VECHICLE COLLISION DETECTION AND ALERT SYSTEM USING YOLO

A PROJECT REPORT

Submitted by

M.NIHAL [RA1911003010319]

RAHUL KARAKA [RA1911003010340]

Under the Guidance of

Dr. K. PRADEEP MOHAN KUMAR

Assistant Professor, Department of Computing Technologies

In partial fulfilment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR – 603 203

MAY 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this 18CSP109L project report titled "**VEHICLE COLLISION DETECTION AND ALERT SYSTEM USING YOLO**" is the bonafide work of **Mr. M. NIHAL [Reg No. RA1911003010319]** and **Mr. RAHUL KARAKA [Reg No. RA1911003010340]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

DR. K. PRADEEP MOHAN KUMAR **SUPERVISOR**	**DR. K. PRADEEP MOHAN KUMAR** **PANEL HEAD**
Assistant Professor Department of Computing Technologies	Associate Professor Department of Computing Technologies
	DR. M. PUSHPALATHA **HEAD OF THE DEPARTMENT** Department of Computing Technologies
INTERNAL EXAMINER	**EXTERNAL EXAMINER**

Department of Computing Technologies

SRM Institute of Science and Technology

Own Work Declaration Form

Degree/ Course	: B.Tech in Computer Science and Engineering
Student Names	: M. Nihal, Rahul Karaka
Registration Number	: RA1911003010319, RA1911003010340
Title of Work	: Vehicle Collision Detection and Alert System using YOLO

We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.

We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc.)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION
We are aware of and understand the University's policy on Academic misconduct and plagiarism and we certify that this assessment is our own work, except were indicated by referring, and that we have followed the good academic practices noted above. **Student 1 Signature:** **Student 2 Signature:** **Date:**
M. Nihal [RA1911003010319] **Rahul Karaka [RA1911003010340]**

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr. T. V. Gopal**, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the projectwork.

We are incredibly grateful to our Head of the Department, **Dr. M. Pushpalata,** Professor, Department of Computing Technologies, SRM Institute of Science and Technology,for her suggestions and encouragement at all the stages of the project work.

We convey our thanks to our Project Panel Head, **Dr. K. Pradeep Mohan Kumar**, Associate Professor and members, **Dr. G. K. Sandhia ,** Assistant Professor and **Mrs. J. Ramaprabha,** Assistant Professor, **Ms. H. Hema**, Assistant Professor, from Department of Computing Technologies, SRM Institute of Science and Technology, for their inputs during the project reviews and support.

Our inexpressible respect and thanks to our guide, **Dr. K. Pradeep Mohan Kumar**, Associate Professor, Department of Computing Technologies, SRM Institute of Science and Technology,for providing us with an opportunity to pursue our project under her mentorship. She provided us with the freedom and support to explore the research topics of our interest. Her passion forsolving problems and making a difference in the world has always been inspiring.

We register our immeasurable thanks to our Faculty Advisors, **Dr. M. Kanchana**, Associate Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for leading and helping us to complete our course.

We sincerely thank the Computing Technologies Department staff and students, SRM Institute of Science and Technology, for their help during our project. Finally, we would like to thank parents, family members, and friends for their unconditional love, constant support, and encouragement.

M. Nihal [RA1911003010319]

Rahul Karaka [RA1911003010340]

TABLE OF CONTENTS

ABSTRACT

According to global statistics, car accidents are the leading cause of violent deaths. Becauseof this, as well as the widespread usage of video surveillance and intelligent traffic systems, an automated traffic accident detection strategy is becoming more appealing to computer vision researchers. Automatic traffic accident detection (ATAD) based on video has emerged as one of the most promising applications in intelligent transport in recent years, playing an increasingly essential role in maintaining travel safety. Input is a video obtained via surveillance systems. Output results are acquired instantly in real-time and we would benotified if there's a chance for collision or not. The existing systems are simple and effectivebut are only able to analyze the rear portion of the vehicle. Hence, not very effective and have low accuracy. The goal of our project is to detect accidents and vehicle collisions in order to save lives of unattended people involved in the fatal crash. This can even further beapplied to self-driving cars for lesser accidents avoidance in future.

We have incorporated models like YOLO and deep learning techniques-image processing for better accuracy.

LIST OF SYMBOLS AND ABBREVIATIONS

AI	Artificial Intelligence
DL CNN	Deep Learning Conventional Neutral Network
YOLO	You Only Look Once
Convo2D	Conventional 2D networks
ML	Machine Learning
DS	Data Science
RCNN	Regions with convolutional Neutral Network
JSON	Javascript Object Notation
CCTV	Closed-Circuit Television
GPU	Gated Recurrent Unit
CV	Computer Vision
LSTM	Long Short Term Memory
ResNET	Residual Networks

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 General

Vehicle collision detection and alert systems using YOLO are an exciting and rapidly evolving field that combines a range of technologies to improve transportation safety and efficiency. These systems can help drivers avoid accidents by using advanced algorithms to analyze data in real-time, including video feeds from cameras and sensors, GPS data, and other relevant information.

One of the primary benefits of these systems is their ability to detect and alert drivers to potential hazards that may not be immediately visible, such as pedestrians or cyclists in blind spots. By using machine learning algorithms to analyze video feeds and other data sources, these systems can identify potential hazards and alert drivers to take corrective action.

In addition to improving safety, vehicle collision detection and alert systems can also help reduce the environmental impact of transportation by optimizing vehicle routes and reducing congestion. By analyzing data on traffic patterns, weather conditions, and other factors, these systems can help drivers find the most efficient and environmentally friendly routes to their destinations.

Another important benefit of vehicle collision detection and alert systems using YOLO is their ability to improve the overall efficiency of transportation systems. By analyzing data on traffic patterns, weather conditions, and other factors, these systems can optimize vehicle routes, reduce congestion, and help drivers find the fastest and most environmentally friendly routes to their destinations.

As the technology continues to evolve, vehicle collision detection and alert systems using YOLO are likely to become even more sophisticated and effective. In the future, it may be possible for these systems to communicate directly with other vehicles on the road, enabling them to coordinate their movements and avoid collisions in real-time.

In addition to improving safety and efficiency, vehicle collision detection and alert systems using YOLO also have the potential to reduce costs associated with accidents. According to the National Highway Traffic Safety Administration, the economic cost of motor vehicle crashes in the United States was $242 billion in 2010. This includes costs associated with medical expenses, lost productivity, property damage, and legal and insurance fees. By preventing accidents before they occur, vehicle collision detection and alert systems can help reduce these costs and save lives.

Moreover, these systems can also be used to improve the performance and safety of autonomous vehicles. As self-driving cars become more common, it will become increasingly important to develop reliable and accurate collision detection and alert systems. These systems can help ensure the safety of passengers and other road users by identifying potential hazards and taking corrective action in real-time.

Despite the many benefits of these systems, however, there are also some challenges to be addressed. For example, the accuracy of the systems may be affected by factors such as weather conditions or the quality of the video feeds. Additionally, there are also important ethical and legal considerations to be addressed, such as the potential impact on privacy and the liability of the systems in the event of an accident.

In conclusion, vehicle collision detection and alert systems using YOLO have the potential to transform transportation safety and efficiency. By combining the most recent advancements in computer vision, machine learning, and data analytics these systems can help drivers avoid accidents, reduce congestion, and make our roads safer and more sustainable. However, as with any emerging technology. It is critical to thoroughly assess the potential consequences, risks and benefits as well as to handle any ethical or legal issues that may arise.

1.2 Purpose

The purpose of vehicle collision detection and alert systems using YOLO is to improve transportation safety and efficiency by leveraging advanced technologies to detect and prevent collisions between vehicles and other obstacles on the road. These systems are designed to provide real-time insights into the driving environment, alert drivers to potential hazards, and take preventive measures to avoid collisions.

One of the primary goals of these systems is to enhance the safety of drivers and passengers by identifying potential hazards and alerting them to take corrective action. By using data from multiple sources, such as sensors, cameras, and GPS devices, these systems can detect hazards that may not be immediately visible, such as pedestrians or cyclists in blind spots.

In addition to improving safety, vehicle collision detection and alert systems an also help reduce the environmental impact of transportation by optimizing vehicle routes and reducing congestion. By analyzing data on traffic patterns, weather conditions, and other factors, these systems can help drivers find the most efficient and environmentally friendly routes to their destinations.

Furthermore, these systems have the potential to reduce the economic costs associated with accidents by preventing accidents before they occur. According to the National Highway Traffic Safety Administration, car accidents cost the most. US economy $242 billion in 2010, including costs associated with medical expenses, lost productivity, property damage, and legal and insurance fees. By preventing accidents, vehicle collision detection and alert systems can help reduce these costs and save lives.

Overall, the purpose of vehicle collision detection and alert systems using YOLO is to make our roads safer, more efficient, and more sustainable by leveraging advanced technologies and data analytics to prevent accidents and improve transportation safety.

1.3 Scope

The scope for vehicle collision detection and alert systems using YOLO is significant and rapidly expanding. As the adoption of connected and autonomous vehicles continues to grow, the demand for advanced collision detection and alert systems is expected to increase as well.

One area where these systems have significant potential is in improving the safety and efficiency of commercial vehicle fleets. Fleet managers can use these systems to monitor their vehicles in real-time, identify potential hazards, and take corrective action to prevent accidents. By using data analytics to identify patterns and trends in driving behavior, these systems can also help fleet managers develop more effective safety training programs and optimize their fleets for maximum efficiency.

Another area where these systems can have a significant impact is in the development of autonomous vehicles. As self-driving cars become more common, it will become increasingly important to develop reliable and accurate collision detection and alert systems. These systems will be critical in ensuring the safety of passengers and other road users by identifying potential hazards and taking corrective action in real-time.

Moreover, these systems can also be used to improve the safety and efficiency of public transportation systems. By analyzing data on traffic patterns and accident rates, these systems can help identify areas of the road network that are particularly prone to accidents or congestion. This information can be used to develop more effective road safety and congestion management strategies, improving the overall efficiency and safety of transportation systems.

In addition, vehicle collision detection and alert systems can also be used to improve the safety of vulnerable road users such as pedestrians and cyclists. By using advanced sensors and data analytics, these systems can detect hazards that may not be immediately visible to drivers, alerting them to take corrective action and preventing accidents.

Overall, the scope for vehicle collision detection and alert systems using YOLO is broad and multifaceted. These systems have the ability to make better safety, competence, sustainability across a wide range of transportation sectors, ultimately making our roads safer and more sustainable for everyone.

1.4 Machine Learning and Deep Learning for Vehicle collision and alert system

Machine learning (ML) is a subset of artificial intelligence (AI) that entails employing algorithms to analyse data, learn from that data, and apply what has been learned. then use what they've learned to make judgments. ML models can be used for a variety of tasks, such as predicting customer behavior, detecting fraud, and recommending products. ML algorithms can be trained on different types of data, including structured data (e.g., data in a spreadsheet) and unstructured data (e.g., text, images, and audio). The algorithms can be supervised (training data that has been labelled with the right outputs) or unsupervised (training data that has not been labelled). ML models improve over time as they are exposed to more data and learn from it.

An example of a machine learning algorithm is the on-the-spot song player, which uses collaborative filtering to suggest new songs or artists to a listener based on their musical preferences and the preferences of other listeners who share their musical taste. Many other automated recommendation systems use this method, which is often referred to as AI.

Deep learning (DL) is a subtype of machine learning layering algorithms to create artificial neural networks that can learn and make decisions on their own. DL models are designed to continuously learn from data, much like a human would. They can analyze and identify patterns in large sets of unstructured data, such as images, audio, and text, and use that information to make decisions. DL models are particularly powerful for tasks that require high levels of accuracy, image and speech recognition, natural language processing, and self-driving automobiles are a few examples. DL models are made up of multiple layers of algorithms, each of which processes a different aspect of the data. The layers are interconnected and can learn from one another, which allows the model to make increasingly complex decisions over time. DL models are typically trained on large datasets and require significant computational power to run. However, once they are trained, they can make accurate predictions with little to no human intervention.Machine learning is the process of analyzing data and making judgments based on that data using algorithms. Deep learning is a subset of machine learning layering algorithms to create artificial neural networks that can learn and make decisions on their own. Deep learning models are particularly powerful for tasks that require high levels of accuracy and can analyze and identify patterns in large sets of unstructured data.

There are several methodologies for developing a Deep learning is used in a car crash and alarm system.Here are some of the most common approaches:

Object detection is a popular approach in computer vision that includes finding things within an image or video stream. Object detection can be employed in the context of a collision and alarm system to recognise other cars, pedestrians, and road hazards. Object detection approaches based on deep learning, such as the YOLO (You Only Look Once) and Faster R- CNN (Region-based Convolutional Neural Network) algorithms, can be used to recognise and track objects in real-time.

Lane detection: Lane detection is another important aspect of a collision and alert system. This involves detecting and determining the vehicle's position within the lane by tracking the lanes on the road. Lane identification approaches based on deep learning, such as the Hough Transform and the Canny Edge identification algorithm, may be used to recognise and track lanes in real-time.

Image segmentation picture segmentation is the process of splitting a picture into many segments, each corresponding to a different item or location. In the context of a collision and alert system, image segmentation can be used to identify different regions of the road, such as the road surface, sidewalks, and other obstacles. Deep learning-based image segmentation techniques, such as the U-Net and Mask R-CNN algorithms, can be used to segment images and identify different regions of the road.

Motion detection: Motion detection is the technique of detecting changes in the location of an object over time. In the context of a collision and alert system, motion detection can be used to identify the movement of other vehicles, pedestrians, and obstacles on the road. Deep learning-based motion detection techniques, such as the optical flow algorithm, can be used to detect motion in real-time.

Once these techniques have been implemented, they can be used to trigger alerts and warnings to the driver when a collision is imminent. For example, if the system detects that the car is leaving its lane, it might send an auditory warning. to alert the driver. Similarly, if the system detects that another vehicle is approaching too closely, it can issue a visible or aural indication to the driver to take evasive action.

Moreover, machine learning can also be used to develop predictive models that identify areas of the road network that are particularly prone to accidents. By analyzing historical data on traffic patterns and accident rates, these models can identify patterns and trends that indicate high-risk areas, allowing transportation authorities to develop targeted interventions to improve safety.

Overall, machine learning and deep learning techniques offer significant potential for improving the accuracy and effectiveness of vehicle collision detection and alert systems, ultimately helping to reduce the number of accidents on our roads.

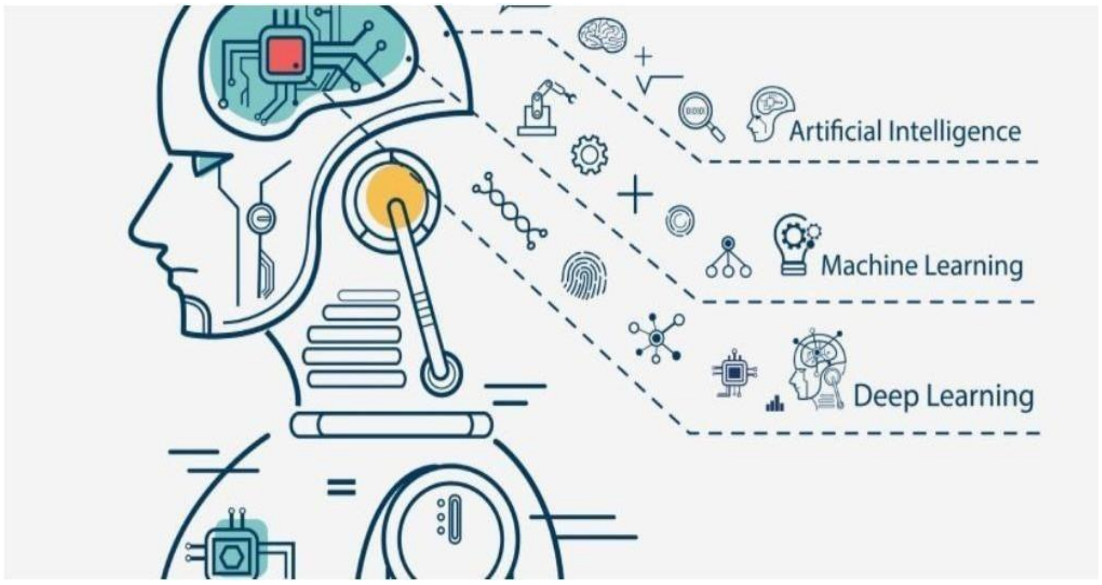
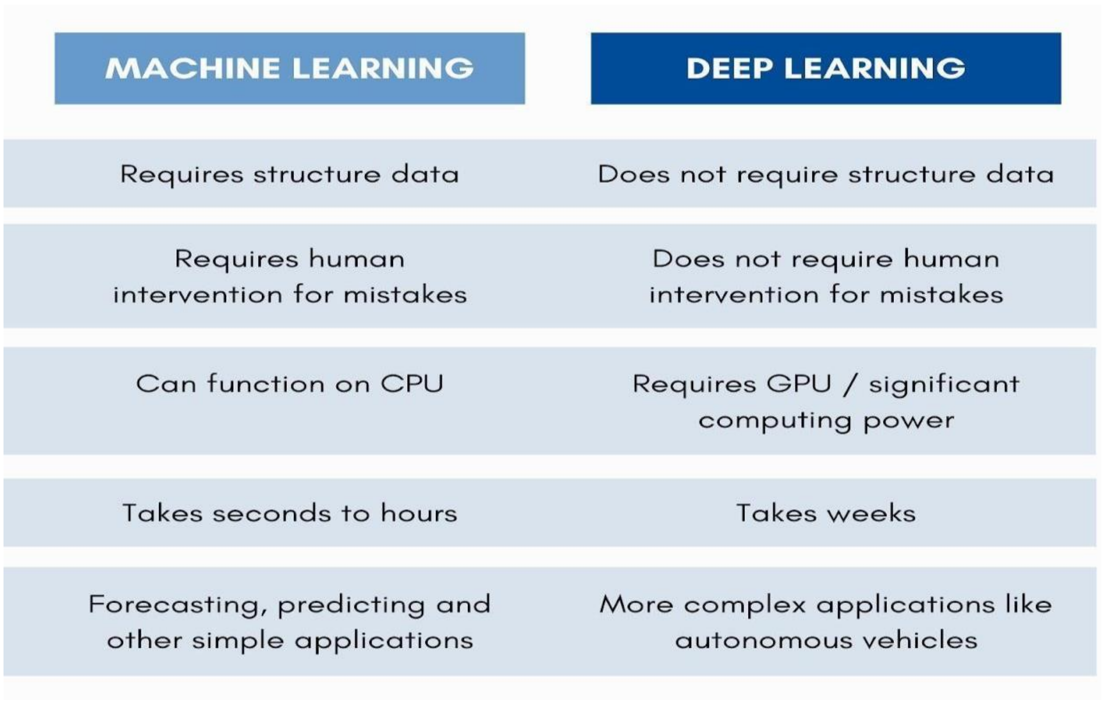


Figure 1.1 Relationship Between AI, ML and DL



MACHINE LEARNING	DEEP LEARNING
Requires structure data	Does not require structure data
Requires human intervention for mistakes	Does not require human intervention for mistakes
Can function on CPU	Requires GPU / significant computing power
Takes seconds to hours	Takes weeks
Forecasting, predicting and other simple applications	More complex applications like autonomous vehicles

Table 1.1 Difference between Machine Learning and Deep Learning

CHAPTER 2

LITERATURE SURVEY

Vehicle collision detection and alert systems have been In recent years, there has been a great deal of study, with many approaches and strategies being offered. Deep learning methods such as You Only Look Once (YOLO), CNN (Convolutional Neural Networks), and others are popular for recognising objects in pictures and videos. OpenCV, etc’

A literature survey on vehicle collision detection and alert systems using YOLO reveals the following studies:

2.1Deep Learning for Collision Priority Estimation in Traffic Videos:

Presented at the previous 2020 IEEE conference describes a Object identification and heuristic vehicle collision estimate system based on deep learning that uses Recorded traffic videos by vehicle dashboard cameras as a primary method. In the input videos, the created collision estimate technique performs quite well. To improve the accuracy of the collision prediction, the road feature semantics such as junctions, lane changes, turnings, signboards, and so on can be incorporated, allowing for full analysis and knowledge of the traffic situation. A conceivable Advanced Driver Assistance Systems (ADAS) utility is the system for collision warning and navigation route recommendations. Based on qualitative factors, the suggested method estimates collision priority. The addition of quantitative measurements of spatiotemporal factors such as vehicle motion direction, velocity, and so on will result in a more robust assessment. It is possible to make the estimate's accuracy reliable and dependable by combining it the driving scenario and the mechanics of the vehicle with additional communication channels, Light detection and Ranging (LIDAR), Inertial Measurement Unit (IMU), and other technologies, and so on.

2.2 Enhancing images in real time for autonomous car accidents:

Detection through CCTV using deep learning" published in springer journal in 2021 they have successfully used a statistically verifiable and accurate automated system for detecting accidents, which may be utilised in settings demanding instantaneous feedback. Mini-YOLO is the use of knowledge distillation in deep learning architecture classes that has a spectacular runtime performance and a high AP rating when against other detection techniques. They tested several algorithms for accident classification and discovered that a radial basis kernel support vector machine works most effective in memory demand latency, too.

It is recommended that comparable work be done using a deep learning-based model employing SENet blocks in the ResNext architecture. The performance of the model is compared to that of well-known deep learning models such as ResNet50, VGG16, and VGG19. The proposed model outperforms the baseline methods currently in use by consuming a significantly lower percentage of the GTACrash simulated data for training and achieving a ROC-AUC of 0.91. This reduces the computational load.

YOLO has been used in study for object and accident detection. Install a real- time, autonomous system for detecting accidents using the least amount of hardware possible. For the detection stage, we propose a deep learning model architecture termed Mini-YOLO based on information distillation. Its precision is equivalent to that of its predecessor.

With minimal computational cost and model size, You-Only-Look-Once (YOLO) can be achieved. Mini-YOLO exceeds every other detection technique in terms of runtime complexity on the MS-COCO dataset, averaging on a low-

end, amazing 28 frames per second machine. This yields an average accuracy (AP) score of 34.2.

2.3 Estimation of Collision Priority on Traffic Videos using Deep Learning

This paper describes a brand-new heuristic unimodal strategy based on the visual system for predicting the collision priority of automobiles on the road. Priorities are determined from the perspective of an ego automobile, which might be outfitted with either a fully autonomous vehicle or a vision-based driving aid system. The proposed qualitative method works well for input films, and by integrating it with other quantitative interpretations of traffic data, a more trustworthy estimate may be achieved.

Overall, these investigations show that adopting YOLO for automobile accident detection is beneficial. and alert systems and highlight the potential of it

CHAPTER 3

SYSTEM ARCHITECTURE AND DESIGN

3.1 Motivation

The motivation for vehicle collision and alert systems using YOLO is primarily to improve the safety of drivers, passengers, and other road users. Every year, millions of people are injured or killed in vehicle collisions worldwide, and the economic and social costs of these accidents are significant.

By developing more advanced and effective collision detection and alert systems using YOLO, we can help prevent accidents and reduce the number of injuries and fatalities on our roads. These systems can provide drivers with real-time alerts to potential hazards, allowing them to take corrective action before an accident occurs.

Moreover, vehicle collision and alert systems can help improve the efficiency and sustainability of transportation systems. By identifying areas of the road network that are particularly prone to accidents or congestion, transportation authorities can develop targeted interventions to improve safety and reduce traffic. This can help reduce travel times, lower fuel consumption, and reduce emissions, ultimately contributing to a more sustainable transportation system.

Another motivation for developing vehicle collision and alert systems using YOLO is to support the development of autonomous vehicles. As self-driving cars become more common, it will become increasingly important to develop reliable and accurate collision detection and alert systems. These systems will be critical in ensuring the safety of passengers and other road users by identifying potential hazards and taking corrective action in real-time.

Overall, the motivation for vehicle collision and alert systems using YOLO is to make our roads safer, more efficient, and more sustainable for everyone. By harnessing the power of YOLO and advanced analytics, we can develop systems that help prevent accidents and improve the overall safety and sustainability of our transportation systems.

3.2 Objectives

The primary objective of developing vehicle collision and alert systems using YOLO is to improve the safety of drivers, passengers, and other road users. More specifically, the objectives of such systems include:

Collision detection: The system should be able to accurately detect potential collisions in real-time, using sensors such as cameras, lidar, or radar. This involves developing algorithms that can analyze sensor data and identify potential hazards, such as other vehicles, pedestrians, or obstacles in the road.

Early warning: The system should provide drivers with real-time alerts to potential hazards, allowing them to take corrective action before an accident occurs. This involves developing algorithms that can analyze sensor data and identify patterns that indicate potential hazards, as well as developing user interfaces that can effectively communicate these alerts to drivers.

Overall, the objective of developing vehicle collision and alert systems using YOLO is to develop devices capable of detecting possible risks in real time with accuracy and reliability provide early warning to drivers, and ultimately help prevent accidents and improve the safety of our roads.

3.3 Block Diagram

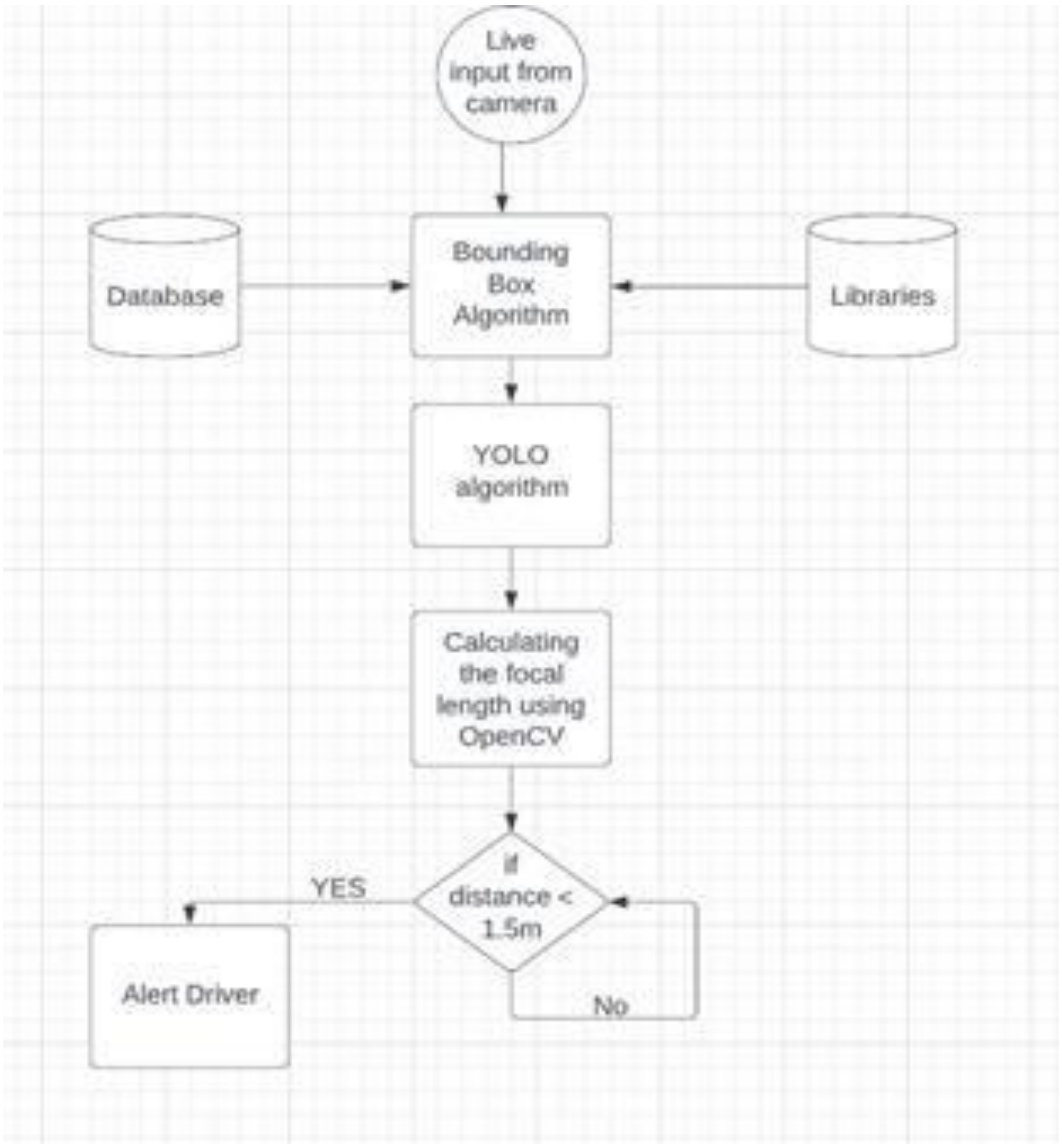


fig 3.3.1 Block diagram for dashboard cameras

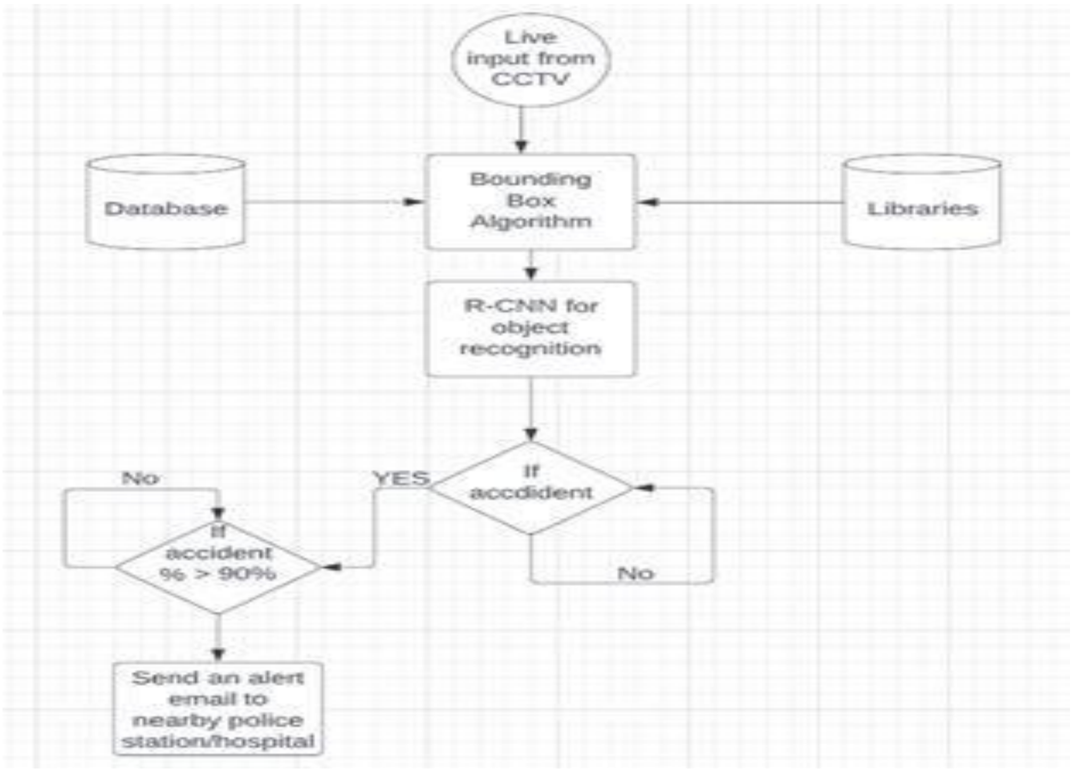


fig 3.3.2 Block Diagram for CCTV cameras

Our thought-of architecture constitutes of the below mentioned components –

- Data Pre-processing
- Data Cleaning
- labeling of quality datasets
- images/videos being assigned
- bounding Box Prediction

- YOLO algorithm
- Alerting the driver (yes/no)

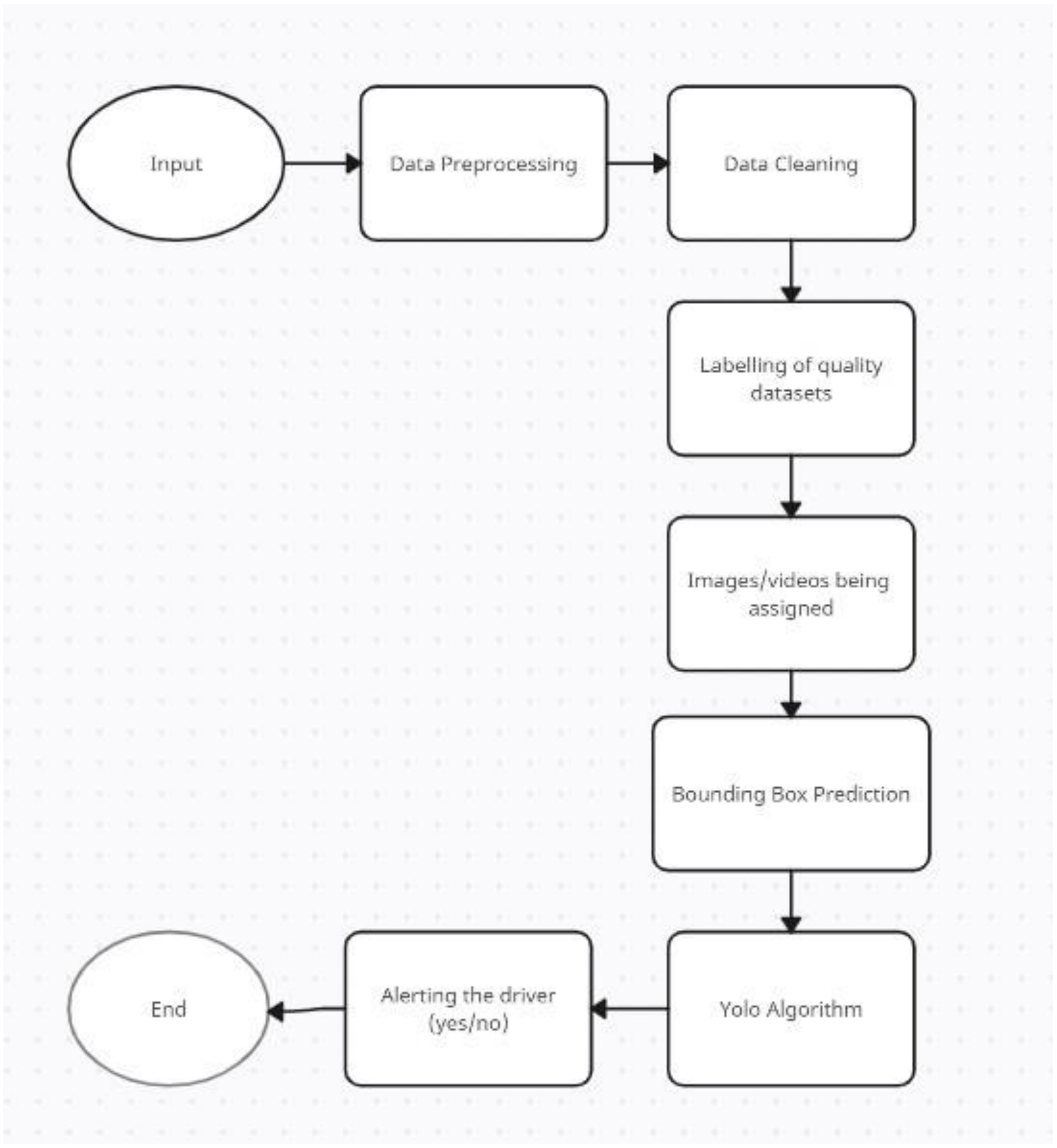

Input
Data Preprocessing
Data Cleaning
Labelling of quality datasets
Images/videos being assigned
Bounding Box Prediction
Yolo Algorithm
Alerting the driver (yes/no)
End

Figure 3.3 Activity Diagram of the Proposed Model

3.4 Hardware Specifications

Table 3.4 Hardware Specifications

Hardware	**Specification**
Operating System	Windows / macOS
CPU	Single Core and above
	Frequency: 2.2 GHz
Memory	2 GB and above
Hard Disk	30 GB and above

CHAPTER 4

METHODOLOGY

4.1 Existing Method

There are several existing methods for deep learning-based car collision detection and alarm systems. Here are some examples:

Tesla Autopilot: Tesla's Autopilot system deep learning is used to identify and analyse the vehicle's surroundings. The system uses cameras, radar, and ultrasonic sensors to Detect and monitor other cars, pedestrians, and road hazards. Lane departure alerts, automated emergency braking, and adaptive cruise control are among the features.

Honda Sensing: Honda Sensing is an ADAS system Deep learning is used to identify and analyse the vehicle's surroundings. The system detects and tracks other cars, pedestrians, and road obstructions using cameras and radar. Lane departure alerts, forward collision warnings, and automated emergency braking are among the features.

Waymo: Waymo is a self-driving technology company that uses deep learning to detect and analyze the vehicle's surroundings. The system uses a combination of cameras, lidar, and radar to detect and track other vehicles, pedestrians, and obstacles on the road. It includes features such as lane departure warnings, forward collision warnings, and pedestrian detection.

These systems use a combination of object detection, image segmentation, motion detection, and other deep learning-based techniques to detect and analyze the vehicle's surroundings in real-time. They can issue alerts and warnings to the driver when a collision is imminent, or take evasive action automatically to prevent a collision.

4.2 Dataset Description

The dataset contains 700 images of both accident and No accident images . The images have undergone pre-processing techniques to enhance their quality, which could involve techniques like adjusting brightness and contrast, removing noise, or sharpening the image. The images have also been segmented, which means they have been divided into separate regions or objects, perhaps to isolate areas of interest or remove background noise.

In addition to pre-processing, the data has also been augmented, which typically involves creating new data by modifying existing data in some way. Augmentation techniques might include rotating, flipping, or cropping the images, or adding artificial noise or distortion to the images.

The goal of all these pre-processing and augmentation techniques is to improve the quality of the dataset, which in turn will hopefully lead to more accurate and reliable classification models. The paragraph states that two different classification models will be used: a CNN (Convolutional Neural Network). Both of these are machine learning algorithms that can be taught to classify images based on their features. The text implies that the project's purpose is to produce a cost-effective solution. method for predicting accident.

Overall, this project involves creating a dataset of pre-processed images, augmenting the dataset to improve its quality, and using two different machine learning algorithms to classify the images and predict vehicle collision

4.3 Data Preprocessing

Data preprocessing: Once the dataset has been collected, the images or videos need to be preprocessed to extract the relevant features. This can include techniques such as image resizing, normalization, and augmentation.

CNN architecture design: The next step is to design a CNN architecture that can be trained on the dataset. This can include designing the number and size of convolutional and pooling layers, as well as the number and size of fully linked layers.

Training the CNN: The CNN is trained on the preprocessed dataset using backpropagation and gradient descent. The goal is to optimize the CNN's parameters to minimize the classification error.

Testing the CNN: Once the CNN has been trained, it can be tested on a separate dataset to assess its performance. Accuracy, precision, recall, and F1 score are examples of performance measurements.

Integrating the CNN into the collision and alert system: Once the CNN has been trained and tested, it can be integrated into the collision and alert system. The CNN can be used in real-time to detect and classify objects, and issue alerts and warnings to the driver when a collision is imminent.

Overall, a CNN-based vehicle collision detection and alert system can provide real-time object detection and classification capabilities, and can help prevent collisions and improve driver safety.

4.4 Bounding Box Object Detection

A bounding box is a rectangular structure that encompasses all the significant features of a specific object in an image. It is a quick and straightforward technique used for image annotation, where the annotator draws a box around the object based on the project requirements.

The primary goal of using bounding boxes is to narrow down the search range for object features, which conserves computing resources. This technique not only aids in object classification but also assists in object detection.

To perform object detection, bounding boxes are typically drawn around objects in the image, which contain information about the object and coordinates that indicate its location in the image. Hence, object detection is a combination of object classification and localization.

4.5 YOLO

The Yolo method works by applying a neural network to a picture. The picture is segmented into a SxS grid and given a bounding box. This technique consists of 24 convolutional layers, followed by two fully linked layers. Alternating 1x1 convolutional layers from prior layers reduces feature space. The object identification issue is a regression problem with the goal of geographically bounding box separation and the likelihood of related classes in the bounding boxes. In just one evaluation, a single neural network can predict the bounding boxes and class probabilities directly from the input images.

The main advantage of YOLO is that it is fast and can perform object detection in real-time, making it well-suited for accident detection applications where time is critical. Here are the basic steps involved in using YOLO for accident detection:

Dataset Collection: Collect a dataset of images or videos that contain examples of accidents. The dataset should be large enough and diverse enough to cover different types of accidents and scenarios.

Annotation: Annotate the dataset with bounding boxes around the relevant objects in the images or videos. In the case of accident detection, the relevant objects could include vehicles, pedestrians, and other potential hazards.

Training: Train the YOLO model on the annotated dataset Using a deep learning framework such as TensorFlow or PyTorch. During training, the model learns to recognise important items in input photos or videos and forecast their bounding boxes.
Testing: Test the trained YOLO model on a new set of images or videos to evaluate its accuracy and performance. In the case of accident detection, the model should be able to accurately identify and localize potential hazards in real-time.

Integration: Once the YOLO model is trained and tested, it can be integrated into an accident detection system. The system can be designed to trigger an alert or warning when potential hazards are detected in real-time.

Overall, YOLO can be an effective tool for accident detection, as it is fast, accurate, and can be easily integrated into existing systems. However, as with any deep learning algorithm, the quality of the results depends on the quality and diversity of the training data, and the specific requirements of the application should be carefully considered during the design and implementation of the system.

4.6 OpenCV

OpenCV (Open Source Computer Vision Library) is a free and open source software library for computer vision and machine learning. OpenCV was created to offer a standard foundation for computer vision applications and to speed up the incorporation of machine perception into commercial goods. It is primarily concerned with image processing, video recording, and analysis, including features like as face detection and object detection. In our project OpenCV helps in detecting the car accidents through real-time CCTV camera and detecting the distance between 2 vehicles through front and rear cameras of the vehicle there are some of the basic steps involved in using OpenCV for accident detection:

Dataset Collection: Collect a dataset of images or videos that contain examples of accidents. The dataset should be large enough and diverse enough to cover different types of accidents and scenarios.

Preprocessing: Preprocess the images or videos to enhance their quality and reduce noise. This may involve techniques such as color correction, contrast adjustment, and noise reduction.

Object Detection: Use OpenCV's object detection functions to identify potential hazards in the input images or videos. Object detection algorithms can be trained on the annotated dataset to learn to recognize relevant objects, such as vehicles, pedestrians, and other potential hazards.

Tracking: Once potential hazards have been detected, use OpenCV's tracking functions to track their movements over time. This can provide valuable information about the trajectory of the hazards and their potential impact.

Alert Generation: Once potential hazards have been detected and tracked, generate an alert or warning if the system detects an imminent collision or potential accident.

Overall, OpenCV can be a powerful tool for accident detection, because it offers a diverse set of functions and algorithms for image and video processing. However, The quality of the output is determined by the quality of the incoming data, the specific requirements of the application, and the design and implementation of the system.

4.7 CNN

Convolutional neural networks (CNNs) are a sort of deep learning algorithm that is extremely strong which are commonly used for computer vision tasks, such as image and video analysis. CNNs are particularly well-suited for vehicle collision detection and alert systems, as they can be used to detect and classify objects in real-time.

CNN architecture design: The next step is to design a CNN architecture that can be trained on the dataset. This can include designing the number and size of convolutional and pooling layers, as well as the number and size of fully linked layers.

Training the CNN: The CNN is trained on the preprocessed dataset using backpropagation and gradient descent. The goal is to optimize the CNN's parameters to minimize the classification error.

Testing the CNN: Once the CNN has been trained, it can be tested on a separate dataset to evaluate its performance. The performance metrics can include accuracy, precision, recall, and F1 score.

Integrating the CNN into the collision and alert system: Once the CNN has been trained and tested, it can be integrated into the collision and alert system. The CNN can be used in real-time to detect and classify objects, and a issue alerts and warnings to the driver when a collision is imminent.

Overall, a CNN-based vehicle collision detection and alert system can provide real-time object detection and classification capabilities, and can help prevent collisions and improve driver safety.

A Convolutional Neural Network (CNN) can be a powerful tool for accident detection in various scenarios, such as in transportation or workplace safety. Here are the steps you can follow to build a CNN for accident detection:

Collect a dataset of images or videos that include examples of both accidents and non-accidents. The more diverse and representative the dataset, the better.

To prepare the dataset for training a Convolutional Neural Network (CNN), it is essential to split the data into three parts: training, validation, and testing sets. The training set is used to train the CNN, the validation set helps tune the hyperparametersand evaluate the CNN's performance during training, while the testing set evaluates the final performance of the CNN.

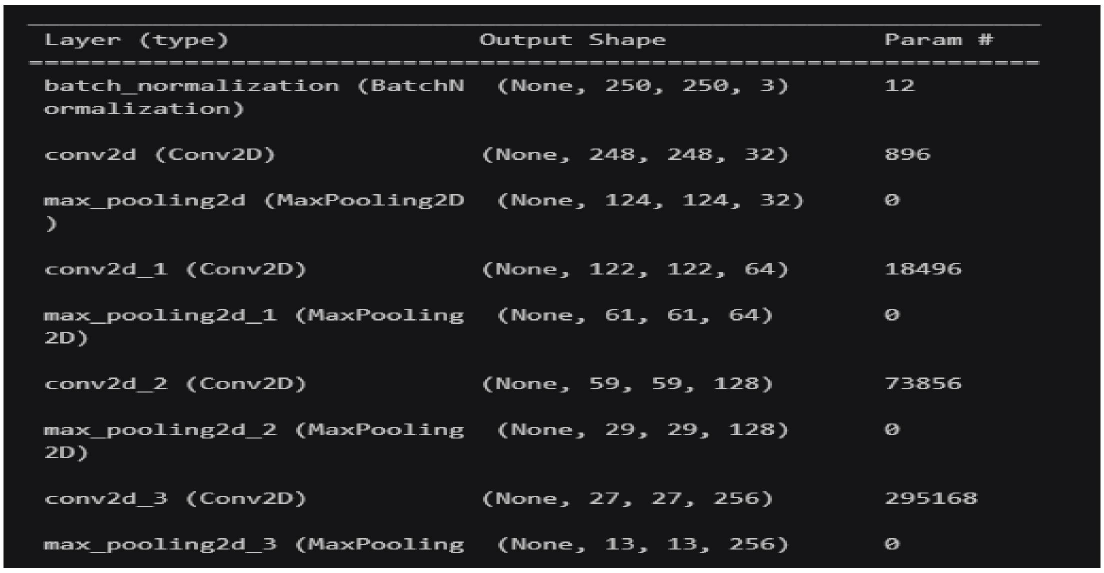
Before training the CNN, it is necessary to preprocess the data by resizing the images to a fixed size, converting them to grayscale or RGB, and normalizing the pixel values to a range between 0 and 1.

To design the CNN architecture, you need to determine the number of convolutional layers, pooling layers, and fully connected layers. Experimentation with different architectures can help you identify the best-performing one on your dataset.

During the training process, it is crucial to monitor the validation accuracy to prevent overfitting. Techniques such as dropout or data augmentation can be employed to avoid overfitting.

Once the CNN is trained, its performance is evaluated using the testing set, and metrics such as accuracy, precision, recall, and F1-score can be calculated.You can also visualize the CNN's predictions and the learned features to gain insights into how it is detecting accidents.Fine-tune the CNN or train it on additional data if the performance is not satisfactory.

In summary, building a CNN for accident detection involves collecting and preprocessing data, designing the architecture, training the model, and evaluating its performance.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 batch_normalization (BatchN  (None, 250, 250, 3)      12
 ormalization)

 conv2d (Conv2D)             (None, 248, 248, 32)      896

 max_pooling2d (MaxPooling2D  (None, 124, 124, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 122, 122, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 61, 61, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 59, 59, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 29, 29, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 27, 27, 256)       295168

 max_pooling2d_3 (MaxPooling  (None, 13, 13, 256)      0
```

fig 4.7.1 CNN Model

4.8 Validation gain vs Validation loss

In vehicle collision detection and alert systems using deep learning, validation data and validation loss are crucial components for evaluating and improving the performance of the model.

Validation data is a subset of data that is not utilised during training but is used to evaluate the model's performance during training. The goal of validation data is to assess how effectively a model will generalise to new data which it has not seen before. Typically, validation data is randomly sampled from the dataset, and it's important that it is representative of the dataset.Validation loss is a metric that measures how well the model performs on the validation data during training. The discrepancy between the expected and actual output is represented by the loss function. The model's parameters are updated during training to minimise the loss on the training data. The validation loss aids in determining if the model is overfitting or underfitting. If the validation loss is significantly greater than the training loss, the model is most certainly overfitting, which indicates it is memorising the training data rather than learning the overall pattern of the data. On the other side, a large validation loss suggests that the model is underfitting and not learning enough from the training data.

In summary, in Validation data is used to test the model's generalisation performance in automobile collision detection and alarm systems that employ deep learning., while validation loss is a metric used to monitor how well the model is performing during training. By monitoring the validation loss, the model's performance can be optimized and prevent overfitting or underfitting.

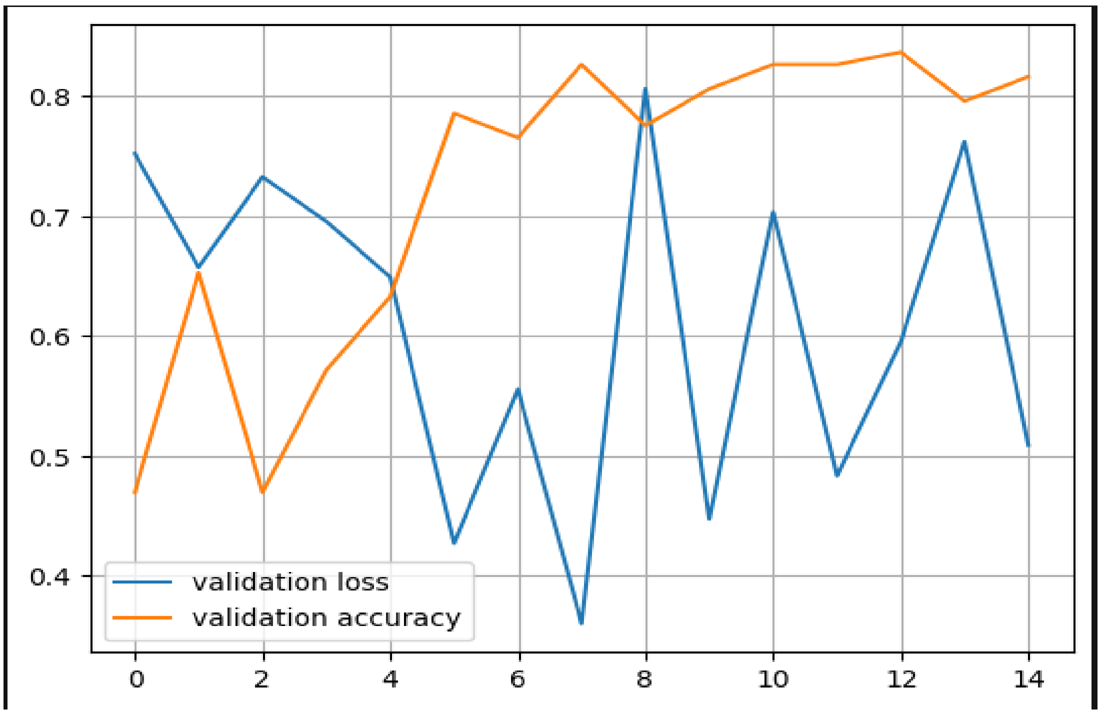


Fig 4.8.1 validation loss vs validation accuracy graph

4.9 Training loss vs training accuracy

During the training of a Convolutional Neural Network (CNN), two common metrics that are used to assess the model's performance are training loss and training accuracy.

The training loss measures the dissimilarity between the actual output of the model and the predicted output, and is usually calculated utilizing a loss function like cross-entropy or mean squared error. The primary objective is to minimize the training loss since it suggests that the model is generating precise predictions.

Training accuracy, on the other hand, measures the percentage of correctly classified samples in the training dataset. It is calculated by comparing the predicted output of the model to the true output, and counting the number of correct predictions. The goal is to maximize the training accuracy as much as possible, as this indicates that the model is correctly classifying samples in the training dataset.

It is important to note that training loss and training accuracy are not always perfectly correlated. In some cases, minimizing the training loss may not necessarily result in maximizing the training accuracy. For example, if the model is overfitting to the training data, it may achieve low training loss but poor accuracy on unseen data.

Therefore, when evaluating the performance of a CNN during training, it is important to consider both training loss and training accuracy. In general, the goal is to minimize the training loss while maximizing the training accuracy, and to monitor both metrics throughout the training process to ensure that the model is not overfitting to the training data.

4.10 Validation Accuracy

The validation accuracy of a deep learning-based car collision detection and alarm system might vary depending on the model architecture and dataset utilised for training and validation.

In general, a good model for this task should have a validation accuracy of at least 90%. However, it is important to note that accuracy alone may not be the only metric used to assess the performance of such a system, as precision, recall, and F1 score may also be considered.

Furthermore, the model's performance can be influenced by the quality and quantity of training data, the preprocessing techniques used, and the hyperparameters selected during training. Therefore, it's essential to properly evaluate and fine-tune the model to achieve the desired performance metrics.

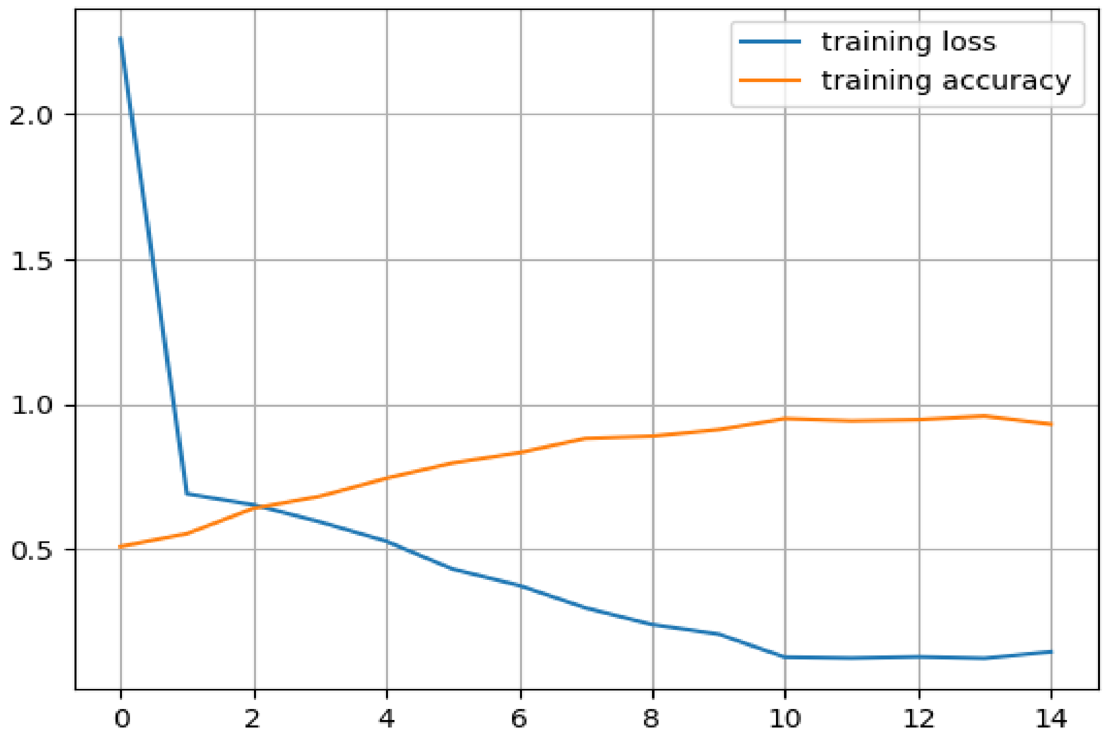


Fig 4.10.1 Training loss Vs accuracy graph

4.11 Training Loss

The training loss for the car collision detection and alarm system based on deep learning is determined by the model architecture and training data used.

The training loss for a vehicle collision detection and alert system using YOLO (You Only Look Once) is typically calculated using the mean squared error (MSE) or the binary cross-entropy loss function. The specific loss function used depends on the problem formulation and the desired output format.

In general, the purpose of deep learning model training is to minimise the loss function, which assesses the difference between expected and actual output. Binary cross-entropy, focused loss, and dice loss are all extensively used loss functions for binary classification problems like car accident detection.

During the training process, the loss is calculated at each iteration or epoch, and the weights of the model are adjusted to minimize the loss. As the training progresses, the loss typically decreases, indicating that the model is improving and learning to make better predictions.

The final value of the training loss can vary depending on factors such as the complexity of the model, the size of the training data, and the number of epochs used for training. Generally, a lower training loss indicates a better-performing model, but it's essential to ensure that the model does not overfit to the training data. Therefore, it's essential to monitor both the training and validation loss and adjust the model and hyperparameters accordingly.

4.12 Training Accuracy

The training accuracy for a vehicle collision detection and alert system using deep learning depends on the specific model architecture and training data used.

The model produces predictions on the training data during the training phase, and the accuracy is determined by comparing the anticipated output to the actual target output. The percentage of properly categorised instances in the training set is referred to as training accuracy.

The training accuracy typically increases as the model trains for more epochs, and the model learns to make better predictions. However, a high training accuracy does not necessarily mean that the model will perform well on unseen data.

Therefore, it's important to monitor both the training and validation accuracy during the training process. The validation accuracy is the percentage of correctly classified examples in a separate validation set that the model has not seen during training.

If the training accuracy is much greater than the validation accuracy, it may suggest that the model is overfitting to the training data and will not generalise effectively to new data. In such circumstances, the model may need to be adjusted and architecture, regularization techniques, or hyperparameters to improve the model's performance.

4. 13 Accuracy Calculation

There are some of the basic steps involved in using OpenCV for accident detection:

Dataset Collection: Collect a dataset of images or videos that contain examples of accidents. The dataset should be large enough and diverse enough to cover different types of accidents and scenarios.

Preprocessing: Preprocess the images or videos to enhance their quality and reduce noise. This may involve techniques such as color correction, contrast adjustment, and noise reduction.

Object Detection: Use OpenCV's object detection functions to identify potential hazards in the input images or videos. Object detection algorithms can be trained on the annotated dataset to learn to recognize relevant objects, such as vehicles, pedestrians, and other potential hazards.

Tracking: Once potential hazards have been detected, use OpenCV's tracking functions to track their movements over time. This can provide valuable information about the trajectory of the hazards and their potential impact.

Alert Generation: Once potential hazards have been detected and tracked, generate an alert or warning if the system detects an imminent collision or potential accident.

Overall, OpenCV can be a powerful tool for accident detection, as it provides a wide range of functions and algorithms for image and video processing. However, the quality of the results is determined by the quality of the input data, the application's specific requirements, and the design and implementation of the system.

4.14 TECHNOLOGIES USED

Google Colab:

Google Colab, short for "Google Collaboratory", is a free cloud-based development environment that allows users to execute machine learning on computer resources such as CPU, GPU, and TPU and YOLO projects. Colab is built on top of Jupyter Notebooks, which allows users to create, share, and collaborate on code and documents.

One of the primary advantages of utilising Google Colab is that it gives users access to powerful computing resources without the need to invest in expensive hardware. This is particularly useful for users who do not have access to high-performance computing resources or who cannot afford to purchase expensive hardware for their projects. With Colab, users can access powerful computing resources through their web browser, without the need to install any additional software on their local machine.

Another key feature of Google Colab is that it allows users to collaborate with others on their projects. Users can share their notebooks with others, allowing several people to work on the same project at the same time. This simplifies things for teams to collaborate on projects, as everyone can work on the same codebase in real-time.

Overall, Google Colab is a powerful and user-friendly development environment that gives users access to high-performance computing resources and collaborative work on machine learning and YOLO projects. Its ease of use and flexibility make it a popular choice for researchers, students, and professionals alike.

VS CODE:

VS Code is a robust code editor that may be used to create a deep learning-based automobile collision detection and alarm system. Here are the general steps you can take to create such a system using VS Code:

Gather and preprocess data: To train a deep learning model for collision detection, you will need a large dataset of images or videos of vehicles in various driving scenarios. You will also need to preprocess this data to ensure it is formatted correctly for use with your model.

Build and train your model: Using a deep learning framework such as TensorFlow or PyTorch, you can create a model that takes in images or video frames and predicts whether a collision is likely to occur. You will need to train your model using your preprocessed data.

Integrate your model into your vehicle: Once you have a working model, you will need to integrate it into your vehicle's systems. This may involve writing code that interfaces with your vehicle's sensors or cameras to collect data in real-time.

Implement an alert system: Using the output from your deep learning model, you can implement an alert system that warns drivers or takes action to prevent a collision. This may involve sounding an alarm, automatically applying the brakes, or taking other corrective action.

To accomplish all of these steps in VS Code, you will likely need to use a combination of programming languages such as Python and C++. You may also need to install additional libraries or dependencies for your chosen deep learning framework. With careful planning and attention to detail, however, you should be able to develop a robust and effective collision detection and alert system using VS Code.

KAGGLE:

Kaggle is a popular platform for YOLO competitions and projects, and it can be a great resource for developing it. Here are some general steps you can take to use Kaggle for this project:

Find relevant datasets: Kaggle hosts many datasets related to vehicles and driving, and you may be able to find a suitable dataset for your project. Look for datasets that contain images or videos of vehicles in various driving scenarios.

Preprocess the data: Depending on the format of the data, you may need to preprocess it to ensure it is formatted correctly for use with your deep learning model. This could involve resizing images, converting video to individual frames, or applying filters to remove noise.

Build and train your model: Using a deep learning framework such as TensorFlow or PyTorch, you can create a model that takes in images or video frames and predicts whether a collision is likely to occur. You will need to train your model using your preprocessed data.

Test and optimize your model: Once you have a working model, you will need to test it using a separate validation dataset to ensure it is accurate and effective. You may also need to optimize your model by adjusting hyperparameters or using more advanced techniques such as transfer learning.

Integrate your model into your vehicle: Once you have a working model, you will need to integrate it into your vehicle's systems. This may involve writing code that interfaces with your vehicle's sensors or cameras to collect data in real-time.

Implement an alert system: Using the output from your deep learning model, you can implement an alert system that warns drivers or takes action to prevent a collision. This may involve sounding an alarm, automatically applying the brakes, or taking other corrective action.

Kaggle provides a wealth of resources for each of these steps, including datasets, notebooks, and forums where you can ask questions and get help from other data scientists. By leveraging the power of Kaggle, you can accelerate the development of your project.

CHAPTER 5

CODING AND TESTING

5.1 Coding

Dataset Collection: We collected the dataset of medical images of vehicles which met with accidents and which did not meet with accident by searching for publicly available datasets and collaborating with healthcare organizations or hospitals to obtain the images. The dataset included a sufficient number of images to train and test the models effectively.

Image Preprocessing: We optimized the images and eliminated false negatives by applying various image processing techniques such as noise reduction, image enhancement, and normalization. This ensured that the models received high-quality images as input, leading to more accurate predictions.

Model Implementation: We implemented the few machine learning algorithms, Convolutional Neural Network (CNN), YOLO by writing code to train and test each model on the preprocessed dataset. We used Scikit-learn, TensorFlow, and Keras are examples of popular machine learning libraries. to implement the models.

Model Training: We trained the models on the preprocessed dataset by writing code that split the dataset into training and validation sets and then trained the models using the appropriate algorithms. For instance, we used Keras to build and train the CNN model, while scikit-learn was used to train KNN,

Model Testing: We tested the models on a separate set of images by writing code that loaded the trained models and evaluated their accuracy and efficiency on the test dataset. We used metrics such as accuracy.

5.2 Testing

Dataset Splitting: By building code that randomly chose a piece of the dataset for training and the remaining half for testing, we divided the dataset into training and testing sets.

Model Training: We trained the models on the training set by writing code that implemented the appropriate algorithms and hyperparameters. We used techniques such as cross-validation.

Model Evaluation: We evaluated the performance of the models on the testing set by writing code that used metrics such as accuracy.

CHAPTER 6

RESULTS AND DISCUSSION

We decided to test our project based on the following algorithms –

6.1 Convolution Neural Network (CNN)

As of now in our early phase of the project, CNN has proven to been more accurate than any other algorithm. The chat bot application which we have created to establish a connect between a doctor and a patient is working properly. The payment gateway application which aims at raising funds for the needy patients is successfully implemented. We successfully built a website in order to accommodate all the modules in one place.

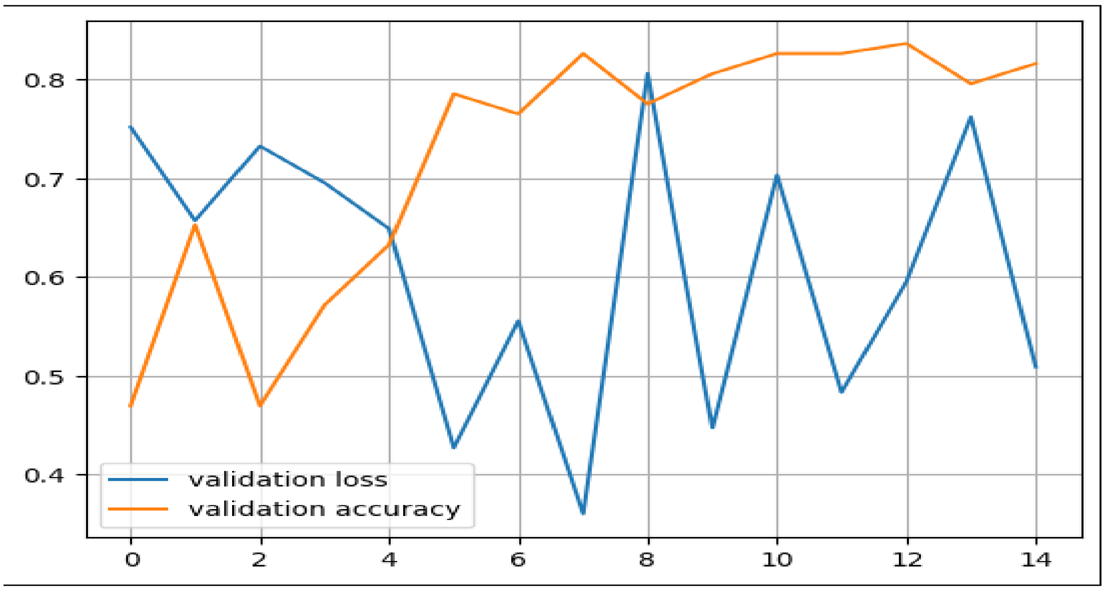


Figure 6.1 Graph of our training model

CHAPTER 7

CONCLUSION & FUTURE ENHANCEMENT

The conclusion for a vehicle collision detection and alert system using YOLO (You Only Look Once) would depend on the specific implementation and evaluation of the system. However, in general, such a system can offer several benefits and considerations.

Accuracy: YOLO is a popular real-time object detection algorithm that can accurately detect and classify objects in images or video frames. Its ability to detect multiple objects simultaneously and its fast inference time make it suitable for real-time collision detection systems.

Early Warning: By using YOLO to detect vehicles and their positions in real-time, a collision detection and alert system can provide early warnings to drivers or autonomous vehicles. This can help reduce the risk of accidents by giving drivers more time to react or allowing autonomous systems to initiate appropriate maneuvers

Limitations: While YOLO is a powerful object detection algorithm, it may have certain limitations. In some cases, it may struggle with detecting small or occluded objects, especially in challenging lighting or weather conditions. Therefore, it's important to consider these limitations and incorporate additional safety measures or sensor fusion techniques to ensure robust collision detection

Versatility: YOLO can be trained on a wide range of vehicle types, making it versatile for different collision scenarios. Whether it's detecting cars, trucks, motorcycles, or bicycles, YOLO can adapt to different environments and handle various collision detection requirements.

REFERENCES

[1] Yeong-Kang Lai, Chu-Ying Ho, Yu-Hau Huang, Chuan-Wei Huang, Yi-Xian Kuo, Yu-Chieh Chung, “Intelligent Vehicle Collision-Avoidance System with Deep Learning”, 2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), 2018

[2] Jae Gyeong Choi, Chan Woo Kong, Gyeongho Kim, Sunghoon Lim, “Car crash detection using ensemble deep learning and multimodal data from dashboard cameras”, Expert Systems with Applications, Volume 183, 30 November 2021,

[3] Aloukik Aditya, Liudu Zhou, Hrishika Vachhani, Dhivya Chandrasekaran, Vijay Mago, ” Collision Detection: An Improved Deep Learning Approach Using SENet and ResNext”, 2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2021

[4] Wan-Jung Chang, Liang-Bi Chen, Ke-Yu Su, “DeepCrash: A Deep Learning-Based Internet of Vehicles System for Head-On and Single-Vehicle Accident Detection With Emergency Notification”, IEEE Access, Vol 7, P-148163 – 148175, 2019

[5] Manu S. Pillai, Gopal Chaudhary, Manju Khari, Rubén González Crespo, “Real-time image enhancement for an automatic automobile accident detection through CCTV using deep learning”, Soft Computing volume 25, pages11929–11940, 2021

[6] G. Madhumitha, R. Senthilnathan, K. Muzammil Ayaz, J. Vignesh, Korada Madhu, ” Estimation of Collision Priority on Traffic Videos using Deep Learning”, 2020 IEEE International Conference on Machine Learning and Applied Network Technologies (ICMLANT), 2020

[7] Reagan L. Galvez , Argel A. Bandala, Elmer P. Dadios,Ryan Rhay P. Vicerra,Jose Martin Z. Maningo, “Object Detection Using Convolutional Neural Networks, IEEE 2018”

[8] Sen Ma, Shi-Huang Chen, Chun-Sheng Yang,Shu-Chuan Chu,Jeng-Shyang Pan, “Vision Based Front and Rear Vehicle Collision Warning System, IEEE Conference,2015”

[9] Swadesh Kumar Maurya, Ayesha Choudhary, “Deep Learning based Vulnerable Road User

Detection and Collision Avoidance,IEEE , 2018

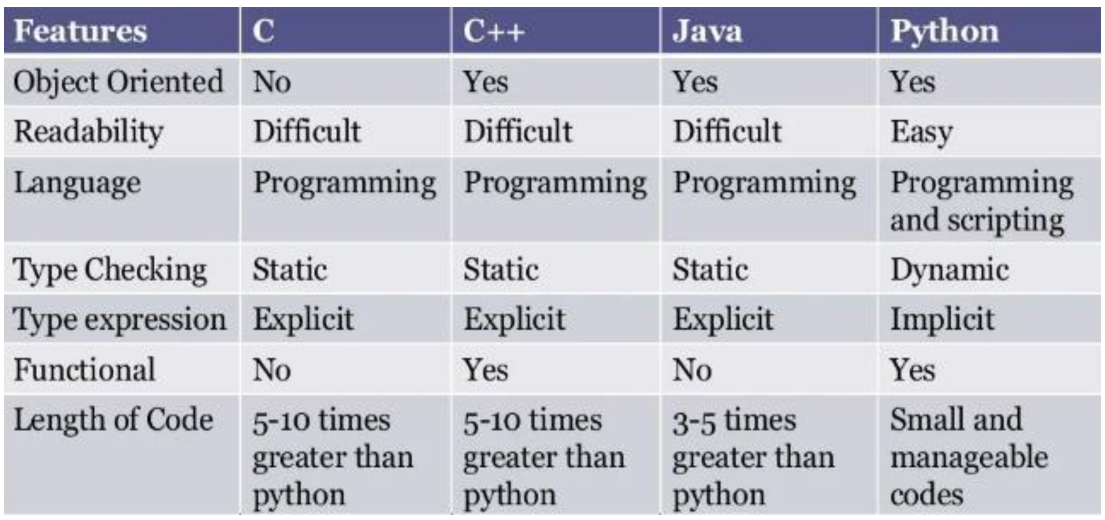
APPENDIX 1

This section contains details on the language, software and packages used in our project

Python Language

When Python was initially introduced as a high-level programming language in 1991, it was widely recognized for its reliability and wide range of capabilities. Even today, Python is still considered a popular programming language. For beginners, it appears to be very easy to learn since its structure is quite similar to the English language. Besides web development and scripting, Python is used for various other tasks such as machine learning and scripting. Additionally, it works seamlessly with servers and other third-party programs, making it a preferred alternative for many users. Depending on how the program is implemented, it can be categorized as functional or object-oriented.

Comparison of various programming languages



Features	C	C++	Java	Python
Object Oriented	No	Yes	Yes	Yes
Readability	Difficult	Difficult	Difficult	Easy
Language	Programming	Programming	Programming	Programming and scripting
Type Checking	Static	Static	Static	Dynamic
Type expression	Explicit	Explicit	Explicit	Implicit
Functional	No	Yes	No	Yes
Length of Code	5-10 times greater than python	5-10 times greater than python	3-5 times greater than python	Small and manageable codes

Jupyter Notebook

Jupyter Notebook is a web-based, open-source tool that enables users to create and share documents containing code, equations, text, and visuals. It has a broad range of applications, including statistical modeling, machine learning, data visualization, data transformation, and cleaning. Originally, it emerged from a project called the J Notebook, which included its own notebook. Jupyter supports Julia, Python, and R, which gave rise to its name, but more than a hundred additional kernels are presently available. The IPython kernel, which supports Python programming, comes with Jupyter by default.

Jupyter Notebook is an interactive computational environment that operates on the web, enabling users to create Jupyter notebook documents. The term "notebook" can refer to various objects, including the Jupyter web application, Jupyter Python web server, or Jupyter document format, depending on the context. Project Jupyter is focused on developing open-source software, standards, and services for interactive computing across numerous programming languages. Jupyter Book is another open-source project that enables users to create documents and books using computational data. Users can create content utilizing Markdown, MyST, MathJax, Jupyter Notebooks, reStructuredText, and the output of running Jupyter Notebooks at build time for Math & Equations.

Output can be generated in various formats. To learn more about Jupyter Notebooks and Project Jupyter's groundbreaking nature in today's world, one should obtain a basic understanding of the concept. Installing the Anaconda package is the most effective approach to install Jupyter Notebooks. In this case, Python 3.3 or higher or Python 2.7 is a prerequisite. Installing Python and the notebook application through the Anaconda distribution is recommended. With Anaconda, users can easily install more than 720 packages because it includes conda, which is a package, dependency, and environment manager.

If a user prefers to install Jupyter Notebook using the Pythonic way, without using

the Anaconda package manager, they should acquire the most recent version of Python (currently 3.9) and then run the following command on the command prompt or PowerShell: "pip install jupyter."

Numpy

NumPy is a powerful numerical computing library in Python that provides an efficient and easy-to-use interface for performing numerical operations on large sets of data. It is designed to handle the multidimensional arrays, which are fundamental building blocks of scientific computing. These arrays are similar to lists or vectors, but can hold more than one dimension, which makes them useful for representing and manipulating data that have multiple dimensions, such as images or time series data.

NumPy is built on top of the C programming language, which provides fast and efficient computation capabilities. This means that NumPy is able to handle large data sets and perform complex calculations quickly and efficiently. In addition, NumPy provides a variety of mathematical and statistical functions, such as matrix operations, Fourier transforms, and random number generation.

One of the key benefits of NumPy is its ability to perform vectorized operations, which means that it can perform the same operation on multiple elements of an array at once, without the need for a loop. This significantly speeds up the computation and makes it more efficient.

NumPy is widely used in scientific computing, data analysis, and machine learning, as it provides a solid foundation for working with large data sets and performing complex calculations. It is also a key component of many other popular Python libraries, such as pandas, scipy, and scikit-learn.

Overall, NumPy is a powerful tool for numerical computing in Python that provides a flexible and efficient interface for working with multidimensional arrays and performing complex mathematical operations.

VS Code

VS Code is a powerful code editor that can be used for developing a vehicle collision detection and alert system using deep learning. Here are the general steps you can take to create such a system using VS Code:

Gather and preprocess data: To train a deep learning model for collision detection, you will need a large dataset of images or videos of vehicles in various driving scenarios. You will also need to preprocess this data to ensure it is formatted correctly for use with your model.

Build and train your model: Using a deep learning framework such as TensorFlow or PyTorch, you can create a model that takes in images or video frames and predicts whether a collision is likely to occur. You will need to train your model using your preprocessed data.

Integrate your model into your vehicle: Once you have a working model, you will need to integrate it into your vehicle's systems. This may involve writing code that interfaces with your vehicle's sensors or cameras to collect data in real-time.

Implement an alert system: Using the output from your deep learning model, you can implement an alert system that warns drivers or takes action to prevent a collision. This may involve sounding an alarm, automatically applying the brakes, or taking other corrective action.

To accomplish all of these steps in VS Code, you will likely need to use a combination of programming languages such as Python and C++. You may also need to install additional libraries or dependencies for your chosen deep learning framework. With careful planning and attention to detail, however, you should be able to develop a robust and effective collision detection and alert system using VS Code.

KAGGLE

Kaggle is a popular platform for YOLO competitions and projects, and it can be a great resource for developing a vehicle collision detection and alert system using deep learning. Here are some general steps you can take to use Kaggle for this project:

Find relevant datasets: Kaggle hosts many datasets related to vehicles and driving, and you may be able to find a suitable dataset for your project. Look for datasets that contain images or videos of vehicles in various driving scenarios.

Preprocess the data: Depending on the format of the data, you may need to preprocess it to ensure it is formatted correctly for use with your deep learning model. This could involve resizing images, converting video to individual frames, or applying filters to remove noise.

Build and train your model: Using a deep learning framework such as TensorFlow or PyTorch, you can create a model that takes in images or video frames and predicts whether a collision is likely to occur. You will need to train your model using your preprocessed data.

Test and optimize your model: Once you have a working model, you will need to test it using a separate validation dataset to ensure it is accurate and effective. You may also need to optimize your model by adjusting hyperparameters or using more advanced techniques such as transfer learning. Integrate your model into your vehicle: Once you have a working model, you will need to integrate it into your vehicle's systems. This may involve writing code that interfaces with your vehicle's sensors or cameras to collect data in real-time.

Implement an alert system: Using the output from your deep learning model, you can implement an alert system that warns drivers or takes action to prevent a collision. This may involve sounding an alarm, automatically applying the brakes, or taking other corrective action.

Kaggle provides a wealth of resources for each of these steps, including datasets, notebooks, and forums where you can ask questions and get help from other data scientists. By leveraging the power of Kaggle, you can accelerate the development of your vehicle collision detection and alert system using deep learning

OpenCV

OpenCV (Open Source Computer Vision) is a powerful library of computer vision algorithms and tools that can be used for a wide range of applications, including accident detection. Here are the steps you can take to use OpenCV for accident detection:

Obtain video footage: The first step is to obtain video footage of the area where accidents may occur. This can be done using CCTV cameras or other types of cameras.

Preprocess the video: The video footage may contain noise or other artifacts that can affect the accuracy of accident detection. You can use OpenCV to preprocess the video and remove any unwanted elements.

Object detection: The next step is to use object detection algorithms to identify objects in the video that may indicate an accident. For example, you can use Haar cascades or deep learning-based object detection models to identify vehicles, pedestrians, or other objects that are likely to be involved in an accident.

Tracking: Once you have detected objects of interest, you can use OpenCV to track their movements over time. This can help you determine if there is any unusual behavior or if an accident has occurred.

Alert system: Finally, you can use the information obtained from the object detection and tracking algorithms to trigger an alert system. This can be done using audio alarms, visual alerts, or other types of notifications.

Overall, OpenCV can be a powerful tool for accident detection, but it requires careful tuning and testing to ensure that it is accurate and reliable.

Tensorflow

TensorFlow is a well-known open-source software framework used to create and train machine learning models. It is widely utilised in a wide range of applications, including natural language processing, computer vision, and predictive analytics. TensorFlow can also be used for accident detection by using image and video processing techniques.

To use TensorFlow for accident detection, a model can be trained on a dataset of accident images and videos. The model can then be used to detect accidents in real-time by analyzing incoming video streams. The model can detect accidents by analyzing various factors such as the presence of smoke, fire, or debris, sudden movements, and changes in sound patterns.

When an accident is identified, the model can activate an alarm system that notifies emergency services or other parties. The alert system can be configured to send notifications via email, text message, or phone call.

A collection of accident photos and videos may be gathered and labelled to train a TensorFlow model for accident detection. The labelled data may then be used to train the model with approaches like convolutional neural networks (CNNs) and recurrent neural networks (RNNs).

In summary, TensorFlow can be used for accident detection by training a model on a dataset of accident images and videos and analyzing incoming video streams in real-time. The model can trigger an alert system to notify emergency services or other relevant parties, potentially saving lives and minimizing damage.

YOLO

The YOLO (You Only Look Once) method is a common object detection technique that may be used to identify accidents. YOLO is a real-time object detection system that can recognise objects with high accuracy and speed in an image or video stream.

To use YOLO for accident detection, a model can be trained on a dataset of accident images and videos. The model can then be used to analyze incoming video streams in real-time and detect accidents by identifying various objects and patterns associated with accidents.

The YOLO algorithm divides an image or video stream into cells and predicts the likelihood of an item being present in each cell. The system then uses regression to refine the estimated bounding boxes around the items. With excellent precision and speed, YOLO can recognise several objects in an image or video stream.

To train a YOLO model for accident detection, a dataset of accident images and videos can be collected and labeled. The labeled data can then be used to train the model using various techniques such as transfer learning and data augmentation.

Once an accident is detected, the YOLO model can trigger an alert system to notify emergency services or other relevant parties. The alert system can be configured to send notifications via email, text message, or phone call.

In summary, YOLO can be used for accident detection by training a model on a dataset of accident images and videos and analyzing incoming video streams in real-time. The model can trigger an alert system to notify emergency services or other relevant parties, potentially saving lives and minimizing damage.

Bounding box object detection

Bounding box detection, also known as object detection, is a computer vision technique used to locate and classify objects in an image or video stream. It involves identifying objects of interest and drawing a bounding box around them to define their location and size within the image.

The process of bounding box detection typically involves several steps:

Image Preprocessing: Preprocess the input image or video to enhance its quality and reduce noise. This may involve techniques such as color correction, contrast adjustment, and noise reduction.

Feature Extraction: Extract features from the preprocessed image or video that are relevant to the objects of interest. These features could include color, texture, and shape information.

Object Detection: Use a machine learning or deep learning algorithm to detect objects in the input image or video. The algorithm learns to recognize objects based on the features extracted from the image or video.

Bounding Box Generation: Once an object is detected, a bounding box is generated around it to define its location and size within the image. The bounding box typically includes the coordinates of the top-left and bottom-right corners of the box, as well as the width and height of the box.

Object Classification: Once the objects have been detected and bounded, they can be classified into different categories. For example, objects in a traffic scene could be classified into cars, pedestrians, bicycles, and other categories.

Bounding box detection has many practical applications, such as in surveillance systems, autonomous vehicles, and medical imaging. It is a critical component of many computer vision systems and is essential for accurate object recognition and tracking. However, the quality of the results depends on the quality and diversity of the input data, the specific requirements of the application, and the design and implementation of the system.

Keras

Keras is a Python-based open-source high-level neural network framework. It is intended to provide a straightforward and user-friendly interface for developing and training deep learning models. Keras is a deep learning framework that is built on top of various lower-level deep learning frameworks such as TensorFlow, Microsoft Cognitive Toolkit, and Theano.

Keras provides a wide range of tools and functions for building and training deep learning models, including:

1. Model Architecture: Keras allows you to define and customize the architecture of your deep learning model. You can choose from a wide range of layers and activation functions, and easily add or remove layers to fine-tune your model.

2. Training: Keras provides a simple interface for training your deep learning model. You can choose from a wide range of optimization algorithms and loss functions, and monitor the training process in real-time.

3. Evaluation: Keras allows to assess the performance of your deep learning model using a validation dataset. To evaluate the performance of your model, you may compute metrics such as accuracy, precision, recall, and F1-score.

4. Integration: Keras can be integrated with other popular deep learning frameworks such as TensorFlow, making it easy to build and train complex deep learning models.

5. Community Support: Keras has a big and active user and developer community that provides a plethora of resources and assistance for building and training deep learning models.

Keras is extensively used in business and academics for a variety of deep learning applications like as computer vision, natural language processing, and speech recognition. Its simple and user-friendly interface makes it accessible to beginners, while its flexibility and customization options make it suitable for advanced users.

Detection:

Vehicle collision detection and alert systems are critical for ensuring road safety and reducing the number of accidents. Deep learning algorithms can be used for detecting collisions and alerting drivers in real-time.

One approach to building such a system is to use a camera mounted on the front of the car to take pictures of the road ahead. A deep learning technique is then used to detect probable collisions in the photos. A huge collection of labelled photos may be used to train the algorithm. that include various types of collisions, such as rear-end, head-on, and side collisions.

The deep learning algorithm can be designed to identify key features in the images, such as the distance between vehicles, the relative speed of the vehicles, and the direction of travel. Based on these features, the algorithm can determine whether a collision is likely to occur and provide an alert to the driver.

Another approach is to use sensor data from the vehicle, such as radar or lidar, to detect potential collisions. Deep learning algorithms can be trained using data from these sensors to identify patterns and predict collision risks.

To improve the accuracy of the collision detection system, it is important to ensure that the deep learning algorithm is trained on a diverse range of data, including different weather and lighting conditions, road types, and traffic situations. The system should also be tested extensively to ensure that it can detect collisions accurately and provide timely alerts to drivers.

In summary, a vehicle collision detection and alert system using deep learning can improve road safety by detecting potential collisions and alerting drivers in real- time. By leveraging advances in deep learning algorithms and sensor technology, such systems can be designed to accurately detect collisions and provide timely warnings to drivers, potentially saving lives and reducing the number of accidents on the road.

OS in python:

There are numerous operating systems that may be utilised to create a car collision detection and alarm system utilising Python deep learning.. Here are some of the commonly used operating systems:

Ubuntu: Ubuntu is a popular Linux-based operating system This is a popular programming language for creating deep learning applications. It offers a safe and secure framework for creating and deploying Python-based machine learning

algorithms. applications. Ubuntu also offers a wide range of pre-installed packages and tools that can be used for deep learning.

Windows: Windows is a popular operating system that supports a wide range of applications of development tools and packages for Python-based deep learning. It provides a user-friendly interface and supports a wide range of hardware devices and drivers that can be used for developing deep learning applications.

macOS: macOS is an operating system designed for Apple's hardware, and it supports a wide range of development tools and packages for Python-based deep learning. It provides a user-friendly interface and supports a wide range of hardware devices and drivers that can be used for developing deep learning applications.

Regardless of the operating system you choose, it is important to ensure that your system meets the requirements for deep learning development, such as having a powerful CPU/GPU, sufficient memory, and storage capacity. You should also ensure that the necessary deep learning libraries and packages, such as TensorFlow, PyTorch, and Keras, are installed on your system to develop and train deep learning models.

pref_counter:

Preference counting (pref_counter) is a technique used in machine learning to help rank and prioritize predictions made by a model. In the context of a project pref_counter can be used to improve the accuracy and reliability of collision detection by giving more weight to predictions that are more likely to result in a collision.

The basic idea behind pref_counter is to keep track of the number of times a particular prediction is made by the model. The prediction with the highest count is considered the most likely to be correct. However, it is also important to take

into account the confidence of the model in each prediction. Predictions that are made with high confidence should be given more weight than predictions that are made with low confidence.

To implement pref_counter in a vehicle collision detection and alert system, you would first need to train a deep learning model on a large dataset of labeled collision and non-collision scenarios. Once the model is trained, you can use it to make predictions in real-time based on the input from sensors, such as cameras or radar.

For each prediction, you would increment a counter for that particular prediction, and also calculate the confidence of the prediction. This can be done using a metric such as probability or certainty. The weight given to each prediction can then be calculated as a function of the count and confidence of the prediction. Predictions with a higher weight are more likely to result in a collision, and should trigger an alert to the driver.

In summary, pref_counter is a technique that can be used to improve the accuracy and reliability of collision detection in a vehicle alert system using deep learning. By prioritizing predictions based on the count and confidence of the model, pref_counter can help to reduce false positives and ensure that drivers are alerted only when a collision is likely to occur.

Plot_model:

Plot_model is a function in the Keras library that allows you to visualize the architecture of a deep learning model. It can be a useful tool in the development of a vehicle collision detection and alert system using deep learning, as it allows you to get a better understanding of the structure of your model and identify any potential issues.

To use plot_model in your vehicle collision detection and alert system, you wouldfirst need to define your model using the Keras API. This typically involves specifying the input shape, the number of layers, and the activation functions to be used. Once your model is defined, you can use the plot_model function to generate a diagram of the architecture of your model.

The plot_model function allows you to specify a number of different parameters, such as the color and style of the nodes and edges in the diagram, as well as the direction in which the diagram is displayed. This can be helpful in making the diagram more readable and easier to understand.

In addition to visualizing the architecture of your model, plot_model can also be used to generate diagrams of other aspects of your model, such as the training and validation metrics. This can be useful in tracking the performance of your model over time and identifying areas for improvement.

In summary, plot_model is a powerful tool for visualizing the architecture of a deep learning model, and can be a useful tool in the development of a vehicle collision detection and alert system using deep learning. By providing a clear and intuitive representation of your model's structure, plot_model can help you to identify potential issues and improve the overall performance of your system.

Layers in Python:

YOLO (You Only Look Once) is a popular deep learning model for object detection, including in the context of vehicle collision detection and alert systems. YOLO uses convolutional neural networks (CNNs) to detect objects in images or video frames in real-time. In Python, YOLO can be implemented using the Darknet framework.

Here are some commonly used layers in YOLO for vehicle collision detection and alert system:

The YOLO framework utilizes various layers such as Convolutional Layers to extract image features, Max Pooling Layers to reduce the spatial size of the feature maps, Fully Connected Layers to classify detected objects, Dropout Layers to prevent overfitting, and Activation Layers to add nonlinearity to the model. In addition, YOLO uses anchor boxes and objectness scores to enhance the accuracy of object detection. By employing these techniques, YOLO provides a flexible and robust platform for developing vehicle collision detection and alert systems with deep learning in Python.

Matplotlib:

Matplotlib is a popular Python library for creating visualizations and plots. It can be used in the development of a vehicle collision detection and alert system using YOLO to visualize the output of the YOLO model, such as the detection boxes, object classes, and confidence scores.

In addition to displaying images with detection boxes and class labels, Matplotlib can also be used to plot training and validation metrics, such as loss and accuracy, during the training of the YOLO model. This can be helpful in monitoring the performance of the model and identifying areas for improvement.

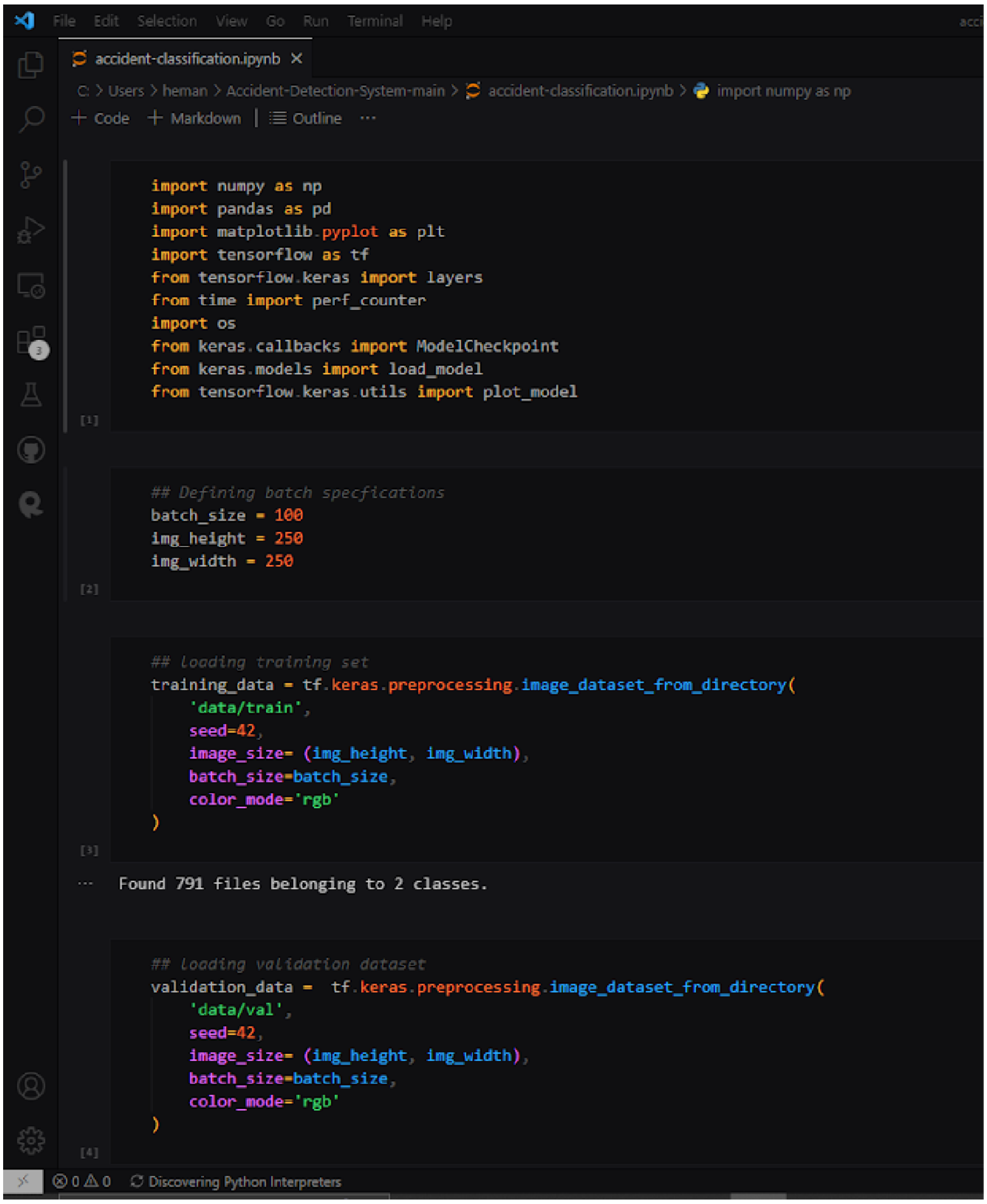
Pyplot:

pyplot is a sub-library of Matplotlib, which provides a set of functions to create a variety of charts and plots in Python. It can be used to visualize and analyze the output of YOLO-based vehicle collision detection and alert systems.

pyplot can also be used to create other types of plots, such as line charts, scatter plots, and bar charts, to visualize and analyze the output of YOLO-based vehicle collision detection and alert systems. The specific type of plot will depend on the type of data and analysis needed.

APPENDIX 2

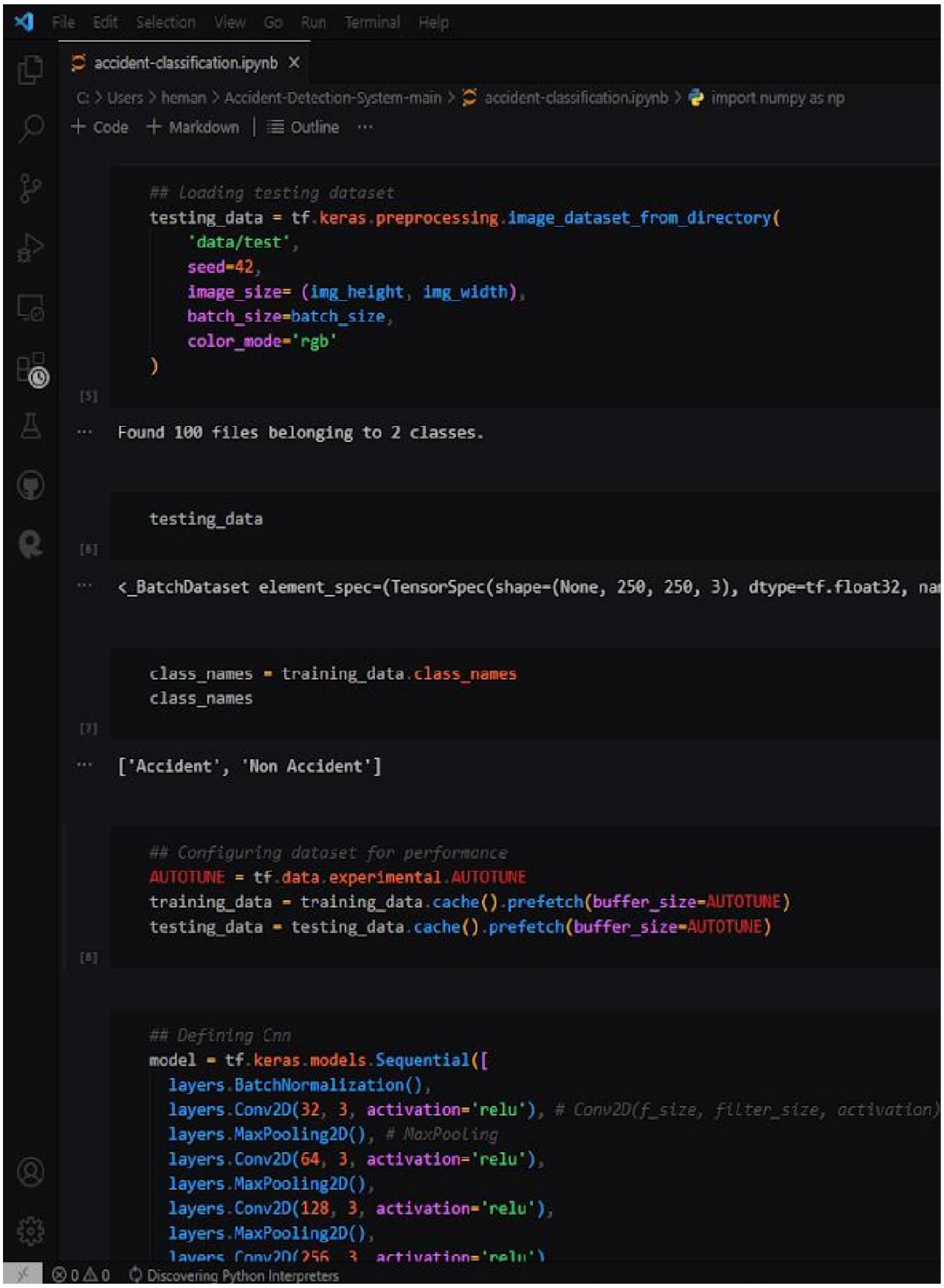
A 2.1 Code for Accident Classification



File Edit Selection View Go Run Terminal Help
accident-classification.ipynb
C: > Users > heman > Accident-Detection-System-main > accident-classification.ipynb > import numpy as np
+ Code + Markdown | Outline ...
Discovering Python Interpreters

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers
from time import perf_counter
import os
from keras.callbacks import ModelCheckpoint
from keras.models import load_model
from tensorflow.keras.utils import plot_model
[1]

## Defining batch specfications
batch_size = 100
img_height = 250
img_width = 250
[2]

## loading training set
training_data = tf.keras.preprocessing.image_dataset_from_directory(
    'data/train',
    seed=42,
    image_size= (img_height, img_width),
    batch_size=batch_size,
    color_mode='rgb'
)
[3]
... Found 791 files belonging to 2 classes.

## loading validation dataset
validation_data =  tf.keras.preprocessing.image_dataset_from_directory(
    'data/val',
    seed=42,
    image_size= (img_height, img_width),
    batch_size=batch_size,
    color_mode='rgb'
)
[4]
```

A 2.2



File Edit Selection View Go Run Terminal Help
accident-classification.ipynb
C: > Users > heman > Accident-Detection-System-main > accident-classification.ipynb > import numpy as np
+ Code + Markdown | Outline ...

```
## Loading testing dataset
testing_data = tf.keras.preprocessing.image_dataset_from_directory(
    'data/test',
    seed=42,
    image_size= (img_height, img_width),
    batch_size=batch_size,
    color_mode='rgb'
)
```

[5]

```
Found 100 files belonging to 2 classes.
```

```
testing_data
```

[6]

```
<_BatchDataset element_spec=(TensorSpec(shape=(None, 250, 250, 3), dtype=tf.float32, na
```

```
class_names = training_data.class_names
class_names
```

[7]

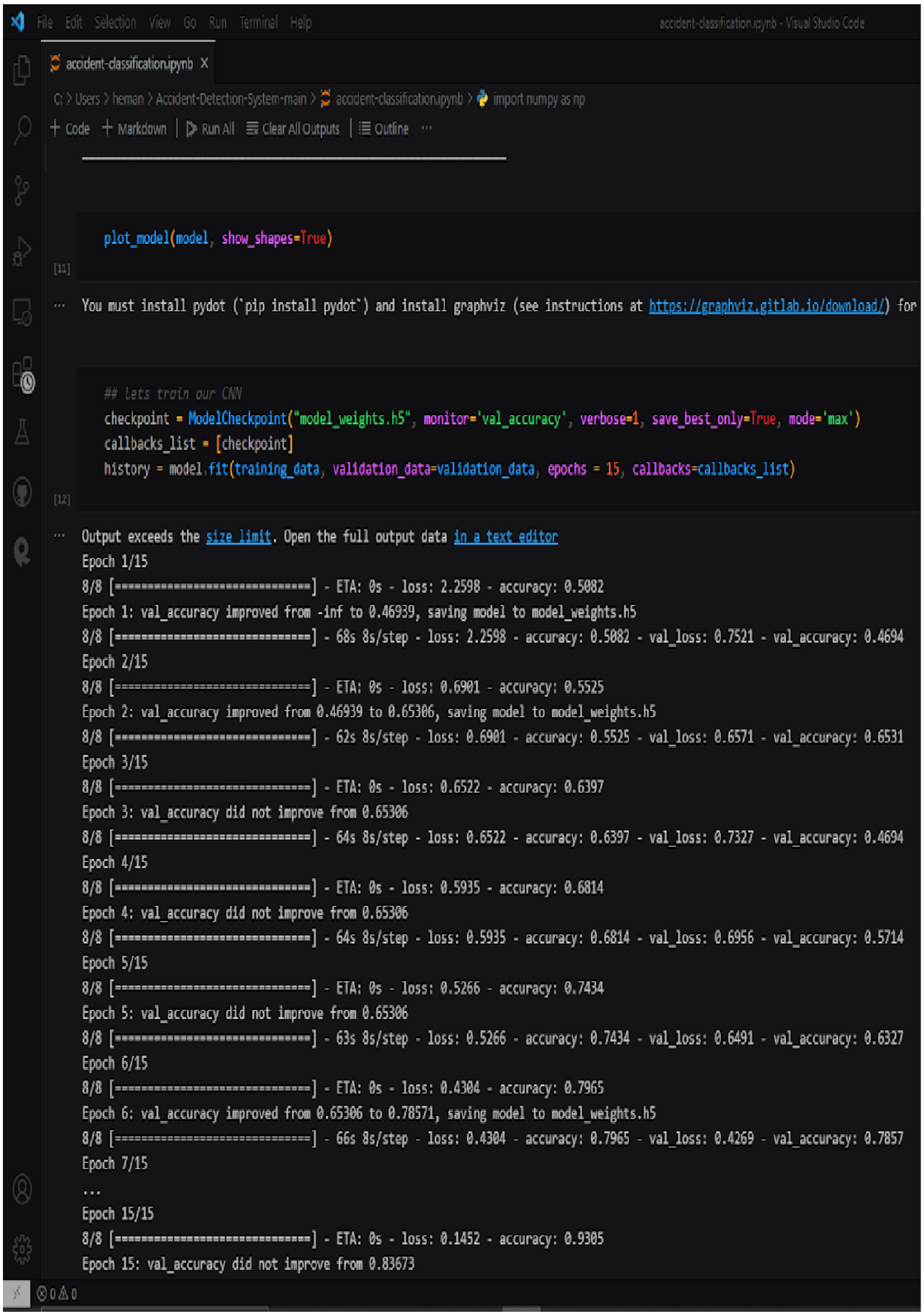
```
['Accident', 'Non Accident']
```

```
## Configuring dataset for performance
AUTOTUNE = tf.data.experimental.AUTOTUNE
training_data = training_data.cache().prefetch(buffer_size=AUTOTUNE)
testing_data = testing_data.cache().prefetch(buffer_size=AUTOTUNE)
```

[8]

```
## Defining Cnn
model = tf.keras.models.Sequential([
  layers.BatchNormalization(),
  layers.Conv2D(32, 3, activation='relu'), # Conv2D(f_size, filter_size, activation)
  layers.MaxPooling2D(), # MaxPooling
  layers.Conv2D(64, 3, activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(128, 3, activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(256, 3, activation='relu')
```

A 2.3


File Edit Selection View Go Run Terminal Help
accident-classification.ipynb - Visual Studio Code
accident-classification.ipynb
C: > Users > heman > Accident-Detection-System-main > accident-classification.ipynb > import numpy as np
+ Code + Markdown | Run All Clear All Outputs | Outline ...

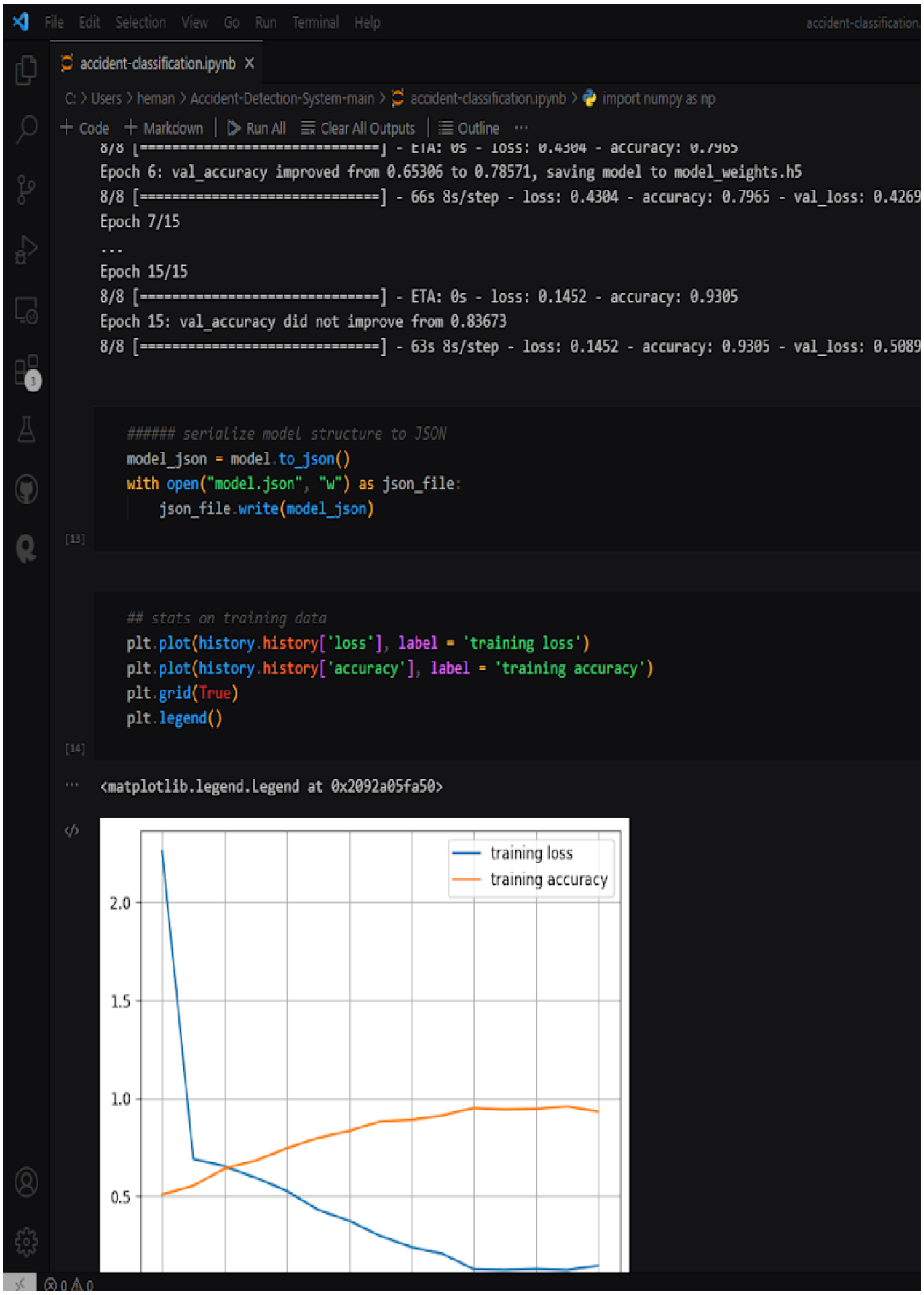
```
plot_model(model, show_shapes=True)
```
[11]

```
You must install pydot (`pip install pydot`) and install graphviz (see instructions at https://graphviz.gitlab.io/download/) for
```

```
## lets train our CNN
checkpoint = ModelCheckpoint("model_weights.h5", monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')
callbacks_list = [checkpoint]
history = model.fit(training_data, validation_data=validation_data, epochs = 15, callbacks=callbacks_list)
```
[12]

```
Output exceeds the size limit. Open the full output data in a text editor
Epoch 1/15
8/8 [==============================] - ETA: 0s - loss: 2.2598 - accuracy: 0.5082
Epoch 1: val_accuracy improved from -inf to 0.46939, saving model to model_weights.h5
8/8 [==============================] - 68s 8s/step - loss: 2.2598 - accuracy: 0.5082 - val_loss: 0.7521 - val_accuracy: 0.4694
Epoch 2/15
8/8 [==============================] - ETA: 0s - loss: 0.6901 - accuracy: 0.5525
Epoch 2: val_accuracy improved from 0.46939 to 0.65306, saving model to model_weights.h5
8/8 [==============================] - 62s 8s/step - loss: 0.6901 - accuracy: 0.5525 - val_loss: 0.6571 - val_accuracy: 0.6531
Epoch 3/15
8/8 [==============================] - ETA: 0s - loss: 0.6522 - accuracy: 0.6397
Epoch 3: val_accuracy did not improve from 0.65306
8/8 [==============================] - 64s 8s/step - loss: 0.6522 - accuracy: 0.6397 - val_loss: 0.7327 - val_accuracy: 0.4694
Epoch 4/15
8/8 [==============================] - ETA: 0s - loss: 0.5935 - accuracy: 0.6814
Epoch 4: val_accuracy did not improve from 0.65306
8/8 [==============================] - 64s 8s/step - loss: 0.5935 - accuracy: 0.6814 - val_loss: 0.6956 - val_accuracy: 0.5714
Epoch 5/15
8/8 [==============================] - ETA: 0s - loss: 0.5266 - accuracy: 0.7434
Epoch 5: val_accuracy did not improve from 0.65306
8/8 [==============================] - 63s 8s/step - loss: 0.5266 - accuracy: 0.7434 - val_loss: 0.6491 - val_accuracy: 0.6327
Epoch 6/15
8/8 [==============================] - ETA: 0s - loss: 0.4304 - accuracy: 0.7965
Epoch 6: val_accuracy improved from 0.65306 to 0.78571, saving model to model_weights.h5
8/8 [==============================] - 66s 8s/step - loss: 0.4304 - accuracy: 0.7965 - val_loss: 0.4269 - val_accuracy: 0.7857
Epoch 7/15
...
Epoch 15/15
8/8 [==============================] - ETA: 0s - loss: 0.1452 - accuracy: 0.9305
Epoch 15: val_accuracy did not improve from 0.83673
```

A 2.4


File Edit Selection View Go Run Terminal Help
accident-classification.ipynb
C: > Users > heman > Accident-Detection-System-main > accident-classification.ipynb > import numpy as np
+ Code + Markdown | Run All Clear All Outputs | Outline ...
training loss
training accuracy
2.0
1.5
1.0
0.5

```
8/8 [==============================] - ETA: 0s - loss: 0.4304 - accuracy: 0.7965
Epoch 6: val_accuracy improved from 0.65306 to 0.78571, saving model to model_weights.h5
8/8 [==============================] - 66s 8s/step - loss: 0.4304 - accuracy: 0.7965 - val_loss: 0.4269
Epoch 7/15
...
Epoch 15/15
8/8 [==============================] - ETA: 0s - loss: 0.1452 - accuracy: 0.9305
Epoch 15: val_accuracy did not improve from 0.83673
8/8 [==============================] - 63s 8s/step - loss: 0.1452 - accuracy: 0.9305 - val_loss: 0.5089
```

```
###### serialize model structure to JSON
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
```

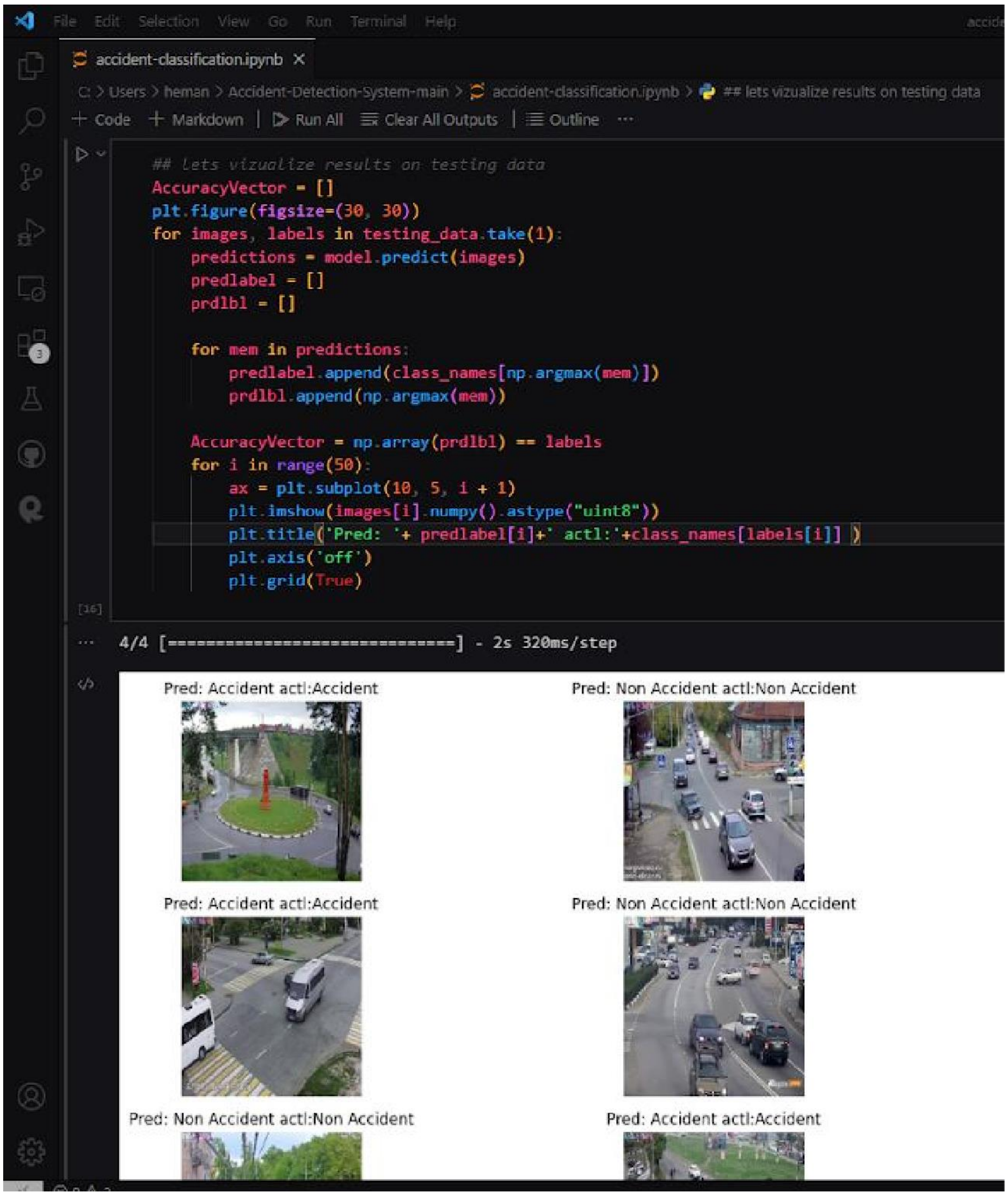
[13]

```
## stats on training data
plt.plot(history.history['loss'], label = 'training loss')
plt.plot(history.history['accuracy'], label = 'training accuracy')
plt.grid(True)
plt.legend()
```

[14]

```
<matplotlib.legend.Legend at 0x2092a05fa50>
```

A 2.5

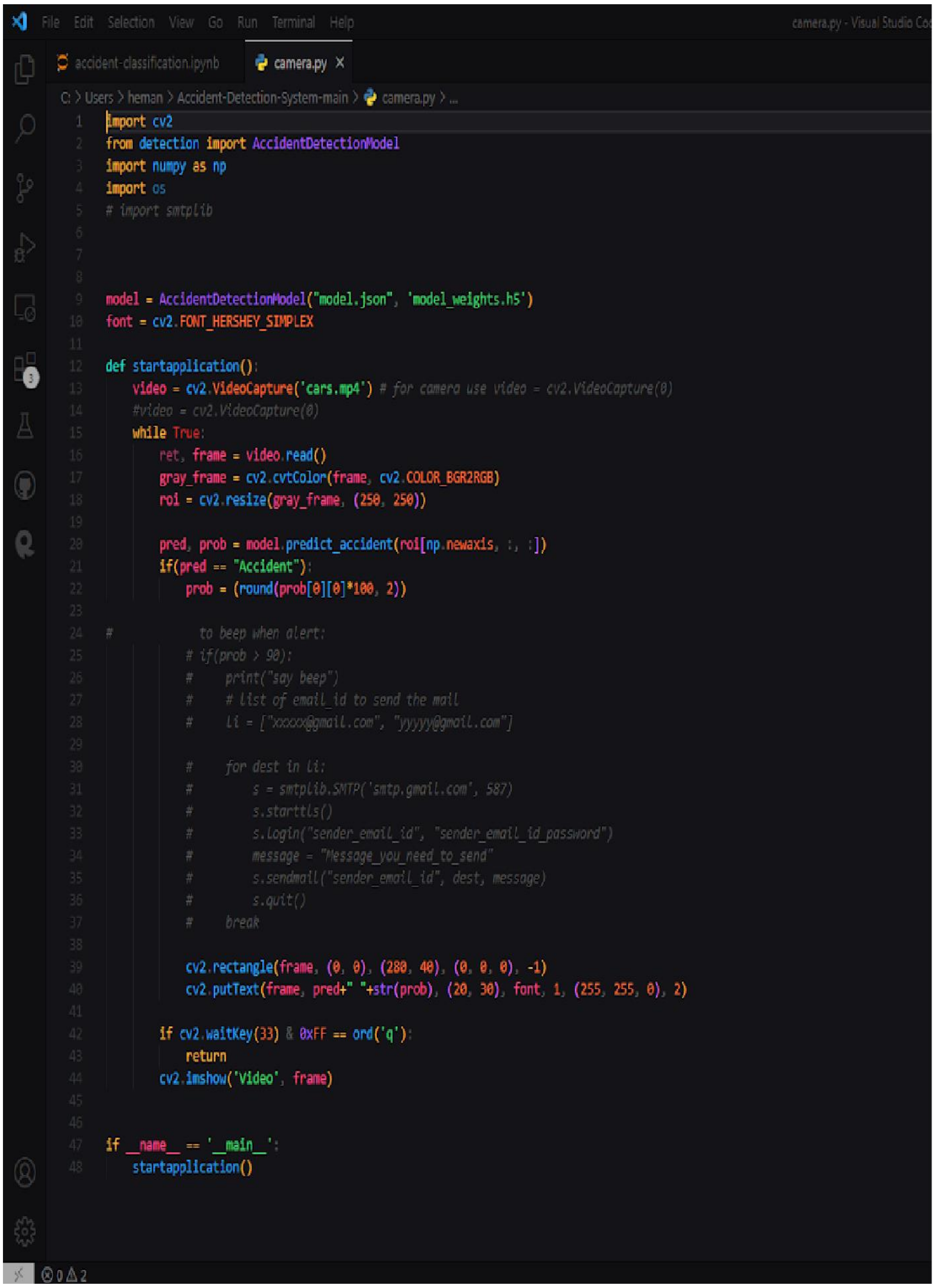

File Edit Selection View Go Run Terminal Help
accident-classification.ipynb
C: > Users > heman > Accident-Detection-System-main > accident-classification.ipynb > ## lets vizualize results on testing data
+ Code + Markdown | Run All Clear All Outputs | Outline ···

```
## lets vizualize results on testing data
AccuracyVector = []
plt.figure(figsize=(30, 30))
for images, labels in testing_data.take(1):
    predictions = model.predict(images)
    predlabel = []
    prdlbl = []

    for mem in predictions:
        predlabel.append(class_names[np.argmax(mem)])
        prdlbl.append(np.argmax(mem))

    AccuracyVector = np.array(prdlbl) == labels
    for i in range(50):
        ax = plt.subplot(10, 5, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title('Pred: '+ predlabel[i]+' actl:'+class_names[labels[i]] )
        plt.axis('off')
        plt.grid(True)
```

```
[16]
4/4 [==============================] - 2s 320ms/step
```

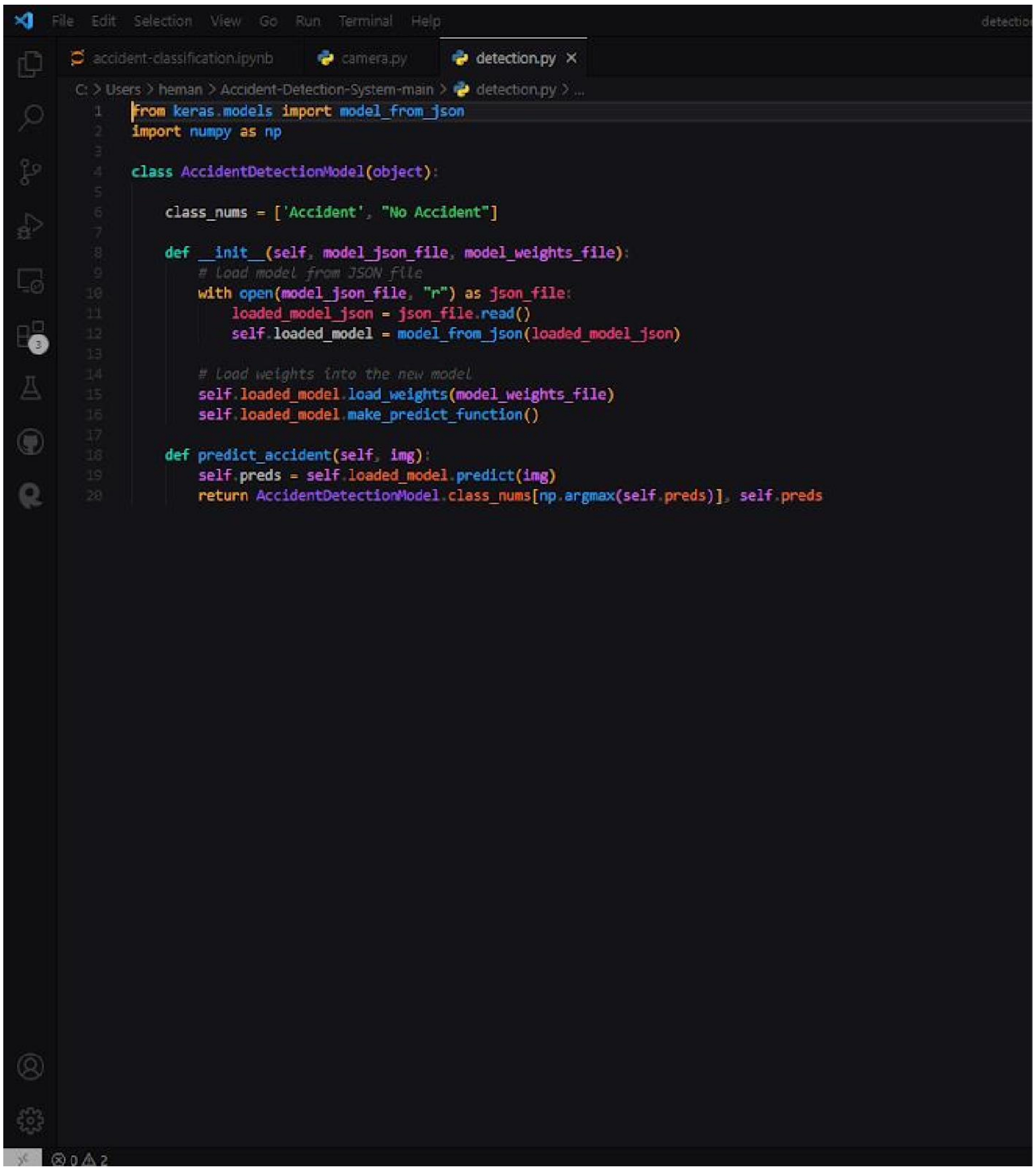
Pred: Accident actl:Accident
Pred: Non Accident actl:Non Accident
Pred: Accident actl:Accident
Pred: Non Accident actl:Non Accident
Pred: Non Accident actl:Non Accident
Pred: Accident actl:Accident

A 2.6 Code for CCTV accident detection


File Edit Selection View Go Run Terminal Help
camera.py - Visual Studio Co
accident-classification.ipynb
camera.py
C: > Users > heman > Accident-Detection-System-main > camera.py > ...
⊗ 0 ⚠ 2

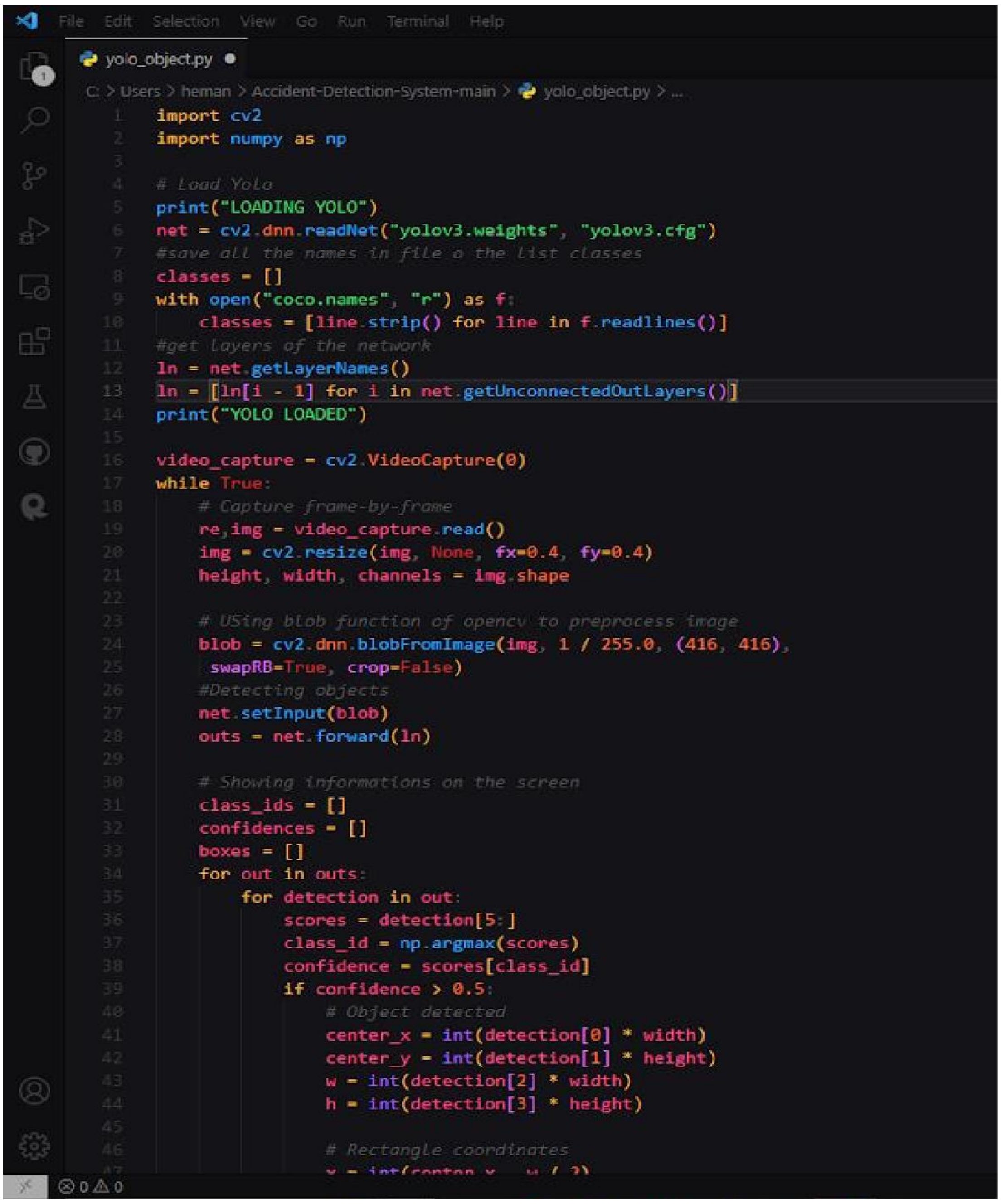
```python
import cv2
from detection import AccidentDetectionModel
import numpy as np
import os
# import smtplib



model = AccidentDetectionModel("model.json", 'model_weights.h5')
font = cv2.FONT_HERSHEY_SIMPLEX

def startapplication():
    video = cv2.VideoCapture('cars.mp4') # for camera use video = cv2.VideoCapture(0)
    #video = cv2.VideoCapture(0)
    while True:
        ret, frame = video.read()
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        roi = cv2.resize(gray_frame, (250, 250))

        pred, prob = model.predict_accident(roi[np.newaxis, :, :])
        if(pred == "Accident"):
            prob = (round(prob[0][0]*100, 2))

#             to beep when alert:
            # if(prob > 90):
            #     print("say beep")
            #     # list of email_id to send the mail
            #     li = ["xxxxx@gmail.com", "yyyyy@gmail.com"]

            #     for dest in li:
            #         s = smtplib.SMTP('smtp.gmail.com', 587)
            #         s.starttls()
            #         s.login("sender_email_id", "sender_email_id_password")
            #         message = "Message_you_need_to_send"
            #         s.sendmail("sender_email_id", dest, message)
            #         s.quit()
            #     break

            cv2.rectangle(frame, (0, 0), (280, 40), (0, 0, 0), -1)
            cv2.putText(frame, pred+" "+str(prob), (20, 30), font, 1, (255, 255, 0), 2)

        if cv2.waitKey(33) & 0xFF == ord('q'):
            return
        cv2.imshow('Video', frame)


if __name__ == '__main__':
    startapplication()
```

A 2.7

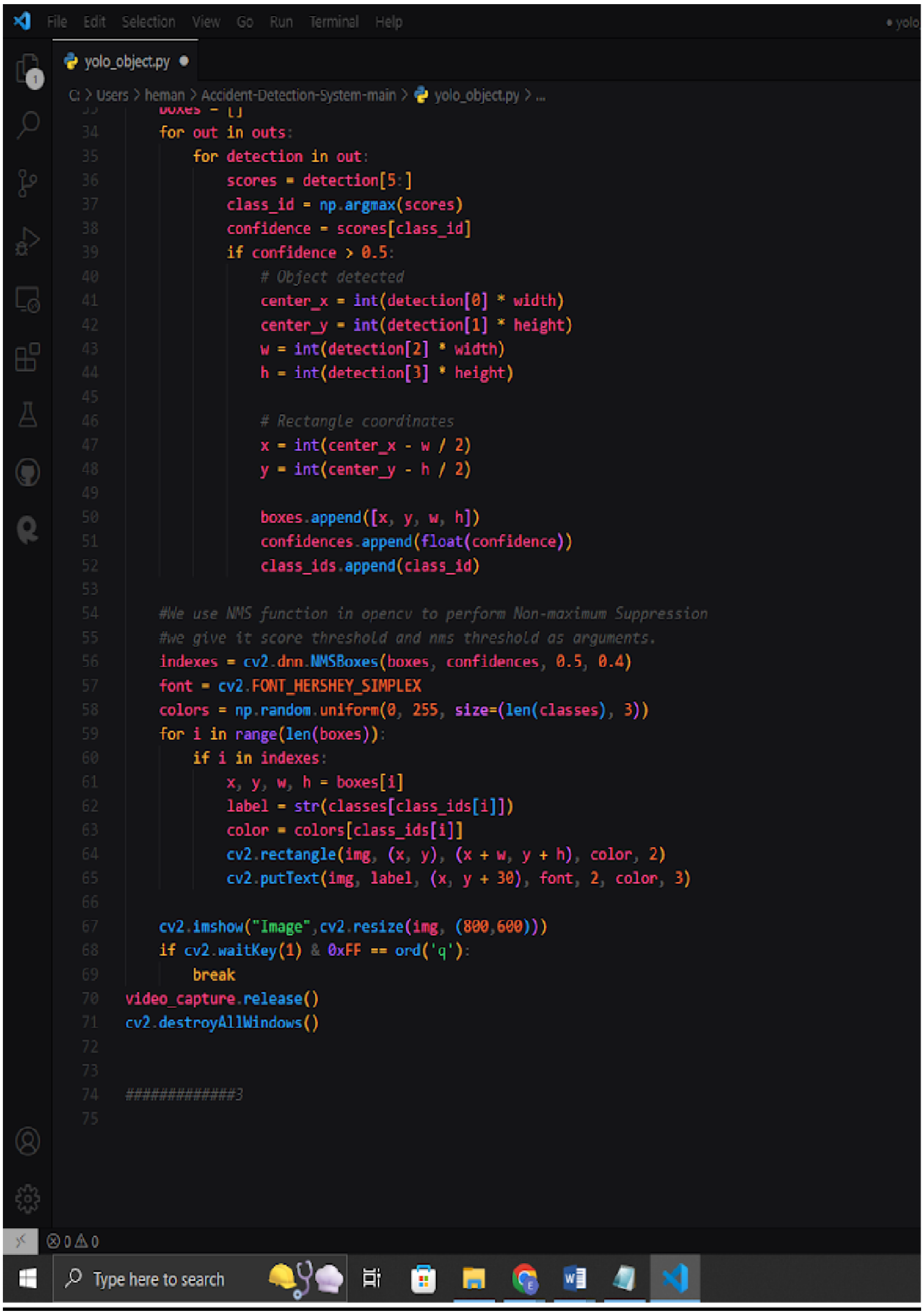

File Edit Selection View Go Run Terminal Help
accident-classification.ipynb
camera.py
detection.py
C: > Users > heman > Accident-Detection-System-main > detection.py > ...

```python
from keras.models import model_from_json
import numpy as np

class AccidentDetectionModel(object):

    class_nums = ['Accident', "No Accident"]

    def __init__(self, model_json_file, model_weights_file):
        # Load model from JSON file
        with open(model_json_file, "r") as json_file:
            loaded_model_json = json_file.read()
            self.loaded_model = model_from_json(loaded_model_json)

        # Load weights into the new model
        self.loaded_model.load_weights(model_weights_file)
        self.loaded_model.make_predict_function()

    def predict_accident(self, img):
        self.preds = self.loaded_model.predict(img)
        return AccidentDetectionModel.class_nums[np.argmax(self.preds)], self.preds
```

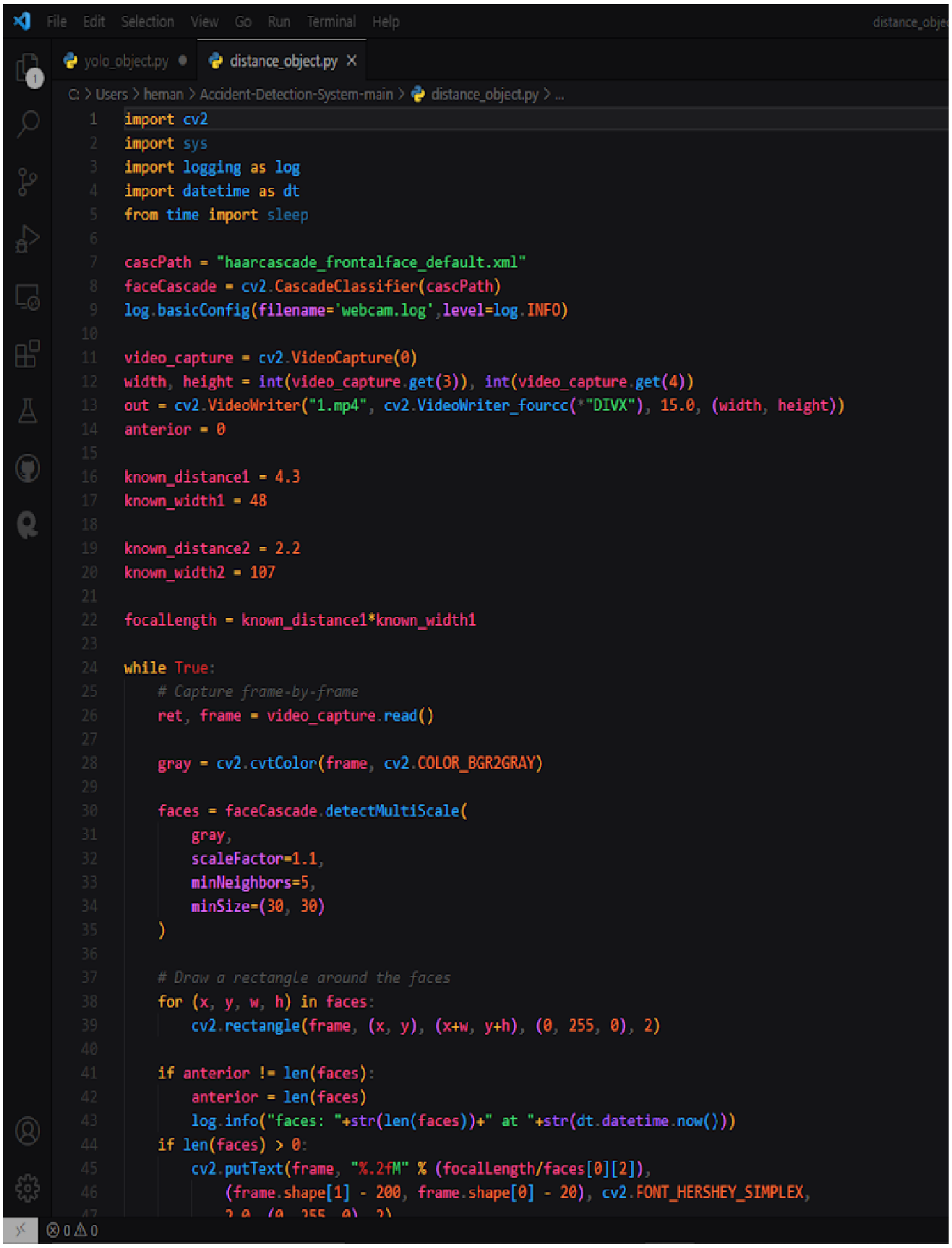
Code for YOLO object detection


File Edit Selection View Go Run Terminal Help
yolo_object.py
C: > Users > heman > Accident-Detection-System-main > yolo_object.py > ...

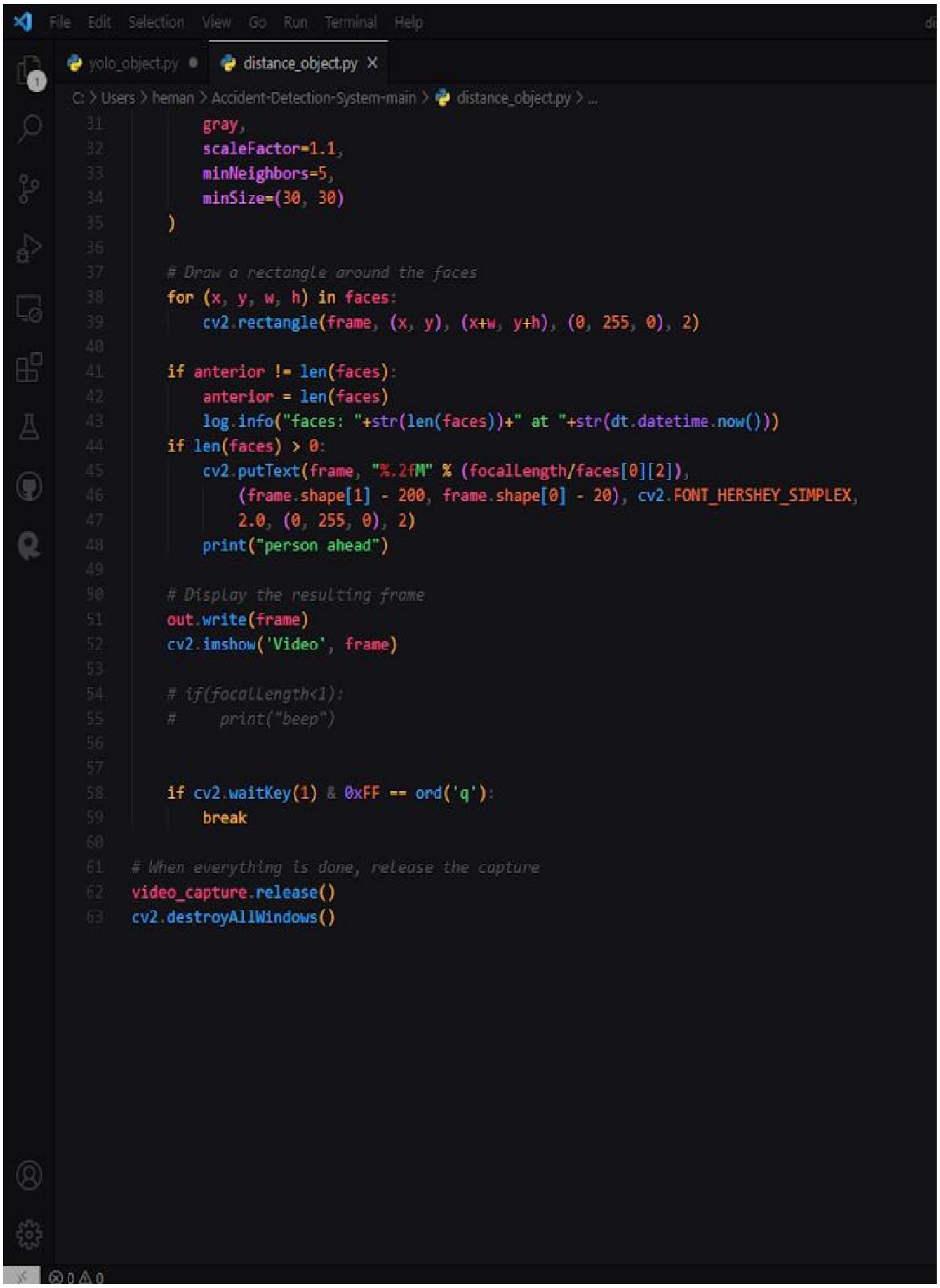
```python
import cv2
import numpy as np

# Load Yolo
print("LOADING YOLO")
net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
#save all the names in file o the list classes
classes = []
with open("coco.names", "r") as f:
    classes = [line.strip() for line in f.readlines()]
#get layers of the network
ln = net.getLayerNames()
ln = [ln[i - 1] for i in net.getUnconnectedOutLayers()]
print("YOLO LOADED")

video_capture = cv2.VideoCapture(0)
while True:
    # Capture frame-by-frame
    re,img = video_capture.read()
    img = cv2.resize(img, None, fx=0.4, fy=0.4)
    height, width, channels = img.shape

    # USing blob function of opencv to preprocess image
    blob = cv2.dnn.blobFromImage(img, 1 / 255.0, (416, 416),
     swapRB=True, crop=False)
    #Detecting objects
    net.setInput(blob)
    outs = net.forward(ln)

    # Showing informations on the screen
    class_ids = []
    confidences = []
    boxes = []
    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5:
                # Object detected
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width)
                h = int(detection[3] * height)

                # Rectangle coordinates
```


File Edit Selection View Go Run Terminal Help
yolo_object.py
C: > Users > heman > Accident-Detection-System-main > yolo_object.py > ...
Type here to search

```python
    boxes = []
    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5:
                # Object detected
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width)
                h = int(detection[3] * height)

                # Rectangle coordinates
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)

                boxes.append([x, y, w, h])
                confidences.append(float(confidence))
                class_ids.append(class_id)

    #We use NMS function in opencv to perform Non-maximum Suppression
    #we give it score threshold and nms threshold as arguments.
    indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
    font = cv2.FONT_HERSHEY_SIMPLEX
    colors = np.random.uniform(0, 255, size=(len(classes), 3))
    for i in range(len(boxes)):
        if i in indexes:
            x, y, w, h = boxes[i]
            label = str(classes[class_ids[i]])
            color = colors[class_ids[i]]
            cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)
            cv2.putText(img, label, (x, y + 30), font, 2, color, 3)

    cv2.imshow("Image",cv2.resize(img, (800,600)))
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
video_capture.release()
cv2.destroyAllWindows()


#############3

```

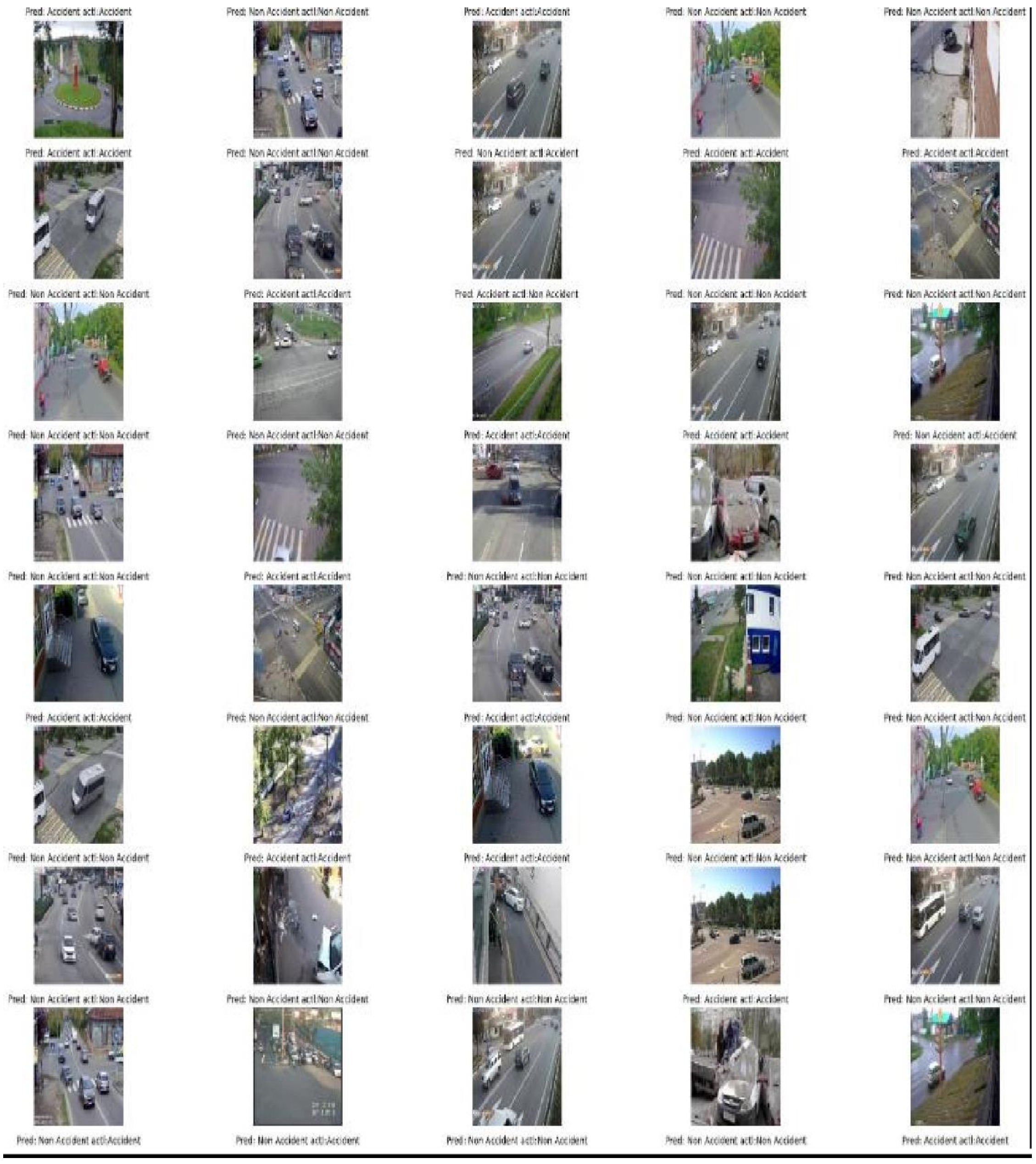

File Edit Selection View Go Run Terminal Help
distance_obje
yolo_object.py
distance_object.py
C: > Users > heman > Accident-Detection-System-main > distance_object.py > ...
⊗ 0 ⚠ 0

```python
import cv2
import sys
import logging as log
import datetime as dt
from time import sleep

cascPath = "haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascPath)
log.basicConfig(filename='webcam.log',level=log.INFO)

video_capture = cv2.VideoCapture(0)
width, height = int(video_capture.get(3)), int(video_capture.get(4))
out = cv2.VideoWriter("1.mp4", cv2.VideoWriter_fourcc(*"DIVX"), 15.0, (width, height))
anterior = 0

known_distance1 = 4.3
known_width1 = 48

known_distance2 = 2.2
known_width2 = 107

focalLength = known_distance1*known_width1

while True:
    # Capture frame-by-frame
    ret, frame = video_capture.read()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.1,
        minNeighbors=5,
        minSize=(30, 30)
    )

    # Draw a rectangle around the faces
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

    if anterior != len(faces):
        anterior = len(faces)
        log.info("faces: "+str(len(faces))+" at "+str(dt.datetime.now()))
    if len(faces) > 0:
        cv2.putText(frame, "%.2fM" % (focalLength/faces[0][2]),
            (frame.shape[1] - 200, frame.shape[0] - 20), cv2.FONT_HERSHEY_SIMPLEX,
            2.0, (0, 255, 0), 2)
```


File
Edit
Selection
View
Go
Run
Terminal
Help
yolo_object.py
distance_object.py
C: > Users > heman > Accident-Detection-System-main > distance_object.py > ...

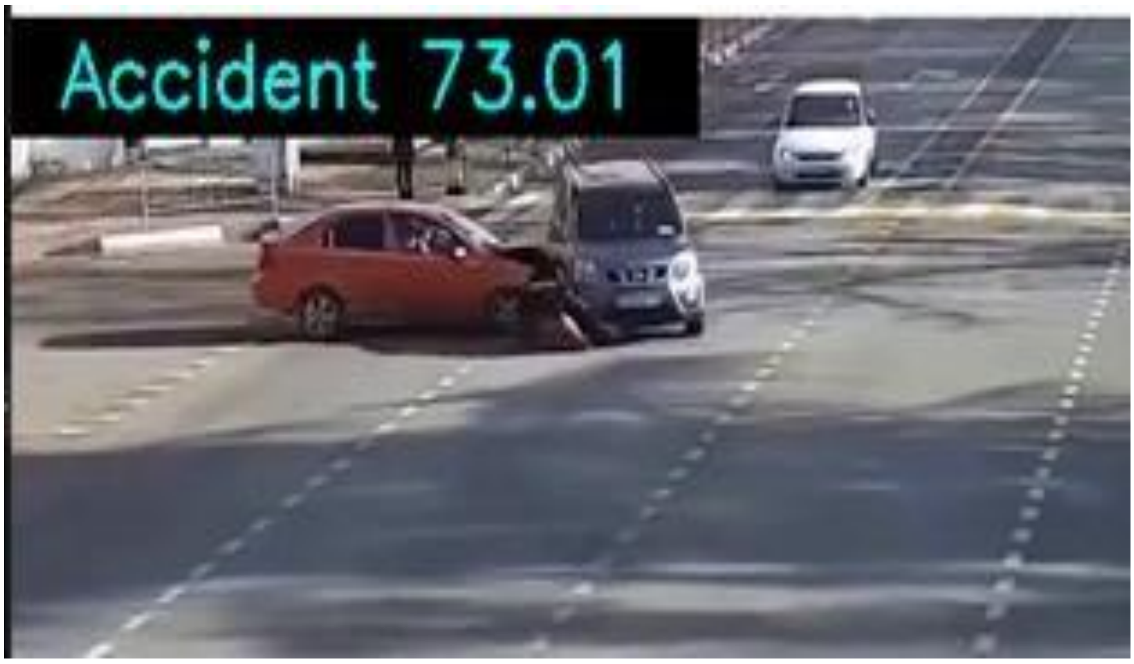
```
            gray,
            scaleFactor=1.1,
            minNeighbors=5,
            minSize=(30, 30)
        )

        # Draw a rectangle around the faces
        for (x, y, w, h) in faces:
            cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

        if anterior != len(faces):
            anterior = len(faces)
            log.info("faces: "+str(len(faces))+" at "+str(dt.datetime.now()))
        if len(faces) > 0:
            cv2.putText(frame, "%.2fM" % (focalLength/faces[0][2]),
                (frame.shape[1] - 200, frame.shape[0] - 20), cv2.FONT_HERSHEY_SIMPLEX,
                2.0, (0, 255, 0), 2)
            print("person ahead")

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

        # if(focalLength<1):
        #     print("beep")


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

    # When everything is done, release the capture
    video_capture.release()
    cv2.destroyAllWindows()
```

Output Module



Pred: Accident actl:Accident
Pred: Non Accident actl:Non Accident
Pred: Accident actl:Accident
Pred: Non Accident actl:Non Accident
Pred: Non Accident actl:Non Accident
Pred: Accident actl:Accident
Pred: Non Accident actl:Non Accident
Pred: Non Accident actl:Accident
Pred: Accident actl:Accident
Pred: Accident actl:Accident
Pred: Non Accident actl:Non Accident
Pred: Accident actl:Accident
Pred: Accident actl:Non Accident
Pred: Non Accident actl:Non Accident
Pred: Non Accident actl:Non Accident
Pred: Non Accident actl:Non Accident
Pred: Non Accident actl:Non Accident
Pred: Accident actl:Accident
Pred: Accident actl:Accident
Pred: Non Accident actl:Accident
Pred: Non Accident actl:Non Accident
Pred: Accident actl:Accident
Pred: Non Accident actl:Non Accident
Pred: Non Accident actl:Non Accident
Pred: Non Accident actl:Non Accident
Pred: Accident actl:Accident
Pred: Non Accident actl:Non Accident
Pred: Accident actl:Accident
Pred: Non Accident actl:Non Accident
Pred: Non Accident actl:Non Accident
Pred: Non Accident actl:Non Accident
Pred: Accident actl:Accident
Pred: Accident actl:Accident
Pred: Non Accident actl:Non Accident
Pred: Non Accident actl:Non Accident
Pred: Non Accident actl:Non Accident
Pred: Non Accident actl:Non Accident
Pred: Non Accident actl:Non Accident
Pred: Accident actl:Accident
Pred: Non Accident actl:Non Accident
Pred: Non Accident actl:Accident
Pred: Non Accident actl:Accident
Pred: Non Accident actl:Non Accident
Pred: Non Accident actl:Non Accident
Pred: Accident actl:Accident

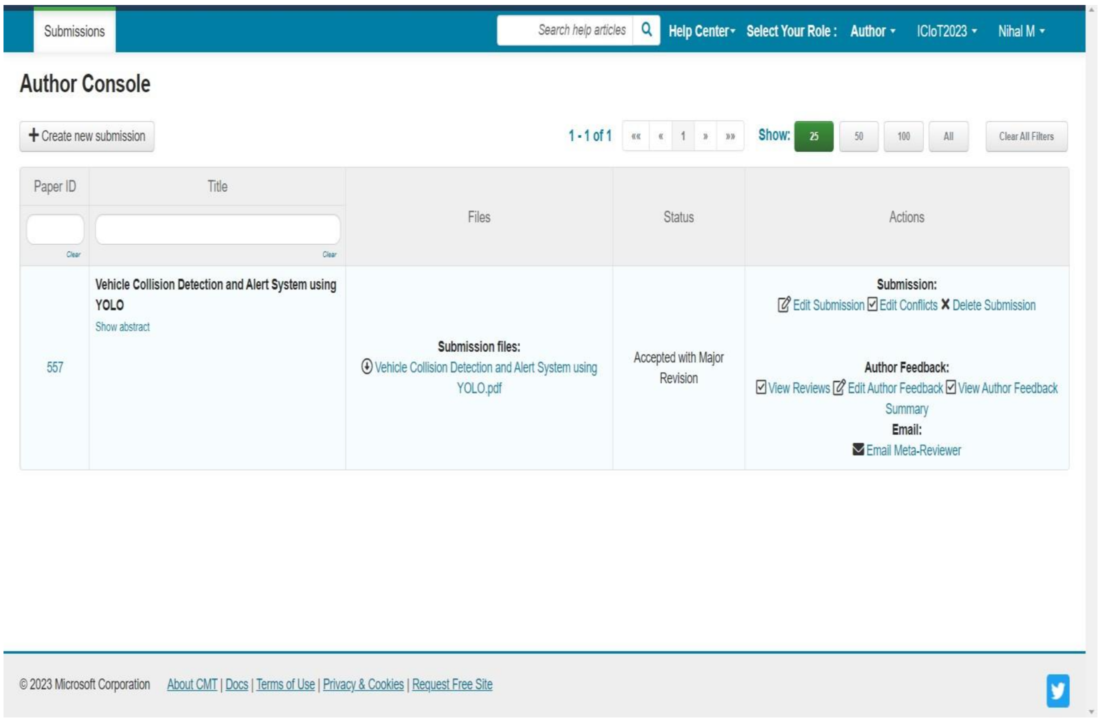
CNN Model Output



Detection

PAPER PUBLICATION STATUS

We submitted and got acceptance of our research paper for publication at ICIOT conference in the month of April 2023



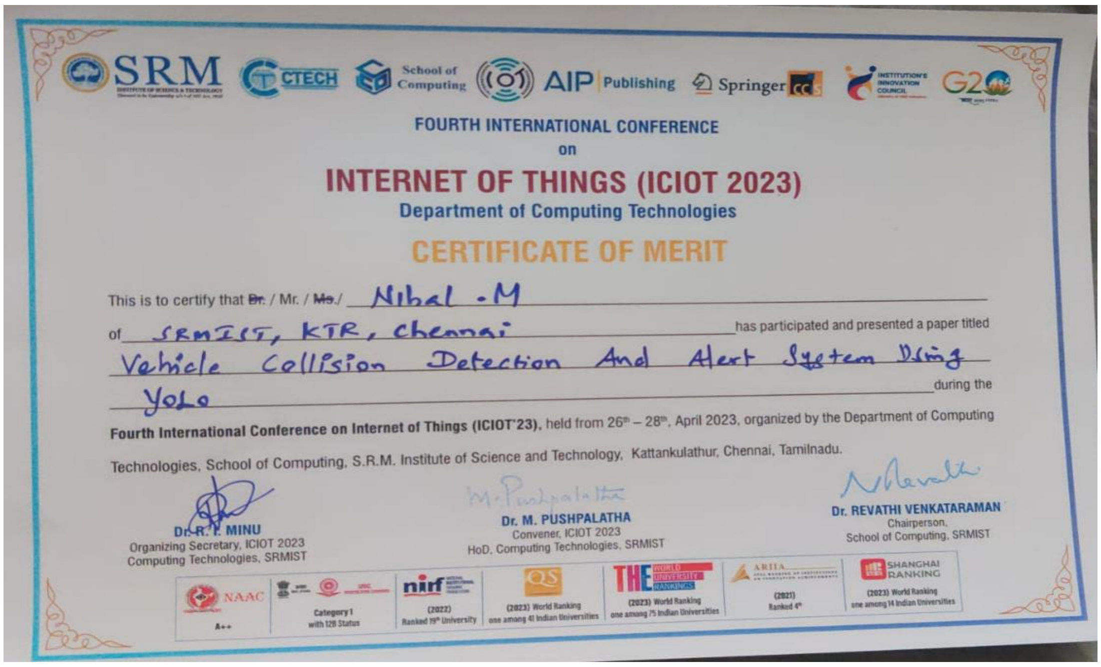
Submissions | Search help articles | Help Center | Select Your Role : Author | ICIoT2023 | Nihal M

Author Console

+ Create new submission | 1 - 1 of 1 | Show: 25 50 100 All | Clear All Filters

Paper ID	Title	Files	Status	Actions
557	**Vehicle Collision Detection and Alert System using YOLO** Show abstract	**Submission files:** Vehicle Collision Detection and Alert System using YOLO.pdf	Accepted with Major Revision	**Submission:** Edit Submission Edit Conflicts ✗ Delete Submission **Author Feedback:** View Reviews Edit Author Feedback View Author Feedback Summary **Email:** Email Meta-Reviewer

© 2023 Microsoft Corporation About CMT | Docs | Terms of Use | Privacy & Cookies | Request Free Site



FOURTH INTERNATIONAL CONFERENCE

on

INTERNET OF THINGS (ICIOT 2023)

Department of Computing Technologies

CERTIFICATE OF MERIT

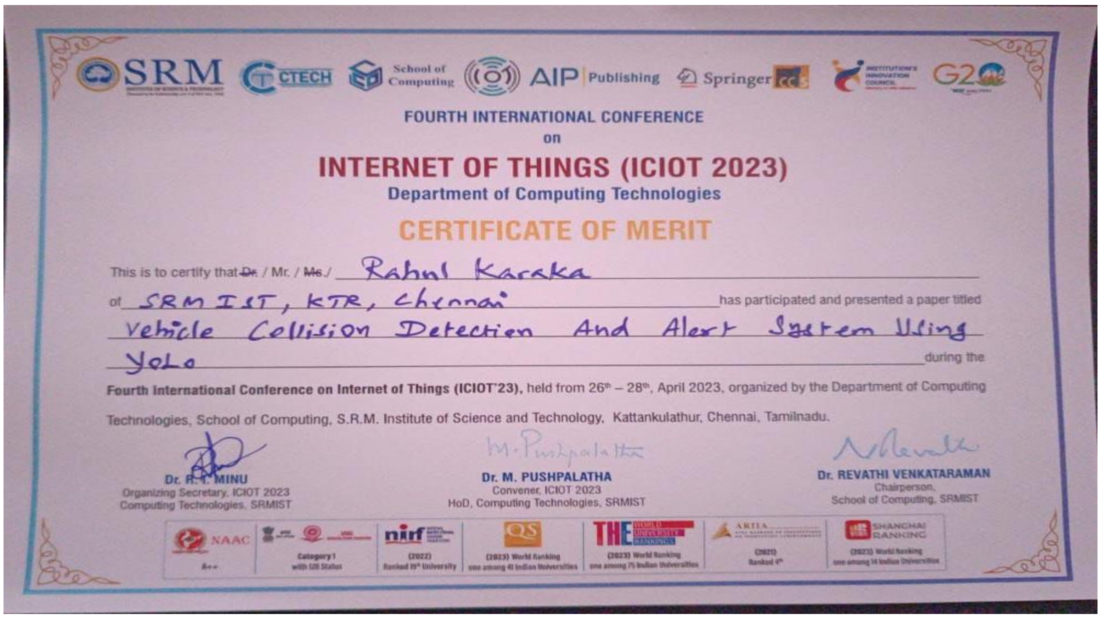
This is to certify that ~~Dr.~~ / Mr. / ~~Ms.~~/ Nibal .M

of SRMIST, KTR, Chennai has participated and presented a paper titled Vehicle Collision Detection And Alert System Using YoLo during the **Fourth International Conference on Internet of Things (ICIOT'23),** held from 26th – 28th, April 2023, organized by the Department of Computing Technologies, School of Computing, S.R.M. Institute of Science and Technology, Kattankulathur, Chennai, Tamilnadu.

Dr. R. I. MINU
Organizing Secretary, ICIOT 2023
Computing Technologies, SRMIST

M. Pushpalatha
Dr. M. PUSHPALATHA
Convener, ICIOT 2023
HoD, Computing Technologies, SRMIST

Dr. REVATHI VENKATARAMAN
Chairperson,
School of Computing, SRMIST



FOURTH INTERNATIONAL CONFERENCE

on

INTERNET OF THINGS (ICIOT 2023)

Department of Computing Technologies

CERTIFICATE OF MERIT

This is to certify that ~~Dr.~~ / Mr. / ~~Ms.~~/ Rahul Karaka

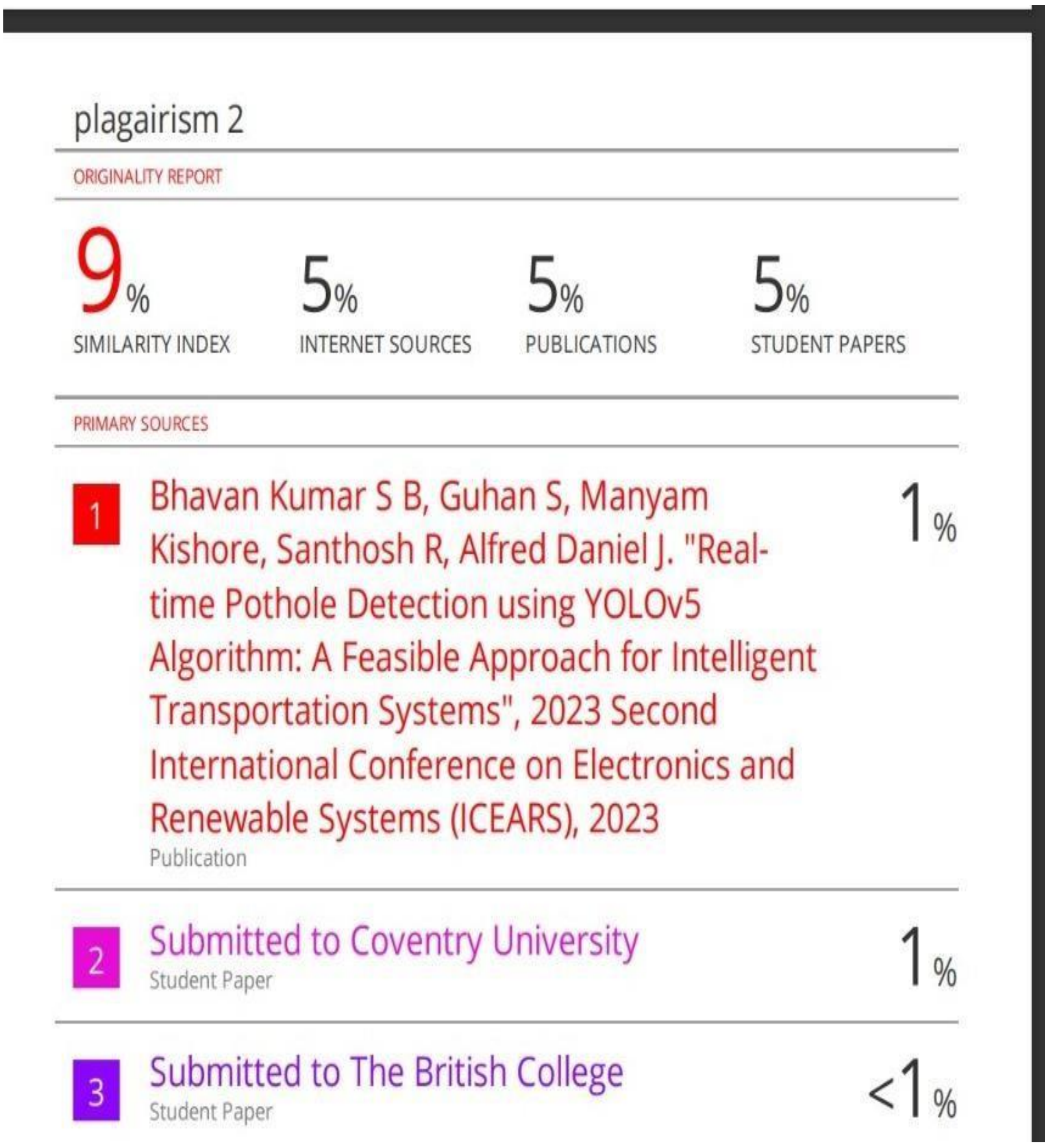
of SRM IST, KTR, Chennai has participated and presented a paper titled Vehicle Collision Detection And Alert System Using YoLo during the **Fourth International Conference on Internet of Things (ICIOT'23),** held from 26th – 28th, April 2023, organized by the Department of Computing Technologies, School of Computing, S.R.M. Institute of Science and Technology, Kattankulathur, Chennai, Tamilnadu.

Dr. R. I. MINU
Organizing Secretary, ICIOT 2023
Computing Technologies, SRMIST

M. Pushpalatha
Dr. M. PUSHPALATHA
Convener, ICIOT 2023
HoD, Computing Technologies, SRMIST

Dr. REVATHI VENKATARAMAN
Chairperson,
School of Computing, SRMIST

PLAGIARISM REPORT



plagairism 2

ORIGINALITY REPORT

9%	5%	5%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Bhavan Kumar S B, Guhan S, Manyam Kishore, Santhosh R, Alfred Daniel J. "Real-time Pothole Detection using YOLOv5 Algorithm: A Feasible Approach for Intelligent Transportation Systems", 2023 Second International Conference on Electronics and Renewable Systems (ICEARS), 2023 Publication	1%
2	Submitted to Coventry University Student Paper	1%
3	Submitted to The British College Student Paper	<1%

S R M I N S T I T U T E O F S C I E N C E A N D T E C H N O L O G Y **(Deemed to be University u/ s 3 of UGC Act, 1956)**		
Office of Controller of Examinations		
REPORT FOR PLAGIARISM CHECK ON THE DISSERTATION/PROJECT REPORTS FOR UG/PG PROGRAMMES **(To be attached in the dissertation/ project report)**		
1	Name of the Candidate **(IN BLOCK LETTERS)**	M.NIHAL
2	Address of the Candidate	2-2-24/A/1 Central excise colony,Bagh Amberpet,Hyderabad,Telangana **Mobile Number:7207444108**
3	Registration Number	RA1911003010319
4	Date of Birth	10 May 2002
5	Department	Computer Science and Engineering
6	Faculty	Engineering and Technology, School of Computing
7	Title of the Dissertation/Project	Vehicle Collision Detection and Alert System using YOLO

8	Whether the above project /dissertation is done by	group : a) If the project/ dissertation is done in group, then how many students together completed the project: 2 (Two) b) Mention the Name & Register number of other candidates : Rahul Karaka(RA1911003010340)
9	Name and address of the Supervisor / Guide	Dr. K.Pradeep Mohan Kumar Associate Professor Department of Computer Science and Engineering SRM Institute of Science and Technology Kattankulatur - 603 203. **Mail ID:** pradeepk@srmist.edu.in **Mobile Number**: 9362447221
10	Name and address of Co-Supervisor / Co- Guide (if any)	NIL

11	Software Used	Turnitin		
12	Date of Verification	05-05-2023		
13	**Plagiarism Details: (to attach the final report from the software)**			
Chapter	**Title of the Chapter**	**Percentage of similarity index (including self citation)**	**Percentage of similarity index (Excluding self citation)**	**% of plagiarism after excluding Quotes, Bibliography, etc.,**

1	INTRODUCTION	1%	1%	1%
2	LITERATURE SURVEY	1%	1%	1%
3	TECHNICAL SPECIFICATIONS	2%	2%	2%
4	ARCHITECTURE	1%	1%	1%
5	MODULES	2%	2%	2%
6	PROJECT DEMONSTRATION AND RESULTS	0%	0%	0%
7	CONCLUSION	0%	0%	0%
8	FUTURE ENHANCEMENTS	0%	0%	0%
9				
10				
Appendices		2%	2%	2%
I / We declare that the above information have been verified and found true to the best of my / our knowledge.				
Signature of the Candidate		**Name & Signature of the Staff (Who uses the plagiarism check software)**		

Name & Signature of the Supervisor/ Guide	**Name & Signature of the Co-Supervisor/Co-Guide**
Name & Signature of the HOD	