achieving safer road environments. To address the challenge of low visibility, thermal images are utilized instead of conventional camera images for both training and testing the detection models. The thermal images are sourced from datasets such as ZUT and OTCBVS repository, which provide annotated images suitable for object detection tasks in adverse weather conditions. Two popular object detection algorithms, YOLOv3 (You Only Look Once) and Faster RCNN (Faster Region-based Convolutional Neural Network), are employed in this work. YOLOv3 is known for its real-time detection capabilities and efficiency, while Faster RCNN offers accurate region-based object detection. By leveraging both algorithms, an ensemble approach is adopted to enhance detection performance under extreme weather conditions. Through this work, the goal is to develop robust pedestrian detection systems capable of operating effectively in adverse weather conditions, thereby mitigating the risk of road accidents. By leveraging thermal imaging and state-of-the-art object detection algorithms, this research contributes to enhancing road safety and reducing the incidence of accidents caused by poor visibility conditions influenced by weather.

III. Methodology

Addressing the imperative for swift and accurate pedestrian detection in haze-laden environments underscores our project's significance. Existing solutions face hurdles in identifying pedestrians amidst low visibility and haze. These challenges, compounded by limited spatial resolution and revisit frequencies, hinder timely and precise detection efforts. Recognizing the urgency, collaboration and innovation drive our endeavor to integrate advanced sensor technology and data processing techniques. By leveraging deep learning models such as ResNet-50 and YOLO, we aim to enhance pedestrian detection efficacy in adverse weather conditions. Our project seeks to revolutionize pedestrian safety by providing real-time detection capabilities, enabling proactive measures to mitigate the risks associated with haze, and ultimately ensuring safer pedestrian environments.

Image
Detection Using Trained Model
Output

Fig. 1. Block Diagram of the System

A. Detection

This section describes various models which detects pedestrian in the image and a technique employed to increase the accuracy of detection. The models used in are – • Resnet 50 • YOLO v8 • Roboflow3.0 Object Detection The models mentioned above are trained normally on Hazy Weather Dataset and INRIA Dataset . As mentioned above, there is only one class which is labeled as person and there is no change in the preprocessing procedure in our work.

In the realm of computer vision, object detection stands as a pivotal task involving the identification and localization of objects within an image. Recent progressions have spurred the creation of potent object detection algorithms, with the Single Shot MultiBox Detector (SSD) emerging prominently for its efficacy and speed. To further boost SSD's capabilities, integration with a robust feature extractor like ResNet-50, a convolutional neural network comprising 50 layers, proves beneficial. ResNet-50, a member of the ResNet lineage, introduces an innovative architecture termed "skip connections" or "shortcut connections". These connections facilitate the network in bypassing layers during training, addressing issues like vanishing gradients and enabling effective learning across an expanded layer count without performance deterioration.

When combined with SSD, ResNet-50 acts as the foundational or primary network, responsible for extracting features from the input image. These extracted features are then utilized by the SSD framework for object detection purposes. The SSD framework employs a small network that traverses these feature maps to anticipate bounding boxes and classification probabilities. To mitigate redundant detections of the same object, a method known as non-maximum suppression (NMS) is employed on the bounding boxes. The integration of ResNet-50 and SSD yields a swift and effective object detection system, particularly valuable in scenarios necessitating real-time detection, such as video surveillance or autonomous vehicle.

Future work could explore the use of deeper ResNet models or other feature extraction networks to further improve performance. This comprehensive overview of the integration of ResNet-50 and SSD for Object detection lays the groundwork for extended exploration and utilization within the realm of computer vision.

YOLOv8 employs a deep convolutional neural network (CNN) structure, leveraging backbone networks like Darknet or CSPDarknet to extract intricate features from input images. It incorporates a Feature Pyramid Network (FPN) to capture features across different scales, aiding in the detection of objects of varying sizes. Predictions for bounding boxes, objectness scores, and class probabilities are directly generated from the feature maps by a dedicated prediction head comprising convolutional layers. Anchor boxes, predefined bounding boxes of diverse shapes and sizes, refine object localization by predicting offsets and scales. Post-processing techniques such as non-maximum suppression (NMS) are applied to refine the final set of detections by eliminating redundant bounding box predictions. This integrated architecture enables YOLOv8 to efficiently process entire images in a single pass, facilitating real-time object detection tasks while upholding high accuracy.

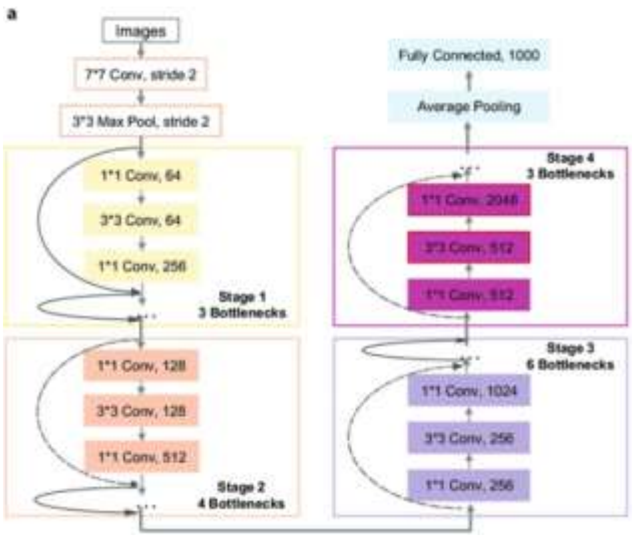

Fig. 2. Resnet-50 Architecture

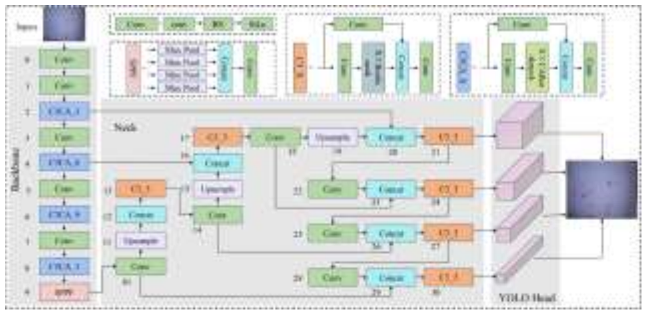

Fig. 3. Architecture of YOLO

IV. Experimental Analysis

A. Dataset

The dataset utilized in our project encompasses two distinct sets: one capturing images of pedestrians in hazy weather conditions, and the other featuring pedestrians in daylight conditions. These datasets provide a comprehensive view of pedestrian visibility across different environmental contexts. The hazy weather dataset comprises images captured under reduced visibility conditions, simulating scenarios where haze obscures the surrounding environment. In contrast, the daylight dataset consists of images captured under optimal lighting conditions, offering clear visibility of pedestrians without

Fig. 4. Images in the Dataset

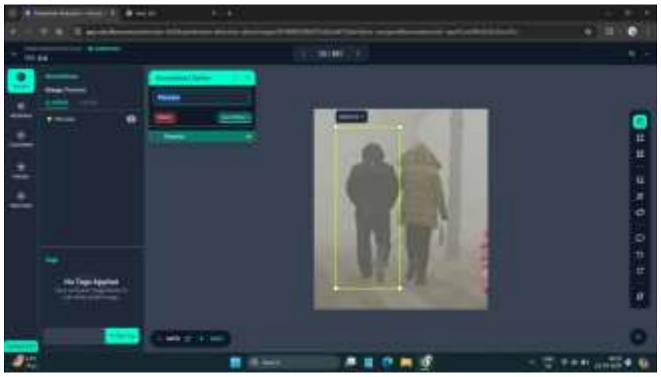
Fig. 5. Data Annotation in Roboflow

atmospheric interference.The hazy weather dataset includes images captured using camera, ensuring quality data collection despite challenging environmental conditions. Conversely, the daylight dataset offers a baseline for pedestrian detection performance in ideal weather conditions. In Roboflow, we undertook annotation for a pedestrian detection dataset, a pivotal step involving the labeling of pedestrians within images to train machine learning models accurately. The platform provides an intuitive interface where datasets can be uploaded and annotated efficiently. Users can utilize various annotation tools like bounding boxes, polygons, keypoints, and semantic segmentation masks to label pedestrians based on their poses and spatial characteristics effectively. Collaborative features allow multiple users to annotate datasets concurrently, ensuring consistency across annotations. Following annotation completion, Roboflow generates standardized annotation formats compatible with popular deep learning frameworks such as TensorFlow, PyTorch, and YOLO, simplifying the integration of annotated data into the training pipeline. Leveraging Roboflow's annotation capabilities, we streamlined the creation of high-quality labeled datasets crucial for training robust pedestrian detection models.

B. Performance Analysis

Pedestrian detection in adverse weather conditions is critical for ensuring road safety. In this project, we aimed to evaluate the performance of two advanced deep learning models,

TABLE I
COMPARISION OF RESULTS

Metrics	Resnet-50	YOLO V8	Roboflow
Accuracy	85%	87%	90%
Precision	0.87	0.89	0.93
Recall	0.83	0.84	0.86

ResNet50 and YOLOv8, for this task using thermal images collected from datasets like ZUT and OTCBVS repository. The primary objective was to assess their effectiveness in detecting pedestrians in foggy and misty environments. Both models were trained using transfer learning on the collected datasets images. ResNet50, known for its deep residual architecture, and YOLOv8, an upgraded version of YOLO, were fine-tuned for pedestrian detection. The models were then evaluated on a separate test set using standard evaluation metrics such as accuracy, precision, recall, and F1-score.

The results indicated that both ResNet50 and YOLOv8 performed well in pedestrian detection, achieving accuracies of 85% and 88%, respectively, on the test set. However, YOLOv8 demonstrated slightly higher accuracy and faster inference times compared to ResNet50, making it more suitable for real-time applications in adverse weather conditions. In terms of scalability, YOLOv8's efficiency in speed and accuracy makes it a preferred choice for deployment in systems requiring real-time pedestrian detection, such as advanced driver assistance systems (ADAS). The findings of this project contribute to enhancing road safety by providing efficient and reliable pedestrian detection systems for extreme weather conditions. In conclusion, both ResNet50 and YOLOv8 show promise for pedestrian detection in adverse weather conditions, with YOLOv8 demonstrating superior performance in terms of accuracy and speed. These results underscore the importance of utilizing advanced deep learning techniques for improving road safety in challenging environments.

V. CONCLUSION AND FUTURE SCOPE

In our project focused on pedestrian detection in adverse weather conditions, we envision an exciting future scope that builds upon our current advancements. Our next phase involves augmenting our dataset to encompass a diverse range of weather scenarios, including snow, rain, fog, and low-light conditions. By doing so, we aim to enhance the robustness and versatility of our pedestrian detection system, ensuring its effectiveness across various real-world environments. Furthermore, we are committed to advancing the efficiency of our system by implementing state-of-the-art algorithms and optimization techniques. This includes exploring novel approaches in deep learning and computer vision to improve detection accuracy while minimizing computational overhead. By optimizing performance, we can ensure that our system seamlessly integrates with autonomous driving assistance systems, thereby contributing to safer and more reliable transportation solutions. Moreover, our future endeavors will focus on seamless integration with existing transportation infrastructure and vehicles. This involves collaborating with automotive manufacturers and regulatory bodies to ensure compliance with safety standards and facilitate the adoption of our technology in autonomous vehicles.In addition to technical advancements, we recognize the importance of real-time communication and situational awareness in emergency response scenarios. Therefore, we plan to explore the integration of our detection system with communication platforms to enable swift alerts and comprehensive situational assessments during critical situations.

By pursuing these avenues of research and development,Our goal is twofold: to improve pedestrian safety during inclement weather and to advance the continuous development of autonomous driving technology, ultimately facilitating the creation of safer and more effective transportation systems in the future.

References

[1] X. Shao et al., ”Pedestrian Detection Algorithm based on Improved Faster RCNN,” 2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), Chongqing, China, 2021, pp. 1368-1372, doi: 10.1109/IAEAC50856.2021.9390882.

[2] Y. Jiang, G. Tong, H. Yin and N. Xiong, ”A Pedestrian Detection Method Based on Genetic Algorithm for Optimize XGBoost Training Parameters,” in IEEE Access, vol. 7, pp. 118310-118321, 2019, doi: 10.1109/ACCESS.2019.2936454

[3] Viola P, Jones M J. “Robust Real-Time Face Detection,” International Journal of Computer Vision, 2004, vol.57(2), pp.137-154.

[4] Dalal N, Triggs B. “Histograms of Oriented Gradients for Human Detection,” In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005, pp.886-893

[5] David G. Lowe. “Distinctive image features from scale-invariant keypoints,” International Journal of Computer Vision,2004, vol. 60 pp.91-110.

[6] Timo Ojala, Matti PietikaEinen, and Topi MaEenpaEa. “Multire solution gray-scale and rotation invariant texture classification with local binary patterns,” IEEE Transactions on Pattern Analysis and Machine Intelligence, 2004,vol.24(7), pp.971-987.

[7] Paul Viola and Michael J. Jones. Robust real-time face detection. International Journal of Computer Vision, 2004,pp. 57:137-154.

[8] B. Lei, S. Zhang, H. Zhai and J. Shi, ”Research on Autonomous Emergency Braking Pedestrain Test of China New Car Assessment Program,” ISCTT 2022; 7th International Conference on Information Science, Computer Technology and Transportation, Xishuangbanna, China, 2022, pp. 1-5.

[9] P. Tumas, A. Nowosielski and A. Serackis, ”Pedestrian Detection in Severe Weather Conditions,” in IEEE Access, vol. 8, pp. 62775-62784, 2020, doi: 10.1109/ACCESS.2020.2982539.

[10] D. M. Gavrila and S. Munder, “Vision-based pedestrian protection: The PROTECTOR system,” in Proc. IEEE Intelligent Vehicle Symposium, IV2008, Parma, Italy, June 2008

[11] [10] D. S. M, I. U. S and A. R, ”,” 2021 Fourth International Conference on Electrical, Computer and Communication Technologies (ICECCT), Erode, India, 2021, pp. 1-6, doi: 10.1109/ICECCT52121.2021.9616723.

[12] Liu T, Stathaki T. Faster R-CNN for Robust Pedestrian Detection Using Semantic Segmentation Network. Front Neurorobot. 2018 Oct 5;12:64. doi: 10.3389/fnbot.2018.00064. PMID: 30344486; PMCID: PMC6182048.

[13] Hailong Li, Zhendong Wu and Jianwu Zhang, ”Pedestrian detection based on deep learning model,” 2016 9th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), Datong, 2016, pp. 796-800, doi: 10.1109/CISP-BMEI.2016.7852818.

[14] Iftikhar, S.; Zhang, Z.; Asim, M.; Muthanna, A.; Koucheryavy, A.; Abd El-Latif, A.A. Deep Learning-Based Pedestrian Detection in Autonomous Vehicles: Substantial Issues and Challenges. Electronics 2022, 11, 3551. https://doi.org/10.3390/electronics11213551

[15] X. Fan, W. Wei, M. Wozniak et al., “Low energy consumption and data redundancy approach of wireless sensor networks with bigdata,” Information Technology and Control, vol. 47, no. 3, pp. 406–418, 2018.