كليـــــــــة العلوم
السملالية - مراكش
FACULTÉ DES SCIENCES
MARRAKECH - SEMLALIA

MARRAKECH
جامعة القاضي عياض
UNIVERSITÉ CADI AYYAD

GRADUATION PROJECT

To obtain the Bachelor degree in Mathematical and Computer Science (MCS) at the Faculty of Sciences Semlalia in Marrakech.

AI-Based Route Planning and Traffic Forecasting for Sustainable Transport

Internship carried out from the 24th of April 2024 until the 29th of June 2024

Defense Scheduled for June 29, 2024

Realized by :

M'KOUKA Btissam

BOUMOULA Abdelouafi

Supervised by :

Pr. Abdellah NABOU, UCA

Jury Members :

Pr. Abdelwahed EL HASSAN, UCA
Pr. Abdellah NABOU, UCA

Session Year : 2023-2024

Acknowledgements

First and foremost, we would like to express our deepest gratitude to ALLAH for granting us the patience, strength, and blessings to reach this significant milestone. The magnitude of our accomplishment would not have been possible without the guidance and support of numerous individuals who played pivotal roles in our journey.

This may be the most challenging part of our writing this report. Not because I am short on people to acknowledge – quite the opposite. Before we began, we want to thank every person who has impact our life indirectly, positively or negatively. Thank you, you contribute to the persons who we are today.

Our heartfelt gratitude goes to our parents, for being the only persons to love us no matter what we do.

We are also grateful to our supervisors and examiners, Dr. Abdellah Nabou and Dr. Abdelwahed El Hassan, for their passion, knowledge, and thorough examination of our research. Their constructive critique, insightful ideas, and rigorous attention to detail have unquestionably improved the depth and rigor of our study. We value their commitment to academic success and their role in molding us into better researchers.We have been undeservedly lucky throughout our project to work with them who are more talented than we are and to get to steal their wisdom and gracefulness and pass it off as our.

My sincere thanks go to our team, this project began as a big, messy thing and required more than just one hand to chisel something comprehensible out of it.

And finally, we are appreciative for the lessons learned, memories formed, and development we have experienced along the way. May our achievements serve as proof of the strength of teamwork, perseverance, and the limitless possibilities that higher education provides.

Abstract

This graduation project focuses on the development and implementation of an AI-based system for detecting emergency vehicles and giving them priority at traffic lights in emergency situations. The primary objective was to create an object detection system for emergency vehicles using deep learning techniques, specifically training the YOLOv8 model on a customized dataset of Moroccan vehicles. The system's accuracy and relevance to local traffic scenarios were ensured through rigorous training and validation processes. In addition, a traffic congestion prediction model was developed using historical traffic data, leveraging Long Short-Term Memory (LSTM) networks conbined with XGBoost model and the accuracy and efficiency were enhanced and reduce prediction error.

The system was successfully simulated using Arduino Uno, sound detectors, and camera for real-time detection and traffic management. Throughout the project, we encountered various challenges, including system integration and data preparation. However, we successfully developed a robust emergency vehicle detection and traffic light priority system. The fulfillment of this project is an initial step toward a more comprehensive and omnipresent traffic management system, with the ultimate goal of creating a safer and more efficient transportation environment by reducing congestion and traffic accidents.

Table of Contents

List of Figures

Chapter 1

General presentation of the project

Whithin this chapter, we will establish the contextual framework of the project and identify the problem. Furthermore, we will not only address the challenges encountered but also provide comprehensive solutions for overcoming these obstacles. Subsequently, we will explore the organization and workflow employed throughout this project.

1.1 Introduction

Vehicular traffic is endlessly increasing everywhere in the world and can cause terrible traffic congestion at intersections. Most of the traffic lights today feature a fixed green light sequence, therefore the green light sequence is determined without taking the presence of the emergency vehicles into account. Therefore, emergency vehicles such as ambulances, police cars, fire engines, etc. must wait in traffic at an intersection which delays their arrival at their destination causing loss of lives and property.In Morocco, especially in urban areas like Marrakesh, the timeliness of ambulance responses is a significant concern. Severe traffic congestion often prevents emergency services from reaching accident sites promptly. In 2022, Morocco recorded 3,499 road deaths, including fatalities from delayed emergency responses due to traffic congestion and logistical issues [7]. Although specific statistics on fatalities solely due to late ambulance responses are not available, it is clear that congestion and infrastructure challenges significantly contribute to delays and adverse outcomes.

In this section, we present an overview of our project that focuses on modelling and simulating a traffic managing by giving emergency vehicles priority in emergency situations.This means that in addition to detecting vehicles, managing traffic lights, we also optimize a model to predict traffic congestion this prediction has led to a growing research area, especially of machine learning of artificial intelligence (AI).

Throughout this report, we will discuss the AI algorithms and techniques we used to make intelligent decisions and automate tasks in the smart cities, using multiple AI algorithms within the same context can often lead to different results because each algorithm has its own strengths, weaknesses, and underlying principles. This comparative analysis allows us to identify which algorithms perform better in terms of accuracy, efficiency, or other relevant metrics and determine which ones yielded the most promising results because the goal is to select the most suitable algorithm or combination of algorithms that can provide improved results, and finally we will also explore the simulation and modeling methods we employed to evaluate the system's performance.

The simulation aspect of the project plays a Key role in testing and refining the AI model. Through simulation, various scenarios and configurations can be evaluated, allowing for the identification of potential issues and the optimization of system performance.

Overall, this project aims to enhance traffic management systems by incorporating advanced algorithms and sensors. Through simulation and modeling, the focus will be on specific functionalities such as detecting ambulances, giving them priority in emergency situations, and predicting traffic congestion to improve overall efficiency and safety on the roads.

1.2 Problematic

Efficient traffic management is critical, especially in urban areas where congestion can significantly impede the movement of emergency vehicles such as ambulances, police and fire truck. Traditional traffic systems face numerous challenges in providing timely and safe passage for these vehicles, often resulting in delayed emergency responses and increased risk to patients. This project aims to address the problematic aspects of current traffic management systems and propose solutions leveraging advanced technologies.

Key issues include :

— Emergency Vehicle Detection : Traditional traffic systems lack the capability to accurately and promptly detect emergency vehicles. This results in inefficient traffic control and delayed priority for ambulances, potentially leading to critical delays in emergency response.

— Dynamic Traffic Light Management : Existing traffic light systems are typically static and do not adapt in real-time to the presence of emergency vehicles. This lack of adaptability can cause significant delays for emergency vehicles, which need the quickest possible route through intersections.

— Traffic Congestion Prediction : Without accurate traffic congestion prediction, it is challenging to manage and optimize traffic flow effectively. This can lead to bottlenecks and increased delays, not only for emergency vehicles but for all road users.

1.3 Challenges and objectives

As beginners in the field of AI research, we encountered a number of difficulties that we overcame in order to finish our project. The first was a lack of information needed to comprehend the task. Despite the fact that it's a recent, complicated project that demands plenty of information, it takes a lot of work to do within the given two months, which is insufficient for such a subject. However, we did enjoy ourselves and gained practical experience using different Python tools and conducting a study.

The primary objective of this project is to develop an intelligent traffic management system that enhances the efficiency and safety of urban transportation by real-time detection and prioritizing emergency vehicles, dynamic traffic light management, traffic congestion prediction, create a predictive model for traffic congestion using historical traffic data and machine learning techniques, such as Long Short-Term Memory (LSTM) networks combined with XGboost additionally, a system integration and simulation to integrate the detection, prioritization, into a cohesive system and simulate its functionality using ArduinoUno, sound detectors to detect siren of emergency situation, and cameras to manage traffic in real-time.

1.4 Planning and tools

The provided Gantt Chart Graph offers a comprehensive overview of the project's timeline. The chart displays a well-structured plan, enabling a clear understanding of the project's progression. Overall, it is a well-structured and comprehensive chart that provides a solid foundation for successful project execution.

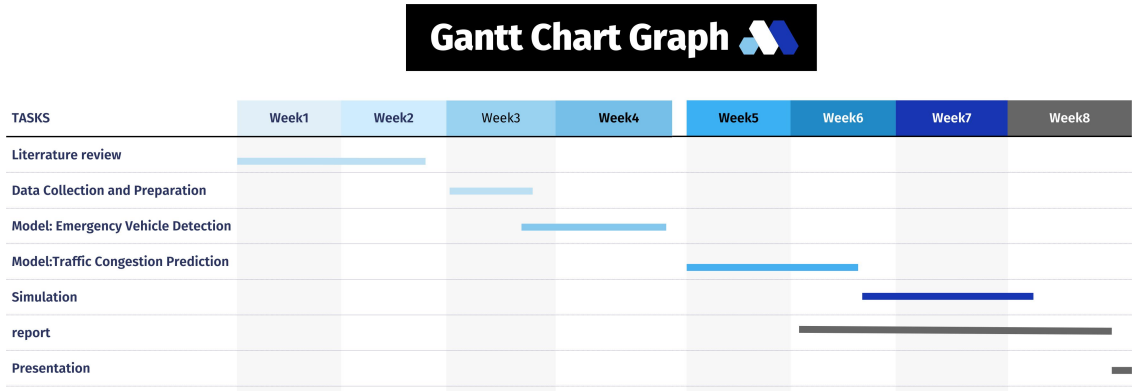


FIGURE 1.1 – Timeline representation

The project's timeline is structured into several critical phases, each presenting its unique challenges and requirements. The initial phase involves an extensive Literature Review, necessitating thorough exploration on platforms like Google Scholar to grasp the intricacies of the topic and identify gaps for proposing innovative solutions. Following this, the Data Collection and Preparation phase emerges as a significant time investment, particularly challenging for obtaining precise, country-specific data, especially concerning emergency vehicle patterns, which are sporadic and rare. This phase is crucial as it lays the foundation for robust model development.

Subsequently, the focus shifts to Model Development, starting with Emergency Vehicle Detection. This stage demands meticulous algorithm selection and parameter tuning to ensure accurate and reliable detection under varied conditions. Simultaneously, developing models for predicting Traffic Congestion requires sophisticated data analysis techniques and modeling approaches to optimize urban mobility solutions effectively. Each phase requires dedicated effort and attention to detail to achieve comprehensive research outcomes and ensure the project's success.

Chapter 2

Technologies and Approaches for Intelligent Traffic Management Systems

Efficient traffic control is essential for smooth driving conditions and pollution reduction. In addition emergency vehicles such as ambulances, fire trucks, and police, provide emergency response to help save lives and ensure security in our society. During emergency conditions, the response time of the emergency vehicle is of significant importance and, in some cases, a few seconds of time saving may save lives. For example, in the case of an ambulance, each minute of delay can reduce the survival rate of patients with cardiac arrest by 7-10 per cent [5]. This project aims to develop an intelligent traffic control system that detects emergency vehicles, their situation (Emergency or not) and predicts traffic congestion. The system uses image processing algorithms to prioritize emergency vehicles by switching traffic lights to green and sensor detector to analyze siren. By integrating these technologies, the system enhances traffic flow, reduces congestion, and ensures the efficient passage of emergency vehicles, significantly improving urban traffic management and public safety.

2.1 Implemented Technologies

2.1.1 Machine learning

Machine learning is a branch of artificial intelligence that focuses on developing algorithms and models capable of learning from data and making predictions or taking actions without being explicitly programmed. It involves the study and development of computational systems that can automatically improve their performance through experience. Machine learning algorithms learn patterns and relationships in the data, enabling them to make accurate predictions or decisions on new, unseen data. This is achieved through a training process where models are exposed to labeled or unlabeled data, and they iteratively adjust their internal parameters to optimize their performance. Machine learning has a wide range of applications, including image and speech recognition, natural language processing, recommendation systems, fraud detection, and many more.

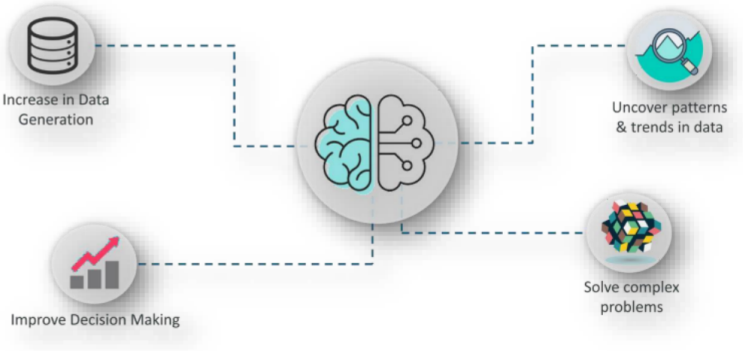


Figure 2.1 – The critical points in machine learning

In this project, machine learning plays a key role in analyzing traffic patterns, predicting congestion, detecting emergency vehicles, and enhancing image processing algorithms.

2.1.1.1 Recurrent Neural Network(RNN)

Recurrent Neural Network (RNN) is a type of neural network where the output from the previous step is fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other. However, in cases where it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember them. Thus, RNN came into existence, which solved this issue with the help of a hidden layer. The main and most important feature of RNN is its hidden state, which remembers some information about a sequence. The state is also referred to as memory state since it remembers the previous input to the network. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike in other neural networks.

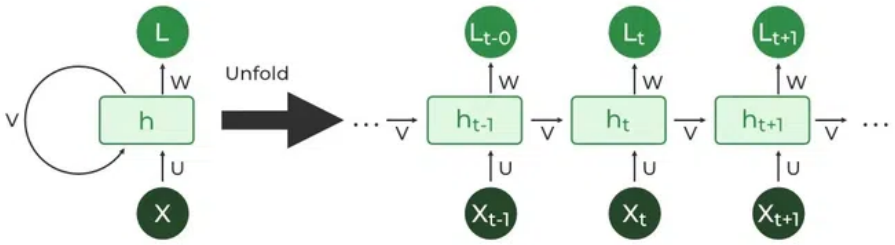


FIGURE 2.2 – Illustration of a recurrent neural network [1]

2.1.1.2 Long Short-Term Memory LSTM

A traditional RNN has a single hidden state that is passed through time, which can make it difficult for the network to learn long-term dependencies. LSTMs address this problem by introducing a memory cell, which is a container that can hold information for an extended period. LSTM networks are capable of learning long-term dependencies in sequential data. LSTMs can also be used in combination with other neural network architectures, and XGBOOST for prediction.

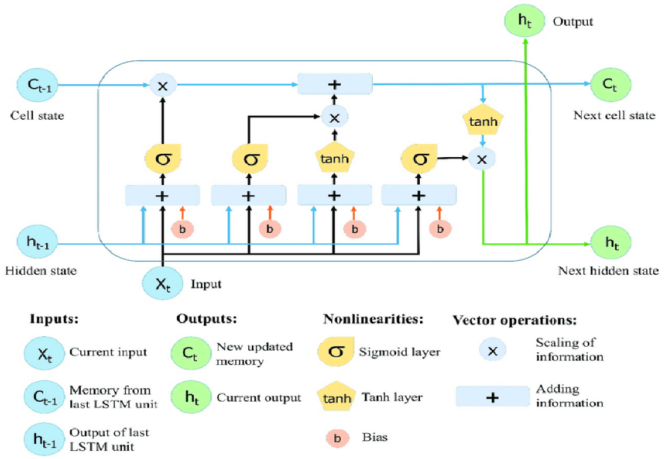


FIGURE 2.3 – Illustration of a recurrent neural network [2]

2.1.2 Artificial Neural Networks (ANN)

Artificial Neural Networks (ANNs) are computational models inspired by the structure and functioning of biological neural networks, particularly the human brain. ANNs are widely used in the field of machine learning

to solve complex problems involving pattern recognition, prediction, classification, and optimization. An ANN consists of interconnected artificial neurons, also known as nodes or units, organized in layers. The three main types of layers in an ANN are the input layer, hidden layer(s), and output layer.

1. Input Layer : The input layer receives input data and passes it to the next layer. Each node in the input layer represents a feature or attribute of the input data.

2. Hidden Layer(s) : The hidden layer(s) receive input from the previous layer and perform computations on the data. These layers are called "hidden" because their outputs are not directly observed or interpreted by the external world. The number of hidden layers and the number of nodes in each layer can vary depending on the complexity of the problem and the network architecture.

3. Output Layer : The output layer provides the final output or prediction of the neural network. The number of nodes in the output layer is determined by the nature of the problem, such as binary classification (2 nodes) or multiclass classification (multiple nodes).

Each node in an ANN receives input signals from the nodes in the previous layer, performs a computation using an activation function, and produces an output. The activation function introduces non-linearity into the network, enabling it to learn complex patterns and make nonlinear decisions.

During the training phase, ANNs learn from labeled examples on training data to adjust the weights and biases associated with the connections between the nodes.

This process is often performed using optimization algorithms like backpropagation, which computes the gradient of a loss function with respect to the network parameters and updates them accordingly. The objective is to minimize the difference between the network's predictions and the expected outputs.

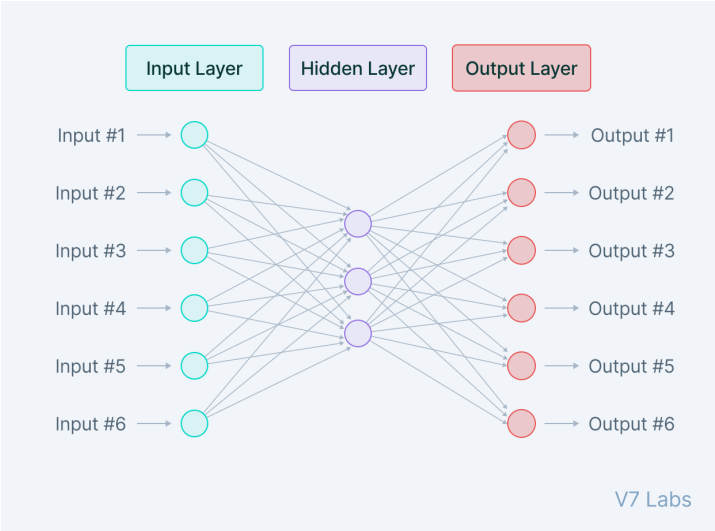


FIGURE 2.4 – The critical points in machine learning [3]

2.1.3 Image Processing

Before we start to process images, it's important for us to establish the dimensions of the image. The number of pixels does not always determine the size of the picture's height and width. On a digital computer, image processing is required for the advancement of image management. Over the last few years, they've grown at an incredible rate. There are a range of applications, from healthcare and entertainment to historic studies in the field of geographical sciences, for example, remote sensing.
The management of digital images, based on today's data culture, is a pillar of the Mixed Media System. Advanced signal processing techniques, together with picture-specific technologies, are part of digital image processing. The image as an example is a visual representation of something whereas a digital image is a binary representation of visual data.

•Image Enhancement :
The use of visual enhancement to enhance a photograph's image quality or to emphasize certain elements Examples are the enhancement of contrast, brightness adjustment, brightening, and noise reduction.

•Image Restoration :
Restoration of images is a method to prevent or reduce image degradation caused by capturing, transmission, and capacity limitation ; other activities include motion blur, defocus, noise reduction, and damaged or bad photo correction.

•Image Analysis :
The process of analyzing images involves exploiting techniques like segmentation, object recognition and tracking, feature extraction, pattern recognition, and image classification so that valuable information can be obtained from the pictures. Computer vision, healthcare imaging, and remote sensing are among the fields that use this significant technology.

2.1.4 Computer Vision

Computer vision is a field of study and research that focuses on enabling computers and machines to interpret, understand, and analyze visual data, such as images and videos, in a manner similar to human vision. It involves developing algorithms and techniques to extract meaningful information from visual inputs, including object recognition, image segmentation, motion tracking, and scene understanding. Computer vision systems leverage image processing, pattern recognition, and machine learning methods to detect and interpret visual patterns, shapes, and features. This technology finds applications in various domains, such as autonomous vehicles, robotics, healthcare, surveillance, and augmented reality. By enabling machines to "see" and comprehend visual data, computer vision opens up possibilities for tasks that require visual perception, enabling machines to interact with and understand the world in a more human-like way. For example, emergency vehicles detection by camera involves utilizing computer vision algorithms to analyze video feeds or images captured in order.

2.1.4.1 Convolutional Neural Networks (CNN)

CNN is considered as a type of artificial neural network architecture designed to process and understand visual data, such as images, it can automatically identify patterns, shapes, and objects within them. It does it by breaking down the image into smaller parts and analyzing them through layers of interconnected "neurons" that are capable to learn, extract features and make a deep understanding of what the images contain. Each neuron within these layers is responsible for detecting specific visual features, such as edges, textures, or shapes, at different levels of complexity.

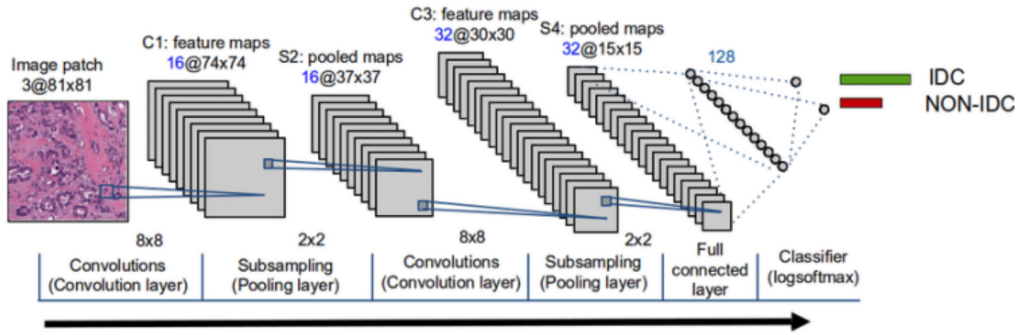


Figure 2.5 – CNN layers [4]

2.1.4.2 YOLO

YOLO (You Only Look Once) is a powerful family of deep learning (DL) neural network architectures for real-time object detection. It simplifies object detection by doing everything at once : classifying objects and placing bounding boxes around them in a single network run. This makes YOLO fast and efficient.

From a machine learning perspective, YOLO utilizes deep learning techniques, specifically convolutional neural networks (CNNs), to learn and detect objects in images and videos. It uses a single neural network architecture to directly predict bounding boxes and class probabilities for multiple objects in a single pass, hence the name "You Only Look Once." YOLO's training process involves optimizing the network's parameters based on annotated data, which is a fundamental aspect of machine learning.

Despite its strengths, YOLO may face challenges with detecting small objects accurately and handling overlapping or crowded scenes. However, advancements like YOLOv7 and YOLOv8 have introduced improvements in performance and addressed some of these limitations. YOLO has achieved remarkable performance in the context of detecting emergency vehicles, delivering excellent results by analyzing the live video feed from security cameras yolo can quickly identify and show accuracy.

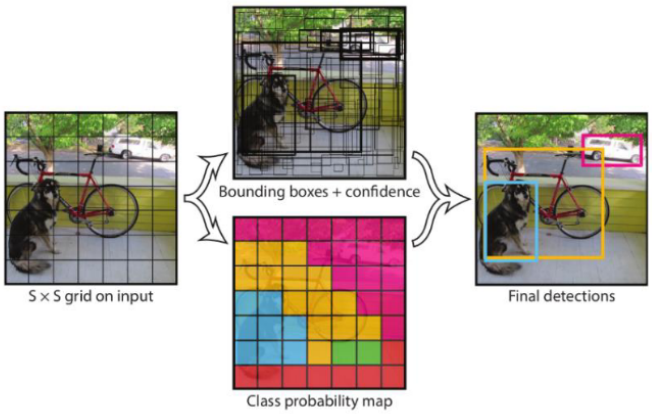


FIGURE 2.6 – YOLO object detection process [5]

2.1.4.2.1 The Importance of YOLO :

Other object detection methods, like regionalized neural networks (R-CNN), including other variants like fast R-CNN, focus on a specific region of the image and train each component separately. This process requires the R-CNN to classify 2000 regions per image, which makes it very time-consuming (47 seconds per individual test image). It therefore cannot be implemented in real time. This makes object detection networks such as R-CNN harder to optimize and slower than YOLO. The YOLO model is much faster (45 frames per second) and easier to optimize than previous algorithms, because it is based on an algorithm that uses only a single neural network to execute all elements of the task [8].

The model	Frames Per Second (FPS)	Real-time speed
Fast YOLO	155	YES
YOLO	45	YES
YOLO VGG-16	21	NO
FAST R-CNN	0.5	NO
FAST R-CNN VGG-16	7	NO
FAST R-CNN ZF	18	NO

FIGURE 2.7 – comparison between R-CNN and YOLO

Why Should We Use YOLOv8 ?

Here are a few main reasons why we should consider using YOLOv8 for next computer vision project :

1. YOLOv8 has a high rate of accuracy measured by Microsoft COCO and Roboflow 100.

2. YOLOv8 comes with a lot of developer-convenience features, from an easy-to-use CLI to a well-structured Python package.

3. There is a large community around YOLO and a growing community around the YOLOv8 model, meaning there are many people in computer vision circles who may be able to assist you when you need guidance.

YOLOv8 achieves strong accuracy on COCO. For example, the YOLOv8m model (the medium model) achieves a 50.2% mAP when measured on COCO. When evaluated against Roboflow 100, a dataset that specifically evaluates model performance on various task-specific domains, YOLOv8 scored substantially better than YOLOv5. Furthermore, the developer-convenience features in YOLOv8 are significant. As opposed to other models where tasks are split across many different Python files that you can execute, YOLOv8 comes with a CLI that makes training a model more intuitive. This is in addition to a Python package that provides a more seamless coding experience than prior models [9].

YOLOv8 Architecture : A Deep Dive

The YOLOv8 architecture for object detection consists of a Backbone and a Head. The Backbone extracts features from the input image through multiple stages (P1 to P5) using convolutional layers (Conv), Cross Stage Partial layers (C2f), and Spatial Pyramid Pooling (SPPF). The Head refines these features and generates detections using additional Conv layers, concatenation layers (Concat), upsampling layers, and final detection layers. Key components like Bottlenecks and different model configurations (varying depth and width) are included to enhance learning and performance. The final output involves predicting object bounding boxes, classes, and confidence scores, with loss calculated using bounding box regression, classification loss.

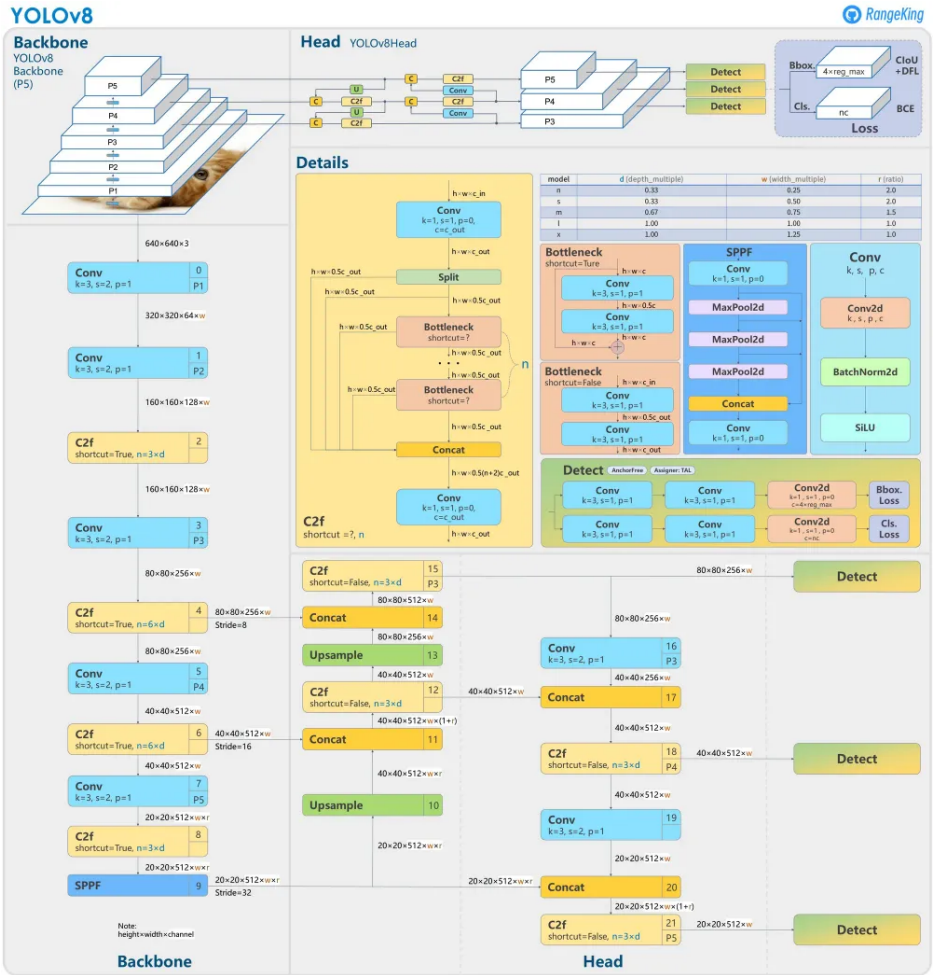


FIGURE 2.8 – YOLOv8 Architecture [6]

2.2 Literature Review

In the literature review, among seven several scientific articles were reviewed within the same general context of our project. These articles highlighted the importance of employing advanced techniques in image processing, machine learning, and predictive modeling to address traffic management challenges effectively. By studying these articles, we gained a deeper understanding of existing solutions and were able to select those that best suited our needs. However, it is not easy to implement all the functionalities at once Among the various approaches explored, two key ideas emerged as potential avenues for improving accuracy in our project.

The first idea stemmed from the recent advancements in object detection algorithms, particularly the latest version(v8) of You Only Look Once (YOLO). YOLO has demonstrated remarkable performance in real-time object detection tasks, making it a promising candidate for accurately identifying emergency vehicles in traffic scenes. By leveraging the capabilities of YOLO, we aim to enhance the precision and efficiency of our vehicle detection system, thereby improving overall traffic control.

The second idea revolves around the utilization of Long Short-Term Memory (LSTM) networks for congestion prediction, combined with a boosting model XGBoost to reduce the error values. LSTM networks excel in capturing temporal dependencies in sequential data, making them well-suited for forecasting future traffic conditions based on historical data. By integrating LSTM with a boosting model, such as XGBoost, we aim to exploit the complementary strengths of both approaches to achieve more accurate predictions of congestion patterns. This hybrid approach has the potential to enhance the robustness and accuracy of our traffic prediction system, ultimately leading to more effective traffic management strategies.

In addition to exploring advanced machine learning techniques, another promising avenue identified in the literature involves the integration of simulation environments with physical hardware, such as ArduinoUNO , Arduino boards, camera, sound detectors and LEDs ...

2.3 Related Work

During our analysis of various scientific articles, we selected a few for their pertinence, especially those concerning object detection techniques using Image Processing, which reviews several approaches for intelligent traffic control system with prioritization during the detection and approving the emergency situation.

In the article titled "A Transfer-Learning-Based Approach for Emergency Vehicle Detection" [5], author Abubakar M. Ashir, Department of Computer Engineering, Faculty of Engineering, Tishk International University, Erbil,Iraq conducted a study on computer-vision based approach for real-time detection of different types of emergency vehicles under heavy traffic conditions it proposed a method of automatic detection of four different categories of emergency vehicle irrespective of the vehicle's shapes, models ormanufacturers using modified version of YOLOv5 object detection algorithm. The proposed model developed here is based on 4 classes which are (Firetrucks, Ambulance, Police Car, and Normal Cars) classes. The top layers (fully connected layers) of the YOLO algorithm were re-designed and retrained to get new learned weights while freezing the bottom layers (convolutional layers) and retaining the pre-trained YOLOv5 weights.

The second interested article titled "LSTM network : a deep learning approach forshort-term traffic" [10] authors Zheng Zhao1, Weihai Chen1 , Xingming Wu1, Peter C. Y. Chen2, Jingmeng Liu11School of Automation Science and Electrical Engineering, Beihang University, Beijing, People's Republic of China2Department of Mechanical Engineering, National University of Singapore .Experiments are conducted to validate the efficiency of the proposed forecast model. According to the comparison between 5 models , the performance of the proposed LSTM network is better than SAE, RBF, SVM and ARIMA model,especially when the forecast time is long.The study focuses on traffic volume prediction, but a comprehensive traffic forecast which includes travel time, traffic speed and occupancy has more significance for commuters.

After reviewing these articles, we were inspired to enhance the existing work to achieve higher accuracy and improved performance using the latest versions.Inspired by those articles we decided to implement and refine the YOLOv8 algorithm for more efficient and accurate detection of emergency vehicles and to improve traffic volume prediction by combining LSTM with advanced boosting techniques like XGBoost. By integrating these modern methodologies and leveraging the latest versions of these models, we aspire to develop a more robust and precise intelligent traffic control system.

The table below presents a comparative summary of another five relevant articles we have studied too. These

articles cover various aspects ranging from intelligent transportation systems for sustainable smart cities to the use of machine learning and deep learning for short-term traffic prediction. This analysis has allowed us to identify current trends, challenges, and opportunities in this rapidly evolving field of research.

Title	**Research problem**	**Objective**	**Methodology**	**Proposed solution**	**Findings and conclusion**
Intelligent transportation systems for sustainable smart cities	How do intelligent transportation systems contribute to the sustainability of smart cities ?	Analyze how ITS can enhance traffic management and reduce greenhouse gas emissions	Literature review, case studies for 8 cities and 7 cars	Integration of ITS technologies like VANET, ITL, VTL to improve traffic flow, enhance safety and promote sustainability	VANET enable real-time traffic data sharing. ITLs dynamically adjust traffic light timings. Mobility prediction anticipates traffic patterns.
Blockchain and AI technology convergence : Applications in transportation systems	How can the challenges in implementing blockchain and AI technologies in transportation systems be effectively addressed ?	Investigate advantages and obstacles of merging blockchain and AI within transportation systems	Review of existing studies, classification based on technology applications, use of "Block5GIntell" framework	Adoption of "Block5GIntell" framework to facilitate convergence of blockchain and AI technologies	Potential energy savings of around 20% achievable, emphasizing efficiency and security enhancements
A novel framework to avoid traffic congestion and air pollution for sustainable development of smart cities	How can we identify congestion and estimate air quality to promote sustainable development in smart cities ?	Propose a single framework to determine shortest path considering traffic congestion and air quality	Use crowdsourcing, k-means, GPS, A* data, and map matching algorithm	Develop a model based on maximum likelihood, crowdsourcing and route optimization	98% accuracy for location identification, 87% accuracy for finding fastest route. NSRD improved accuracy by 3.8%
Short-term traffic flow prediction : An ensemble machine learning approach	How can short-term traffic flow prediction be improved to optimize urban traffic management ?	Propose a machine learning-based approach for accurately predicting short-term traffic flow	Utilization of improved BAT algorithm to optimize VMD, use of LSTM to enhance information scope	Ensemble-based machine learning approach combining VMD and LSTM with improved Bat Algorithm	OVMD-L-BILSTM model can effectively predict short-term traffic flow
Traffic management approaches using machine Learning and deep learning techniques : A survey	How can ML and DL techniques be effectively utilized to address challenges in traffic management ?	Provide analysis of traffic management strategies using ML and DL, evaluate generic architecture, identify research issues	Systematic search technique using databases, keyword selection, evaluation of 20 recent research prototypes	Utilizing ML and DL techniques to enhance traffic management systems, developing generic architecture with 12 assessment criteria	ML and DL models show promise in accurate traffic predictions. Integration of IoT and deep learning models has potential to revolutionize traffic manage

Chapter 3

Emergency Vehicle Detection and Predictive Modeling of Traffic Congestion

Our work focuses on two main areas : emergency vehicle detection and traffic congestion prediction. We initiated our work by prioritizing the different types of emergency vehicles (ambulance, police car, fire truck) detection, recognizing its critical significance in saving lives and ensure security in our society. This chapter constitutes a crucial step in the development of this detection system with YOLOv8. It aims to provide a comprehensive understanding of the overall system architecture, the selection of technologies and tools used, the development methodology adopted, and the detailed design of the various modules that make up the system.

The second part of our work focuses on predicting traffic congestion. Accurately forecasting traffic conditions is essential for improving urban mobility and reducing commute times. In this chapter, we delve into the predictive modeling techniques employed, particularly emphasizing the utilization of LSTM and XGBoost models. We explore the dataset preparation, feature engineering, model training, and evaluation processes.

3.1 Emergency Vehicle Detection

3.1.1 Description of the Overall Architecture of the Emergency Vehicles Detection System

In our approach, we have a pre-built model that has been retrained to detect emergency vehicles from images and videos. Our system follows steps like data acquisition, image and video capture, image preprocessing, emergency vehicles detection, computer vision and machine learning techniques.

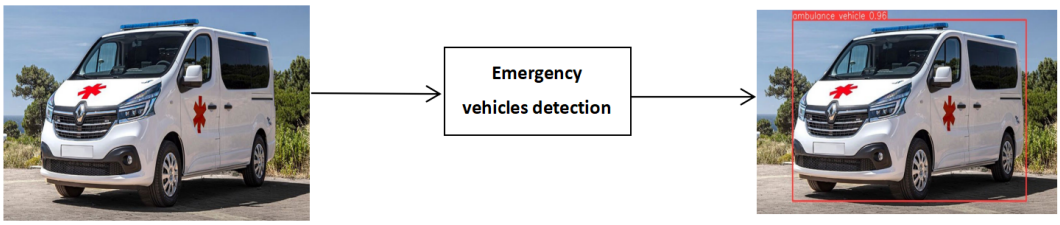


FIGURE 3.1 – Example of detection of emergency vehicle (Ambulance)

3.1.2 Methodology

In the upcoming sections, we will introduce the CRISP-DM methodology, which is a widely adopted framework for data mining and machine learning projects. This methodology provides a structured and systematic approach that guides the entire data mining process, from the initial stages to the final stages. The CRISP-DM method consists of six major phases.

•Business Understanding : The first phase Consists of understanding the project's element and problems, identifying the goals, determining the requirements and the scope of the project and also defining key performance indicators.

•Data Understanding : The second phase aims to precisely determine the data to be collected, explored and analyzed in order to gain insights into its quality, structure, and relationships.

•Data preparation : The third phase involves the selection, cleaning, and transformation of the data to make it suitable for analysis.

•Modeling : The fourth phase data mining and machine learning techniques are applied and combined to develop models that respect the business perspectives and respond to their needs it includes using accurate algorithms, training the models and make parameters' adjustments to get higher performance.

•Evaluation : The fifth phase consists of evaluating the developed model in many criteria depending on the business objectives that needs to be carried out.

•Deployment : The last phase the selected model is implemented in the operational environment, where its performance is monitored, and continuous support and maintenance are provided.

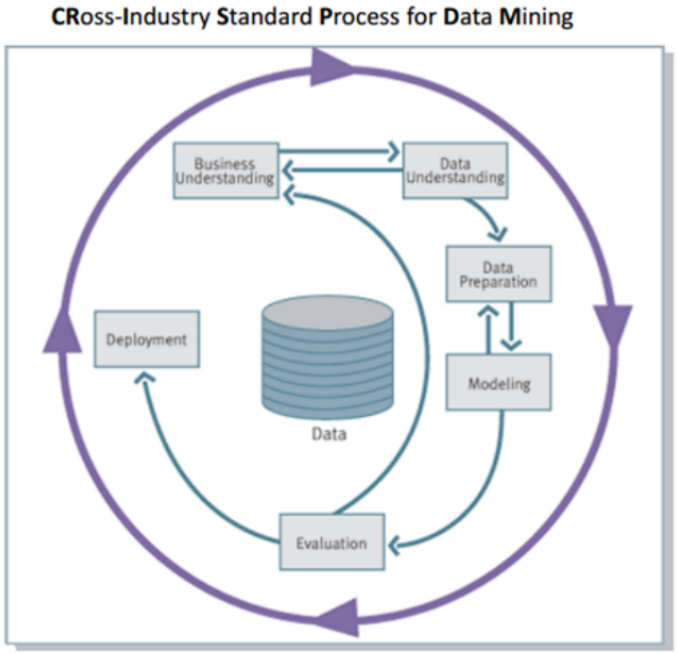


FIGURE 3.2 – CRISP-DM methodology

3.1.3 Dataset

Selecting the appropriate dataset has proven to be a challenging task for us. Initially, we made the mistake of choosing an unlabeled dataset with pictures of European emergency vehicles, but with the guidance of our professor and after conducting thorough research, we were able to create a new dataset featuring with pictures of Moroccan emergency vehicles, specifically those from Marrakech.

Some pictures were originally collected by individuals who had worked on a similar project at the National

School of Applied Sciences (ENSA) in previous years and were subsequently uploaded to Roboflow [11]. However, this dataset was not sufficiently rich for our needs, especially concerning a dataset focused on Moroccan emergency vehicles. Recognizing this limitation, we identified the necessity for a more specific dataset containing images of Moroccan emergency vehicles, particularly those from Marrakech. To address this gap, we augmented our dataset by gathering additional data through various methods. This included capturing screenshots from YouTube videos featuring Moroccan vloggers who attached cameras to their helmets, as well as individuals who filmed emergency situations involving local emergency vehicles.

To accomplish our objective, we split the emergency class into 3 classes (Ambulance, Fire truck and Police car) as seen in following figure.

FIGURE 3.3 – Re-grouped emergency vehicle dataset

To enhance the diversity of the dataset, we included various backgrounds, different types of emergency situations, and incorporated multiple objects in the photographs. Additionally, the images were collected under different lighting conditions, various weather conditions, different light geometries, and they may contain imperfections. This approach aimed to capture a wide range of scenarios that the model could encounter when classifying images of emergency vehicles.

FIGURE 3.4 – Images with different lighting conditions

3.1.4 Data Preprocessing

To effectively train a supervised learning algorithm like YOLO for object detection, we compiled positive sample description files that included spatial coordinates (such as object centers, width, and height) and object classes within the image. For YOLO, these bounding boxes are normalized relative to the entire image. We used a dedicated graphical tool to generate these bounding boxes and extract the necessary data, a process known as object annotation, as depicted in the following figure.

Due to the slow importation of data to Google Drive, we opted to use a specific dataset from Roboflow for training YOLOv8 in Google Colab. This dataset was chosen because it didn't require complete importing and allowed us to explore the capabilities of Google Colab.

We annotated the images to indicate the exact location of the vehicles appearing in each image using the Roboflow tool.

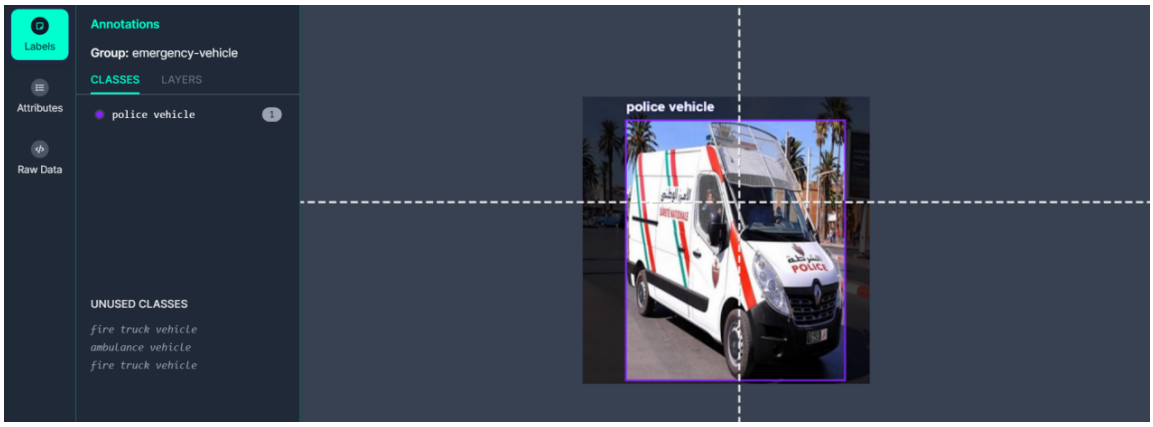


FIGURE 3.5 – Image Annotation with Roboflow

Before proceeding to the training phase, it is essential to verify the pre-processed data to ensure it is correct and ready for model training. Once the image processing is complete, we manage and download a copy of this data. The "data.yaml" file in the data folder contains the model configuration, including a list of classes (our file contains 3 classes), training and validation paths, etc.

3.1.5 Process of Training the Model

We utilized Google Colab's robust computational resources to train our YOLOv8 model. Initially, we installed the Ultralytics library, which offered us a pre-trained YOLOv8m model. Next, we created a YAML file. This configuration file ensured accurate interpretation of the dataset and correct association of labels with detected objects during training.

Before transferring the dataset to Google Colab, we trained it on Roboflow. Using Roboflow's platform, we labeled and annotated our images, then preprocessed and augmented the dataset to improve model performance. After preparing the dataset, we generated an API key from Roboflow to facilitate the data transfer.

To download our dataset, we used the API key from Roboflow to transfer the data to Google Colab. By leveraging Google Colab's capabilities, including access to an NVIDIA Tesla T4 GPU, we conducted the training process over 150 epochs.

To facilitate quick training, we opted for Google Colab as our working environment because it provides powerful processors, collaborative workspace, and better memory capabilities compared to our own machines.

•We installed the Python Ultralytics object detection library using the following command :

!pip install ultralytics

•Afterwards, we at Roboflow have trained and prepared the dataset (labeling and boosting) using the following command :

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="arXTRTzhLHcz0JDceVzO")
project = rf.workspace("abdelouafi-boumoula").project("emergency-vehicle-wuhke")
version = project.version(1)
dataset = version.download("yolov9")
```

FIGURE 3.6 – Command For Preparing The Dataset

•We successfully trained a YOLOv8 model on a custom dataset, enabling it to accurately detect ambulances, fire trucks, and police vehicles. The training process lasted 0.331 hours and yielded outstanding results, with a mean Average Precision (mAP) score of 0.963 across all classes. The model was trained 150 epochs.

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    150/150      7.08G      0.263     0.2249     0.9251          9        640: 100% 10/10 [00:05<00:00,  1.81it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:00<00:00,  3.43it/s]
                   all         38         42      0.844      0.959      0.958      0.869

150 epochs completed in 0.331 hours.
Optimizer stripped from runs/detect/train3/weights/last.pt, 52.1MB
Optimizer stripped from runs/detect/train3/weights/best.pt, 52.0MB

Validating runs/detect/train3/weights/best.pt...
Ultralytics YOLOv8.2.28 🚀 Python-3.10.12 torch-2.3.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 218 layers, 25841497 parameters, 0 gradients, 78.7 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:00<00:00,  2.78it/s]
                   all         38         42      0.927      0.891      0.963      0.902
      ambulance vehicle         20         20      0.939       0.95      0.986      0.941
     fire truck vehicle          4          4      0.969          1      0.995      0.914
         police vehicle         14         18      0.872      0.722      0.908      0.852
Speed: 0.3ms preprocess, 10.8ms inference, 0.0ms loss, 1.2ms postprocess per image
Results saved to runs/detect/train3
💡 Learn more at https://docs.ultralytics.com/modes/train
```

FIGURE 3.7 – Example of epochs

•We used the display function from the IPython.display module to display an image. The filename argument specifies the location of the image and the width argument sets the width of the image in pixels.

```
from IPython.display import display, Image

display(Image(filename="/content/runs/detect/train3/results.png", width=600))
```

FIGURE 3.8 – loss and accurracy

3.1.6 Model Evaluation

3.1.6.1 Confusion Matrix

The confusion matrix is a commonly used technique to evaluate the performance of a detection and classification model in the field of machine learning. It allows visualizing and analyzing the results of a model's predictions compared to the actual values of the data. The confusion matrix is typically represented as a table, with the model's predictions along one axis (usually columns) and the actual values along the other axis (usually rows). The table is divided into four cells, each representing a combination of correct and incorrect predictions.

Here is an explanation of the elements in the normalized confusion matrix :

1.True Labels (X-axis) : These are the actual classes.

2.Predicted Labels (Y-axis) : These are the predicted classes by the model.

3.Values in the matrix : These represent the normalized frequency of predictions. The values range from 0 to 1, indicating the proportion of predictions.

3.2 Traffic Congestion Prediction Using LSTM and XGBoost Models

In recent years, traffic congestion prediction has become a prominent research area, particularly within artificial intelligence and machine learning. Long Short-Term Memory (LSTM) networks, a type of recurrent neural network, excel in this task due to their ability to capture long-term dependencies in data. By analyzing complex temporal patterns, LSTM models accurately forecast traffic congestion, aiding in the development of intelligent transportation systems (ITS) for efficient traffic management and congestion reduction.

3.2.1 Introduction

With the steadily increasing number of vehicles, road traffic congestion has become an increasingly important issue . Predicting traffic congestion is essential for improving urban traffic flow. Traffic congestion prediction, especially short-term traffic congestion prediction is made by evaluating different traffic parameters. Most of the researches focus on historical data in forecasting traffic congestion and combining techniques with deep learning approaches. This combination results in highly accurate traffic flow predictions.

3.2.2 Proposed work

In this work, we focused on improving traffic congestion prediction using advanced machine learning models. Initially, we tested a basic Long Short-Term Memory (LSTM) model, which resulted in high Root Mean Square Error (RMSE) values, indicating significant prediction errors. To enhance the model's performance, we refined the LSTM model, leading to a reduction in RMSE and improved accuracy. Further, we explored the combination of the improved LSTM model with XGBoost, a powerful gradient boosting algorithm. This hybrid approach yielded even lower error rates, demonstrating the effectiveness of integrating different machine learning techniques for traffic prediction.

Throughout our analysis, we meticulously assessed model performance using various metrics, including RMSE, to quantify prediction accuracy. Visual representations in the form of graphs were instrumental in presenting comparative analyses between real data and model predictions. These visualizations offered valuable insights into the efficacy of our methodologies, allowing us to discern trends, patterns, and areas for further refinement.

This chapter provides a comprehensive understanding of our approach and its implications for traffic prediction research and details our steps to obtain our improved model (the combination of LSTM and XGBOOST).

3.2.3 Software Tools

3.2.3.1 Programation language

Python :
is an open-source programming language, one of the most interesting programming languages of the moment due to its power and ease of learning. It has high-level data structures and allows for a simple yet effective approach to object-oriented programming. It is also an interpreted language (meaning it does not need to be compiled before execution). Python is ideal for writing simple scripts and rapidly developing applications in various domains and on most platforms, thanks to its extensive libraries.

FIGURE 3.9 – Logo of Python

3.2.3.2 Development environment

Google Colab :
is a cloud service provided by Google (free of charge), based on Jupyter Notebook and aimed at education and research in machine learning. This platform allows for training machine learning models directly in the cloud, without the need to install anything on your computer except for a web browser.

FIGURE 3.10 – Logo of Google Colab

3.2.4 Methodology

The training methodology for the base LSTM model involves several crucial steps aimed at preparing the data and optimizing the model's performance. Firstly, data preprocessing is conducted, encompassing the encoding of categorical variables to render them suitable for machine learning models and normalization of the data to ensure uniform scaling across features, facilitating model learning. Subsequently, outliers are detected using statistical techniques to identify potentially disruptive data points, which are then removed to enhance model robustness.

Following data preparation, the dataset is partitioned into training and testing sets using the train_test_split function, allowing for the evaluation of the model's performance on unseen data. The LSTM model architecture is then constructed, comprising one or more LSTM layers followed by a Dense output layer. The model's optimizer and loss function are configured, setting the stage for model training.

Training commences by fitting the model to the training data using the fit function, where parameters such as the number of epochs and batch size are specified to control the training duration and optimization process. Once training is complete, the model's performance is evaluated using metrics such as the Root Mean Squared Error (RMSE) to assess the accuracy of its predictions.

Throughout the training process, rigorous evaluation and comparison of predicted values against ground truth data are conducted through visualizations and quantitative metrics. This comprehensive methodology ensures a systematic approach to developing and assessing the base LSTM model's efficacy in predicting traffic congestion.

3.2.5 Data used

The proposed work and models was applied to the data collected from Kaggle [12]. The dataset captures vehicle counts and traffic conditions at 15-minute intervals and captured with cameras,and velocity radars to obtain the traffic data such as Time, Date, Day of the Week, CarCount, BikeCount, BusCount, TruckCount,number of vehicles counted and Traffic Situation categorized as 'low', 'High', 'normal', or 'heavy'.

Time	Date	Day of the...	CarCount	BikeCount	BusCount	TruckCount	Total	Traffic
12:00:00 AM	10	Tuesday	31	0	4	4	39	low
12:15:00 AM	10	Tuesday	49	0	3	3	55	low
12:30:00 AM	10	Tuesday	46	0	3	6	55	low
12:45:00 AM	10	Tuesday	51	0	2	5	58	low
1:00:00 AM	10	Tuesday	57	6	15	16	94	normal
1:15:00 AM	10	Tuesday	44	0	5	4	53	low

FIGURE 3.11 – Traffic Prediction Dataset

3.2.5.1 Data Segmentation

This is a crucial step that allows for effective evaluation of our model by splitting the input data into training, validation, and test subsets. This segmentation helps prevent overfitting and ensures the model's performance is assessed accurately.

Train-Validation-Test Split is a technique used to evaluate the performance of machine learning models, whether for classification or regression tasks. The given dataset is divided into three subsets :

— Training Dataset : This subset of data is used for training the model, i.e., for adjusting the parameters of the machine learning model.

— Validation Dataset : This subset of data is used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. It also plays a role in other model preparation stages, such as feature selection.

— Test Dataset : This subset of data is used to provide an unbiased evaluation of the final model fit on the training dataset and to measure the model's performance.

In our study, we divided the dataset such that most of the data (80% is used for training, a smaller portion (20%) is used for testing, and 10% of the training data is used for validation to evaluate the model.

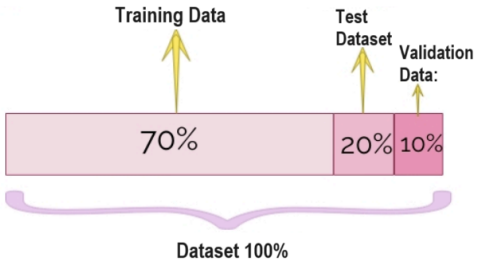


FIGURE 3.12 – Data Segmentation

3.2.5.2 Dataset Normalization

We normalize the data to ensure that all features are on a comparable scale. This is crucial for machine learning models, including neural networks like LSTM, as it helps in faster and more stable convergence during training. Normalization prevents features with larger scales from dominating the learning process and ensures that each feature contributes equally to the model's predictions. Overall, normalization enhances the model's performance and aids in achieving better generalization on unseen data.

3.2.5.3 Data Cleaning : Detecting and Removing Outliers

Identify outliers in the data using the interquartile range (IQR) method. Outliers can skew the model training process, so identifying them helps in cleaning the data. Once identified, outliers are removed from the dataset to improve the quality and reliability of the data, leading to better model performance.
After this detection the outliers of our dataset was :
Outliers for CarCount : 0
Outliers for BikeCount : 77
Outliers for BusCount : 0
Outliers for TruckCount : 0

3.2.5.4 Encoding Categorical Variables

Categorical variables(Day week and traffic situation on our dataset) are encoded into numerical values using OrdinalEncoder. This is necessary because machine learning models generally require numerical input, and the specified order ensures that the ordinal relationships (e.g., traffic levels) are preserved.

Original encoding	Ordinal encoding
Low	0
Normal	1
High	2
Heavy	3

FIGURE 3.13 – Eg. Traffic situation data encoding

3.2.6 Model Conception

The model conception begins with the careful preparation of the data, ensuring it is properly formatted and cleansed. First, categorical variables such as 'Traffic Situation' and 'Day of the week' are encoded using a specified order to maintain the ordinal relationship between categories. Time features are converted into a numerical format by transforming the time into minutes and normalizing it. Additionally, date features are converted into timestamps to facilitate processing.

Before normalizing the dataset, we detect and remove outliers using the Interquartile Range (IQR) method to ensure that extreme values do not adversely affect the training process. With the data cleansed, we normalize the features using MinMaxScaler and reshape them to fit the LSTM model's input requirements.

This robust preprocessing pipeline ensures that our improved LSTM model, equipped with dropout layers to prevent overfitting, can learn and make accurate predictions on the traffic situation. The model is then trained and evaluated, demonstrating improved performance metrics.

3.2.7 Initial Basic LSTM Model

LSTM neural networks have a chain-like structure similar to traditional RNNs, but their internal operations within the LSTM cells are more complex. This complexity allows LSTMs to learn long-term dependencies effectively, addressing issues like exploding and vanishing gradients that often occur when learning long-term dependencies, even when the time lags are very long.

This initial model is a basic Long Short-Term Memory (LSTM) neural network used for predicting traffic situations. It preprocesses data, trains on historical traffic features, and evaluates its performance using various metrics like root mean squared error (RMSE) and mean absolute error (MAE).

The results from this basic model serve as a benchmark, providing error values that are used as thresholds for comparison with subsequent models. By comparing these initial error values with those from more advanced models, we can measure the improvements and effectiveness of the enhancements made. The model's predictions are visualized and saved for further analysis.

3.2.7.1 Modelization and Training

The training process for this model involves several key steps to optimize its performance in making traffic congestion predictions. Firstly, the model architecture is defined using the Keras Sequential API. The architecture consists of a single LSTM layer with 50 units as the primary layer, followed by a Dense layer with a single unit for output. The LSTM layer is pivotal for capturing temporal dependencies inherent in sequential traffic data, while the Dense layer provides the final output for prediction. Throughout training, the dataset is iterated over multiple epochs, with each epoch representing a complete pass through the entire dataset. In this case, the model is trained for 20 epochs, allowing it to learn from the data in multiple iterations. Additionally, the dataset is divided into batches, and the model parameters are updated after processing each batch. A batch size of 32 is chosen for efficient parameter optimization. The Adam optimizer, known for its effectiveness in training neural networks, is employed to minimize the mean squared error loss function, ensuring that the model's predictions align closely with the actual traffic congestion levels. Following training, the model's performance is evaluated using metrics such as mean absolute error (MAE), mean squared error (MSE), and root mean squared error (RMSE) to gauge its accuracy and effectiveness in predicting traffic congestion.

Here are the **hyperparameters** for each basic model :

hyperparameters	Values
LSTM Units	50
Input Shape	1
Epochs	20
Batch Size	32

FIGURE 3.14 – Hyperparameters of our basic LSTM model

After initializing the parameters and optimizing the model's hyperparameters,in this stage, our model learns from the training data to obtain a better-trained model, thus improving its predictive capability and yielding better results. To assess the model's capacity, it is then subjected to testing using test data. The Backpropagation Through Time (BPTT) algorithm is utilized to train the model parameters.

3.2.8 Improved LSTM Model

This second model represents a significant improvement over the initial basic model. It utilizes a more complex LSTM architecture with multiple layers and incorporates additional features for better traffic situation prediction. Compared to the basic model, it offers increased accuracy and reliability due to its enhanced architectural design, expanded feature representation, and more extensive training.

3.2.8.1 Modelization and Training

This second model represents a notable advancement compared to the initial basic model in several crucial aspects. Firstly, the architecture has been substantially upgraded. While the basic model utilized a single LSTM layer with 50 units, the improved model incorporates multiple LSTM layers, each with 100 units, resulting in a deeper and more complex neural network architecture. Additionally, dropout layers have been introduced after each LSTM layer to mitigate overfitting, a feature absent in the basic model. These architectural enhancements aim to capture more intricate temporal patterns in the traffic data, leading to more accurate predictions.

Moreover, the feature set used in training has been expanded significantly. While the basic model relied solely on time-related features such as Time of the day, the enhanced model incorporates additional features such as Date alongside existing traffic metrics. By incorporating date information in a timestamp format, the model gains a broader understanding of temporal dynamics, enabling it to discern more nuanced patterns in traffic behavior over time. This richer feature representation empowers the model to make more informed predictions, especially in scenarios where temporal variations play a crucial role.

In terms of training methodology, the improved model underwent a more extensive training regimen compared to its predecessor. While the basic model was trained for only 20 epochs with a batch size of 32, the enhanced model underwent 100 epochs with a larger batch size of 64. This extended training duration allowed the model to learn more effectively from the augmented feature space, resulting in better convergence and parameter optimization.

Here are the **hyperparameters** for our improved model :

hyperparameters	Values
LSTM Units	100
Input Shape	3
Epochs	100
Batch Size	64

FIGURE 3.15 – Hyperparameters of our improved LSTM model

After initializing the parameters and optimizing the model's hyperparameters,in this stage, our model learns from the training data to obtain a better-trained model, this second model demonstrates a substantial advancement in training methodology compared to the initial basic model. The extended training duration allowed the model to learn more effectively from the augmented feature set and the richer temporal dynamics captured by the additional date-related features. As a result, the enhanced model achieved better convergence and parameter optimization, leading to improved predictive accuracy and robustness in traffic situation prediction tasks.

3.2.9 Combined LSTM and XGBOOST Model

In recent years, ensemble learning–based methods have been more and more widely used for traffic data analysis. The purpose of an ensemble learning algorithm is to achieve an improved result by combining predictions of a group of individual base models. It has been shown that the combined model often generates more stable and accurate predictions in many applications [13].

3.2.9.1 Definition

Boosting is another ensemble learning method which improves the prediction accuracy through developing multiple models in sequence by putting emphasis on the samples in the model that are difficult to estimate.

This combined model leverages the predictive power of both LSTM and XGBoost algorithms to enhance the accuracy and robustness of traffic situation forecasting. By integrating the temporal understanding of LSTM with the ensemble learning capabilities of XGBoost, it seeks to provide more accurate predictions, this aiding in traffic management and planning.

3.2.9.2 XGBoost model

The XGBoost algorithm is an additive model consisting of k an additive model consisting of individual decision tree models, and the prediction accuracy improves as the number of model iterations increases [14]. Assuming that the first k iteration of the tree model to be trained is fk(x) The final prediction for the first i the final prediction for the first sample is :

$$\hat{y}_i^{(k)} = \sum_{k=1}^{k} f_k(x_i) = \hat{y}_i^{(k-1)} + f_k(x_i) \tag{1}$$

Eq:$\hat{y}_i^{(k)}$ --Previous$k$ round model prediction value;$\hat{y}_i^{(k-1)}$--- previous$k-1$ round of model predictions;$f_k(x_i)$--th$k$ decision tree model.

To optimize the objective function, Taylor series, regularized expansion is used to combine the primary and secondary function coefficients and k iterations are performed to obtain the formula for the final prediction :

$$\text{Obj}_k = \Omega(f_k) + \sum_{k=1}^{n} \left[l\left(y_i, \hat{y}_i^{(k-1)}\right) + g_i \cdot f_k(x_i) + \frac{1}{2} h_i \cdot f_k^2(x_i) \right] \tag{2}$$

Eq: $\hat{y}_i^{(k-1)}$ --Previous $k-1$ round model prediction value; $l\left(y_i, \hat{y}_i^{(k-1)}\right)$ --sample x_i of the training error;g_i, h_i--pre-training$k-1$ Residuals at tree;$\Omega(f_k)$--the k regular term of the tree.where the $\Omega(f_k)$ The canonical term indicates the complexity of the structure of this model, the smaller the result, the more accurate the prediction.

The XGBoost algorithm is an advanced version of the gradient boosting decision tree, which grows the decision tree by constant feature splitting, and in the process of learning the decision tree, it fits the error between the actual value and the predicted value of the model to improve the accuracy of prediction as shown :

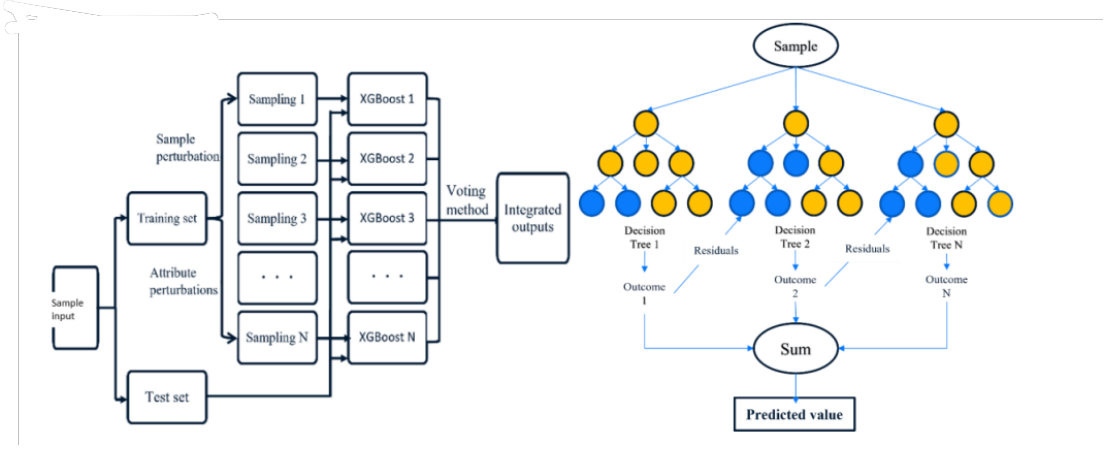


FIGURE 3.16 – XGBoost Model and Algorithm Flowchart

3.2.9.3 Modelization and Training

In contrast to the first basic LSTM model, which relies solely on LSTM layers for prediction, and the second improved LSTM model, which enhances the LSTM architecture by incorporating deeper layers and dropout regularization, the combined model takes a novel approach by integrating and combining XGBoost's results with LSTM's results. While LSTM is adept at capturing temporal dependencies and patterns in sequential data, it may still have limitations in modeling complex relationships within the data. On the other hand, XGBoost, as a gradient boosting algorithm, excels in handling structured/tabular data and capturing non-linear relationships between features.

In the context of traffic situation prediction, the basic LSTM model may struggle to capture all the nuances of traffic patterns due to its simpler architecture. The improved LSTM model addresses this by introducing deeper layers and dropout regularization, allowing it to learn more complex temporal patterns. However, both LSTM models may still have limitations in capturing the full complexity of traffic dynamics, especially when faced with noisy or high-dimensional data.

By conmbining the results with XGBoost's results , the combined model leverages the strengths of both LSTM and XGBoost. This combination allows for a more comprehensive modeling approach, potentially resulting in higher accuracy and reliability in traffic situation prediction. Additionally, the ensemble nature of the combined model provides a form of model diversification, reducing the risk of overfitting and enhancing generalization performance.

In comparison to the previous models, training this combined model involves two stages. Both models, are trained on the training data to capture temporal dependencies and make initial predictions.

Finally, the combined model generates predictions by averaging the outputs of both the LSTM and XGBoost models, resulting in a more refined prediction. Overall, this combined approach capitalizes on the complementary strengths of LSTM and XGBoost, offering a more robust and accurate traffic situation prediction model compared to using either algorithm individually.

3.2.10 Testing and Model Evaluation

At this stage, after obtaining our prediction results, we need to evaluate the performance of our system using the test data. Since the model belongs to regression problems, we will apply metrics specific to this type of problem. The metrics used are as follows :

- **Root Mean Square Error (RMSE) :**

RMSE is a measure of absolute error that squares all errors and deviations to prevent positive deviations from canceling out negative ones. This measure also tends to exaggerate significant errors, which can help identify methods with significant errors.

The equation for RMSE is as follows :

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(yp - yr)^2}$$

Where :
yp : the predicted value
yr : the actual base value
n : the number of all predicted values

- **Mean Absolute Error (MAE) :**

MAE is a popular error measure for regression problems, simply defined as the average of the absolute difference between the predicted output and the actual output. Mean squared error is commonly used because it is independent of whether the prediction was too high or too low ; it simply indicates that the prediction was incorrect.

The equation for MAE is as follows :

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|yp - yr|$$

Where :
yp : the predicted value
yr : the actual base value
n : the number of all predicted values

- **Mean Squared Error (MSE) :**

MSE is the average of the squares of the errors. It is calculated similarly to RMSE but without taking the square root.

The equation for MSE is as follows :

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(yp - yr)^2$$

Where :
yp : the predicted value
yr : the actual base value
n : the number of all predicted values

Furthermore, our experimental setup encompassed rigorous testing and validation procedures, ensuring the robustness and generalizability of our models. By meticulously documenting our methodology and detailing the comparative analysis of model performances,

Chapter 4

Results and analysis

In the previous chapter, we outlined the design of the emergency vehicle detection model and the traffic congestion prediction model, elucidating the various stages of development in detail.

In this chapter, we will introduce the working environment, programming language, and tools utilized to execute our projects. Subsequently, we will delve into the obtained results and conduct a comparative analysis.

4.1 Detection of Emergency Vehicles

4.1.1 Model Testing

Once the training phase is completed, we use the model to make predictions on test images. This script is designed to employ a pre-trained model to make predictions or detect objects in new data. Specifically, it loads the trained model, processes the test images, and outputs the detected objects along with their corresponding confidence scores and bounding boxes. This allows us to evaluate the model's performance on unseen data and verify its accuracy and robustness in real-world scenarios. The results can then be visualized and analyzed to further refine and improve the model if necessary.

Here are some tested images :

FIGURE 4.1 – Validation Batch for Emergency Vehicle Detection

And also, We tested the model on a video of a private ambulance navigating the streets of Marrakech. The results were impressive, with the model achieving high accuracy rates of approximately 92% to 95%. This demonstrates the model's effectiveness in real-world scenarios, accurately detecting emergency vehicles even in complex urban environments. The high accuracy rates validate the robustness and reliability of our model in identifying emergency vehicles, contributing to its potential utility in intelligent traffic control systems.

4.1.2 Model Training and Performance Analysis

The model demonstrates effective learning as evidenced by the decreasing training loss, indicating its improving capability in identifying objects within the training dataset. Moreover, the increasing precision and recall observed on the validation set imply enhancements in the model's proficiency at accurately detecting objects in previously unseen images. Additionally, rising mean Average Precision (mAP) scores further signify an overall enhancement in the model's object detection performance. After the training we can visualize the result in the graphs below :

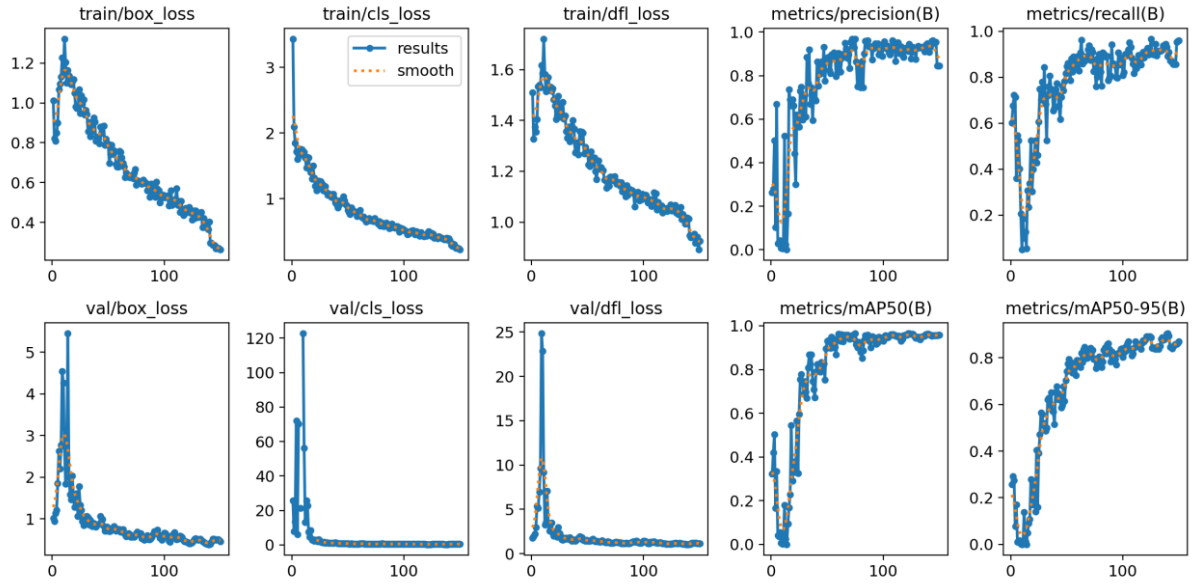


FIGURE 4.2 – Training results performance evaluation

The train/box_loss, train/cls_loss, and train/dfl_loss show a continuous decrease over time, indicating that the model is effectively learning to localize objects, classify them accurately, and refine the bounding box predictions. This demonstrates a robust training process.However, it is worth noting that these loss components don't reach a plateau or stagnation, which suggests that further training could potentially lead to even better performance.On the other hand, the other metrics, such as validation loss exhibit positive signs. The validation loss remains low, indicating that the model is performing well on unseen data.Overall,these graphs indicate that the YOLO model is successfully detecting emergency vehicles with good precision and recall.

4.1.3 Model Evaluation and Results

4.1.3.1 Confusion Matrix

The confusion matrix analysis of the YOLO model for detecting emergency vehicles reveals a nuanced performance across different categories. As seen in following figure the accuracy metrics indicate strong performance, with fire trucks achieving a perfect detection rate of 100%, closely followed by ambulances at 95%, and police vehicles at 83%.

However, the model exhibits notable misclassification patterns : 11% of ambulances are mistakenly identified as police vehicles, and 25% as background objects. Similarly, 5% of police vehicles are misclassified as ambulances, with 6% classified as background.

These findings highlight the model's strengths in accurately detecting fire trucks and maintaining high accuracy for ambulances and police vehicles, yet weaknesses persist in distinguishing between ambulances and police cars, as well as in the tendency to classify them as background objects. Further, false positives are prevalent, notably with 75% of background objects being misclassified as police vehicles. Overall, while the model demonstrates strong capabilities for certain vehicle types, improvements are needed to enhance the differentiation between similar classes and reduce false positives in complex scenarios.

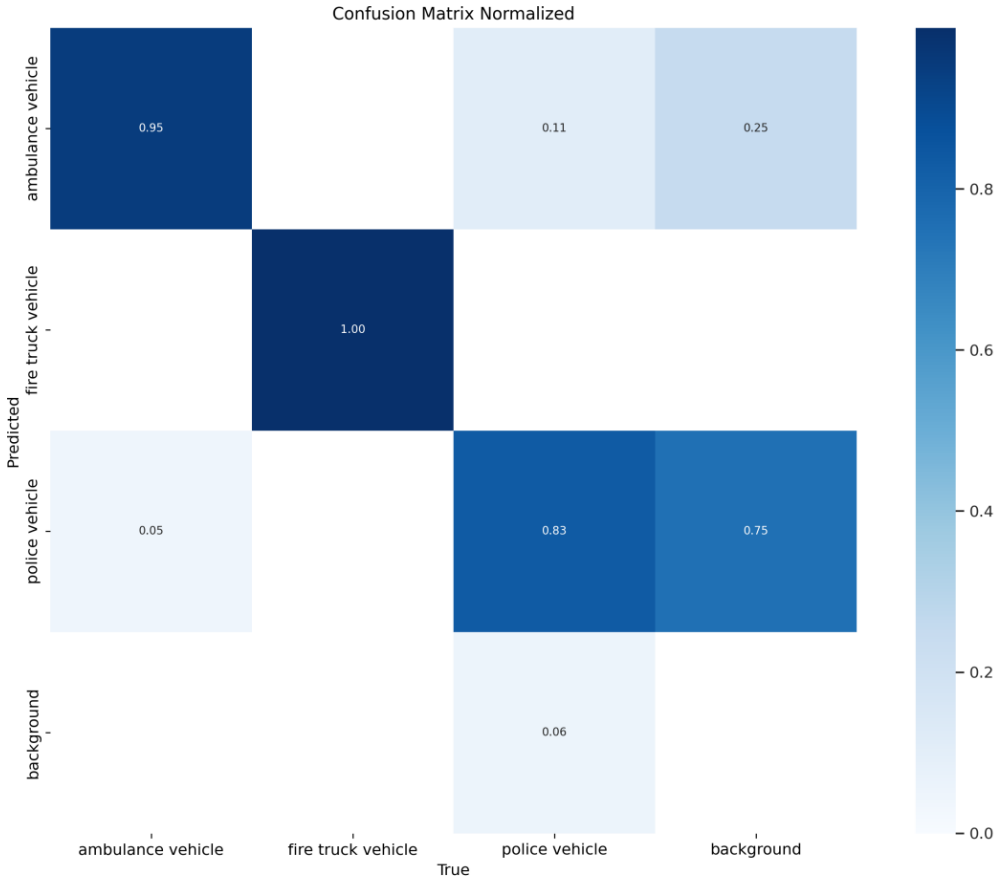


FIGURE 4.3 – Confusion Matrix Normalized

4.1.3.2 Evaluation Metrics : Precision, Recall, and F1-score

To gain deeper insights into the model's performance on individual class levels, we further analyzed and compared the metrics computed during training. Evaluation metrics play a crucial role in assessing the performance of machine learning models, especially in classification tasks. They quantify the accuracy and robustness of a model's predictions compared to actual data. Among the most fundamental metrics are precision, recall, and the F1-score. Each of these metrics provides a different perspective on a model's ability to correctly identify and classify positive and negative instances within a dataset.

Here are the formulas for Precision, Recall, and F1-score [15] :

— **Precision** : Precision is the proportion of correctly predicted positive observations out of all predicted positive observations. It measures the accuracy of positive predictions.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

— **Recall (Sensitivity)** : Recall is the proportion of correctly predicted positive observations out of all actual positive observations. It measures the model's ability to identify positive instances accurately.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

— **F1-score** : The F1-score, derived from the harmonic mean of Precision and Recall, serves to balance these two metrics. It proves especially valuable when a solitary measure is required to assess model performance amidst imbalanced datasets.

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

These measures are commonly employed in classification tasks to assess model performance, particularly in situations where there is an imbalance in class distribution.

- **Precision with Confidence :**

The Precision-Confidence Curve graph illustrates the performance of a model in distinguishing between three classes of emergency vehicles : ambulance, fire truck, and police vehicle. The precision for ambulance vehicles starts at approximately 0.4 at lower confidence levels but improves steadily, reaching nearly 1.0 precision at confidence levels above 0.8. The fire truck vehicle curve shows a rapid improvement in precision from about 0.4 at low confidence levels, achieving near-perfect precision around 0.6 confidence. The police vehicle curve starts at approximately 0.4 as well, with precision improving with confidence but showing fluctuation in the mid-range (0.4 to 0.7 confidence), before reaching high precision near 1.0 at higher confidence levels. The overall performance curve, representing all classes, indicates a consistent increase in precision, achieving perfect precision (1.00) at a confidence level of 0.94. While the model performs well at higher confidence levels, there is room for improvement at lower and mid-confidence levels, particularly for the police vehicle class, to ensure more consistent performance across all classes.

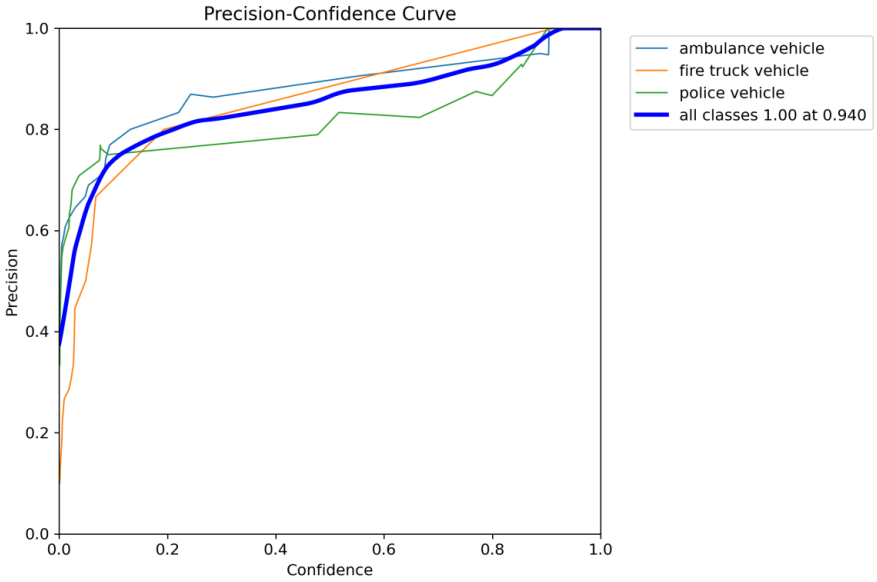


FIGURE 4.4 – Precision-Confidence Curve graph

- **Recall with Confidence :**

The Recall-Confidence Curve graph provides insights into the model's recall performance for three classes of emergency vehicles : ambulance, fire truck, and police vehicle, as well as an overall curve for all classes. For ambulance vehicles, the recall starts at a perfect 1.0 at lower confidence levels and remains high and stable, only slightly declining as confidence approaches 1.0. This indicates the model is highly sensitive to ambulances, capturing nearly all instances correctly until very high confidence thresholds. Fire truck vehicles also start with a perfect recall of 1.0, maintaining high sensitivity across most confidence levels, with a sharp decline near the highest confidence levels. This suggests effective identification of fire trucks, with few missed instances at high confidence thresholds. In contrast, the recall for police vehicles starts slightly below 1.0 and shows more fluctuation, with significant drops around confidence levels of 0.4 to 0.8. This indicates less consistent identification of police vehicles, especially at higher confidence levels. The overall performance curve, representing all classes, starts close to 1.0 recall and remains stable across most confidence levels, with a slight decline and a sharp drop near the highest confidence levels.

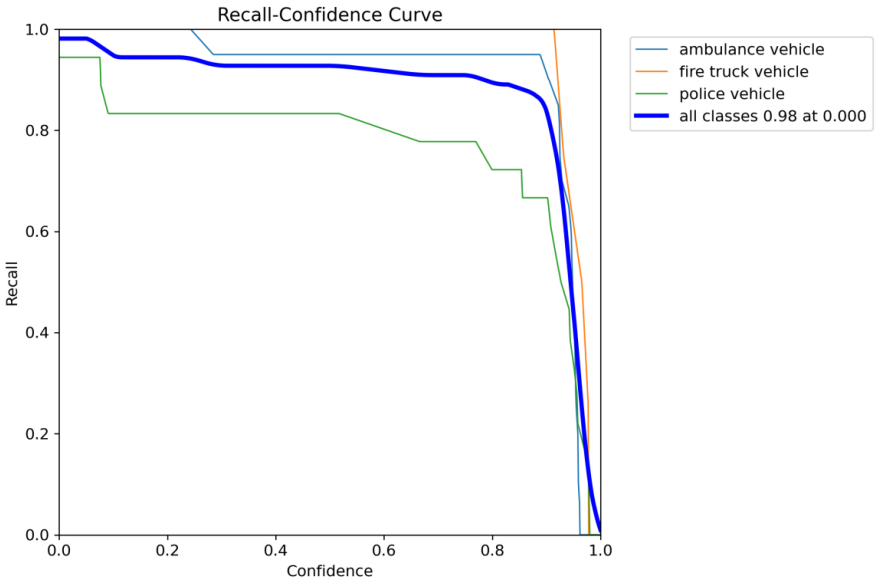


FIGURE 4.5 – Recall-Confidence Curve graph

The high recall values for ambulance and fire truck vehicles suggest strong model performance. The fluctuation and decline in recall for police vehicles highlight the need for improvement in consistent identification. Addressing the recall decline at higher confidence levels and enhancing sensitivity to police vehicles can further improve overall performance. While the model generally achieves high recall for most emergency vehicle classes, refining its performance at higher confidence levels.

- **F1-score with confidence :**

The F1-Confidence Curve graph illustrates the model's F1-score performance across different confidence levels for three classes of emergency vehicles, as well as an overall performance curve for all classes. For ambulance vehicles, the F1-score starts around 0.4 at low confidence levels, quickly rising to approximately 0.9 and remaining stable until around 0.9 confidence, where it then slightly declines. Fire truck vehicles begin with an F1-score near 0.2 at low confidence, rapidly increasing to around 0.95, and maintaining this high value across most confidence levels, with a slight dip near the highest confidence. Police vehicle F1-scores start around 0.4, rising to approximately 0.8, and show some fluctuation between 0.7 and 0.8 across confidence levels, declining more steeply at higher confidence levels. These results highlight the model's strong performance in identifying fire trucks, consistent high performance for ambulances, and more variability and room for improvement in identifying police vehicles.

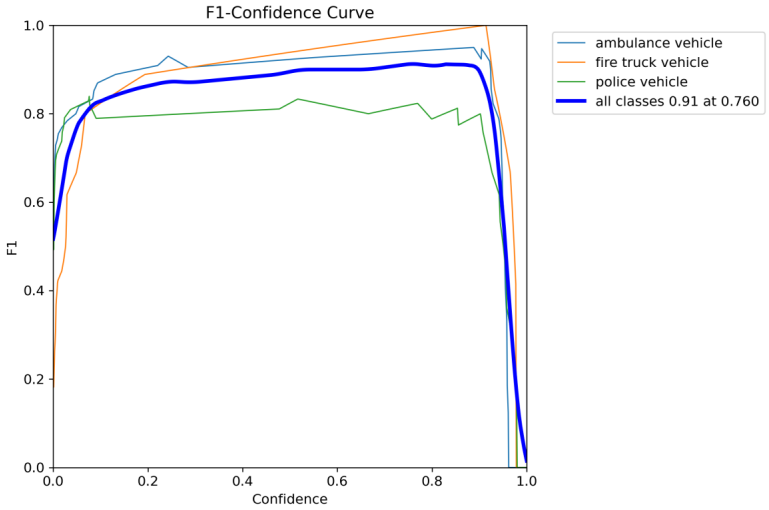


FIGURE 4.6 – F1-Confidence Curve graph

- **Precision with Recall :**

The precision-recall curve depicted in the image provides a comprehensive analysis of the model's performance in identifying different types of emergency vehicles : ambulances, fire trucks, and police vehicles. The curve reveals that the model performs exceptionally well for fire trucks, achieving a high precision and recall, reflected by the area under the curve (AUC) of 0.995. Ambulances also show robust performance with an AUC of 0.986, indicating a good balance between precision and recall. However, the model's performance for police vehicles is notably lower, with an AUC of 0.908, suggesting that distinguishing police vehicles is more challenging for the model. The overall mean average precision (mAP) at a 0.5 intersection over union (IoU) threshold for all classes combined is 0.963, indicating strong overall performance. These results suggest that while the model is highly effective in detecting fire trucks and ambulances, there is room for improvement in accurately identifying police vehicles to enhance the model's overall efficacy.

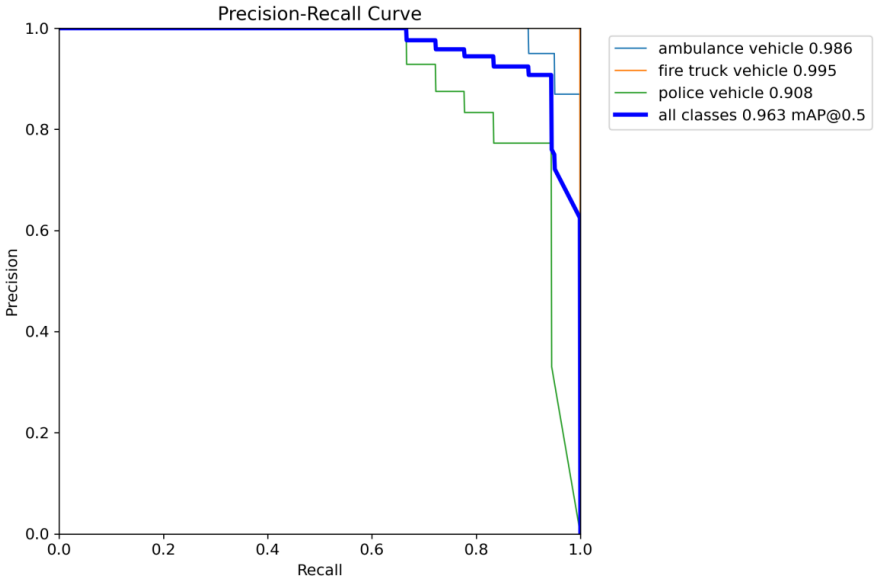


FIGURE 4.7 – Precision-Recall Curve graph

The performance of the trained model was analyzed in detail through computed metrics during training. Precision and recall were plotted against confidence values for individual classes and overall performance. Additionally, F1-score, which represents the harmonic mean between precision and recall, was examined against confidence scores. The comparison between precision and recall across different classes was also studied to understand the model's behavior and effectiveness in distinguishing between emergency vehicle types. These analyses provided insights into how well the model performs across various metrics and highlighted areas where improvements may be necessary for specific classes or overall performance.

- **Conslusion :**

In conclusion, our YOLO model demonstrates robust performance in detecting emergency vehicles, achieving high accuracy rates for fire trucks, ambulances, and police vehicles. While there are minor challenges such as occasional misclassifications and variability in recall for police vehicles, overall, the model proves effective in its task.

4.1.4 Real Experimentation

After evaluating the YOLOv8 model and acknowledging its exceptional performance, which yielded the most favorable results, we proceeded to implement the model using the following Python code :

```python
def update_frame():
    global audio_detected
    ret, frame = cap.read()
    frame = cv2.resize(frame, (640, 480))
    result = model(frame, stream=True)
    visual_detected = False

    for info in result:
        boxes = info.boxes
        for box in boxes:
            confidence = box.conf[0]
            confidence = math.ceil(confidence * 100)
            Class = int(box.cls[0])
            if confidence > 60:
                x1, y1, x2, y2 = box.xyxy[0]
                x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 5)
                cvzone.putTextRect(frame, f'{classnames[Class]} {confidence}%', [x1 + 8, y1 + 100], scale=1.5, thickness=2)
                if Class in [0, 1, 2]:
                    print("Emergency vehicle detected  !")
                    visual_detected = True

    print(f"audio_detected: {audio_detected}, visual_detected: {visual_detected}")

    if visual_detected and audio_detected:
        print("Emergency vehicle with siren detected ! Sending the signal to the Arduino...")
        arduino.write(b'1')
        time.sleep(0.1)
        visual_detected = False
        audio_detected = False
    else:
        arduino.write(b'0')

    cv2image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGBA)
    img = Image.fromarray(cv2image)
    imgtk = ImageTk.PhotoImage(image=img)
    label.imgtk = imgtk
    label.configure(image=imgtk)
    label.after(10, update_frame)
```

FIGURE 4.8 – Code of Detection by Intern Camera

Here is the subsequent result :

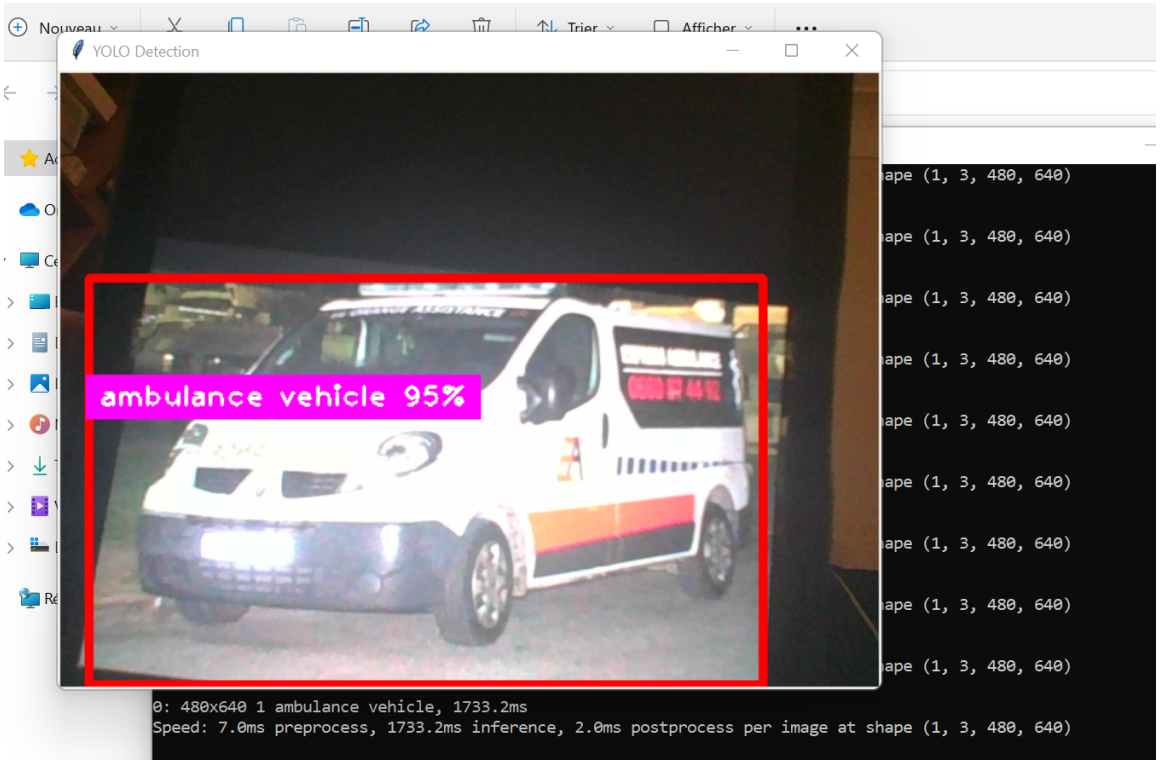


FIGURE 4.9 – Results of Detection using intern Camera

It demonstrates the successful real-time detection of an ambulance vehicle using computer vision technology. The system has identified the ambulance with a high confidence level of 95%, as indicated by the label overlay. This kind of rapid and accurate vehicle detection is crucial for intelligent traffic management systems, especially for prioritizing emergency vehicles.

4.2 Traffic congestion prediction

4.2.1 Software Tools - Libraries

In our programs, we used a set of libraries to leverage predefined functions. They are called at the beginning of the program.

— NumPy (np) : is a powerful Python library used for numerical computations on arrays and matrices. It provides functionalities for efficient data manipulation and mathematical operations.

— Pandas (pd) : is a Python library specialized in data manipulation and analysis. It offers powerful data structures, notably DataFrames, for easy handling of tabular data.

— Plotly press (px) and Plotly Graph Objects (go) : Plotly is an interactive data visualization library in Python. Plotly Express provides a user-friendly interface for quickly creating plots, while Plotly Graph Objects offer more precise control over creating custom graphics.

— Scikit-learn (sklearn) : is an open-source Python library that provides simple and efficient tools for machine learning and data exploration. It offers various machine learning algorithms, data preprocessing tools, and utilities for model evaluation.

— TensorFlow (tensorflow) : is a machine learning library developed by Google. It's widely used for creating, training, and deploying deep learning models. TensorFlow offers flexibility, scalability, and high performance for building complex neural networks.

4.2.2 Basic model LSTM

4.2.2.1 Model definition

In this part, a basic LSTM model is defined using the Keras Sequential API. The model consists of an LSTM layer with 50 units as the primary layer, followed by a Dense layer with a single unit for output. The model is compiled with the Adam optimizer and Mean Squared Error (MSE) loss function.

```
#Model Definition
model_base = Sequential()
model_base.add(LSTM(50, input_shape=(1, X_reshaped.shape[2])))
model_base.add(Dense(1))
```

FIGURE 4.10 – Definition of our basic model LSTM

4.2.2.2 Model training and Prediction

Here, the model is trained using the fit() method. The training data (X_train and y_train) are used for training. The training process is configured to run for 20 epochs with a batch size of 32. Validation data (X_test and y_test) are provided to evaluate the model's performance during training. After training, the model is used to make predictions on the test data (X_test).

```
# Model training
history_base = model_base.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_test, y_test), verbose=2)

# Model Prediction
predictions_base = model_base.predict(X_test)
rmse_base = sqrt(mean_squared_error(y_test, predictions_base))
```

FIGURE 4.11 – Code of our model training and prediction

4.2.2.3 Model Testing

In the context of machine learning, testing a model involves using it to make predictions on a set of data that it has not seen before. This is crucial for evaluating the model's performance and generalization ability. The model makes predictions on the test data (X_test) using the predict method. To visualize the performance of our basic LSTM model, we utilized Plotly to create interactive plots.

First, we created a line plot to compare the real and predicted traffic situations, allowing us to visually assess how well the model's predictions align with the actual values. We used the fig.update_layout method to set the title to "Comparison of Real and Predicted Values (Basic Model)" and labeled the x-axis as "Samples" and the y-axis as "Traffic Situation".

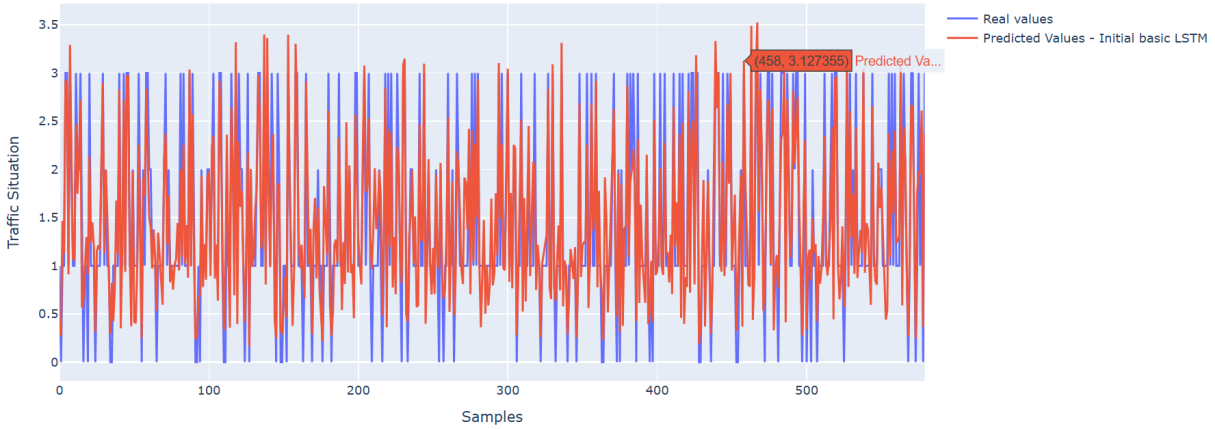


FIGURE 4.12 – Basic Model : Comparison of Real and Predicted Values

Here, a comparison is made between the actual traffic situation values (y_test) and the predicted values (predictions_base) generated by our basic LSTM model. Initially, a pandas DataFrame named comparison_df is created with two columns : 'Real Values' for the true labels and 'Predicted Values' for the model's predictions. The flatten() method is used to convert both arrays into 1D arrays to ensure they can be effectively integrated into the DataFrame. Following this, a random sample of 10 rows is selected from comparison_df using the sample(10) method.

	real values	Predicted Values
570	3.0	2.624698
329	1.0	0.726044
84	1.0	1.079564
78	1.0	1.175381
416	1.0	0.534436
184	1.0	1.105114
501	1.0	1.186373
66	1.0	1.204182
307	1.0	0.787683
72	1.0	1.120828

FIGURE 4.13 – Basic Model : Real Values and Predicted Values

4.2.2.4 Model Evaluation

Model evaluation is a critical step in the machine learning pipeline, providing insights into how well a model performs on unseen data. The evaluation process typically involves comparing the model's predictions to the actual values and computing various metrics to quantify the model's accuracy and reliability. In this example, we use several key metrics : Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared

Error (RMSE). These metrics provide a comprehensive view of the model's performance. MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. MSE, on the other hand, squares the errors before averaging, giving more weight to larger errors. RMSE, the square root of MSE, provides a metric in the same unit as the predicted values and is often used because it is more interpretable.

Additionally, we calculated key evaluation metrics, including Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE). We visualized these metrics using a bar chart to provide a clear view of the model's performance. The fig_metrics.update_layout method was employed to title this plot "Evaluation Metrics" and label the x-axis as "Metric" and the y-axis as "Value". This plot was saved as "base_metrics_evaluation.html". These visualizations and metrics provide a comprehensive overview of the model's accuracy and reliability, facilitating an in-depth evaluation of its predictive capabilities.

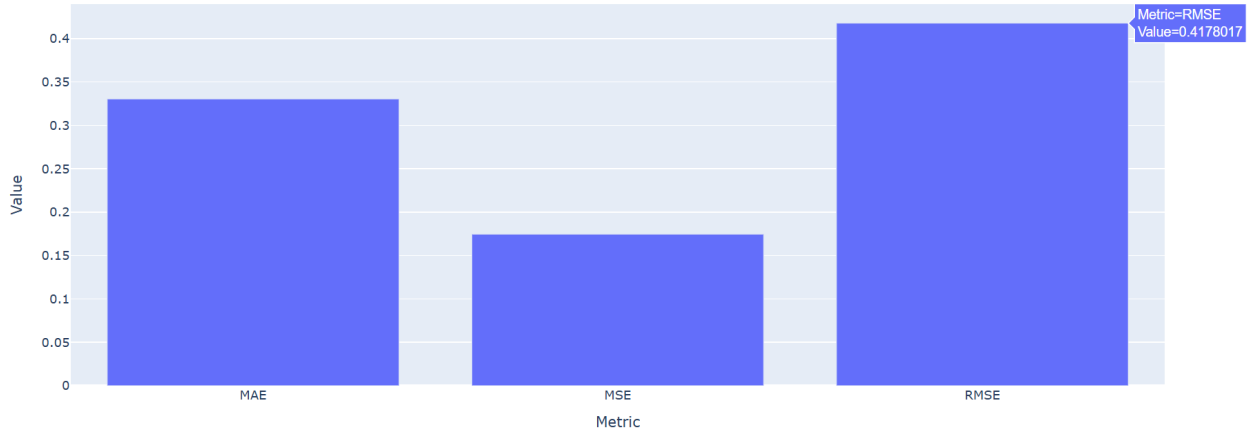


FIGURE 4.14 – Basic Model Evaluation based on RMSE, MAE, MSE values

The results of the Basic LSTM model show an RMSE of 0.397, an MAE of 0.31, and an MSE of 0.15. These values indicate that the model has a significant margin of error in its predictions. To improve the model further, we could consider optimizing hyperparameters such as the number of layers, units per layer, and dropout rates.

4.2.3 Improved LSTM Model

4.2.3.1 Model definition

In this section, we define an improved LSTM model using the Keras Sequential API. The model architecture includes three LSTM layers, each followed by a Dropout layer to reduce overfitting. The first two LSTM layers have 100 units, while the third LSTM layer has 50 units. The Dropout layers have a dropout rate of 30%. A final Dense layer with a single unit is used for output. The model is compiled with the Adam optimizer and the Mean Squared Error (MSE) loss function.

```
# Improved LSTM model definition
model_improved = Sequential()
model_improved.add(LSTM(100, return_sequences=True, input_shape=(1, X_reshaped.shape[2])))
model_improved.add(Dropout(0.3))
model_improved.add(LSTM(100, return_sequences=True))
model_improved.add(Dropout(0.3))
model_improved.add(LSTM(50))
model_improved.add(Dropout(0.3))
model_improved.add(Dense(1))

model_improved.compile(optimizer='adam', loss='mean_squared_error')
model_improved.summary()
```

FIGURE 4.15 – Definition of our improved model LSTM

4.2.3.2 Model training and Prediction

The improved model is trained using the fit() method. Training data (X_train and y_train) are used, with the process configured to run for 100 epochs and a batch size of 64. Validation data (X_test and y_test) are used to evaluate the model's performance during training. After training, the model makes predictions on the test data (X_test). The Root Mean Squared Error (RMSE) is calculated to evaluate the model's performance.

```
# model training
history_improved = model_improved.fit(X_train, y_train, epochs=100, batch_size=64, validation_data=(X_test, y_test), verbose=2)

# model prediction
predictions_improved = model_improved.predict(X_test)
rmse_improved = sqrt(mean_squared_error(y_test, predictions_improved))
```

FIGURE 4.16 – Code of our improved model training and prediction

4.2.3.3 Model Testing

A line plot to compare the real and predicted traffic situations by our improved model.

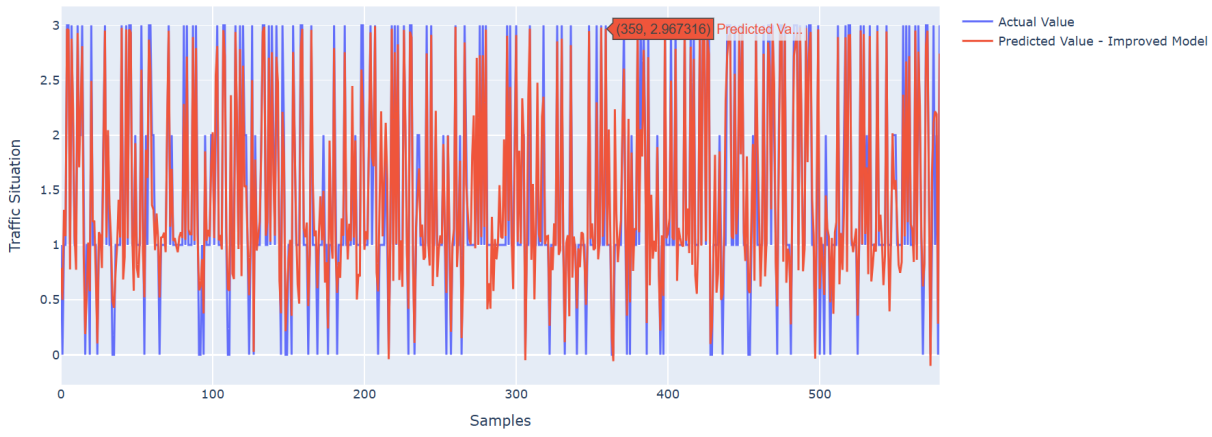


FIGURE 4.17 – Improved model : Comparison of Real and Predicted Values

Here, a comparison between the actual traffic situation values and the predicted values generated by our improved LSTM model.

	Actual Value	Predicted Value
77	1.0	0.901716
91	0.0	0.560345
533	3.0	2.906408
80	1.0	1.050363
345	1.0	0.759833
569	1.0	0.761129
194	2.0	1.831914
292	1.0	1.812472
218	3.0	3.023385
104	1.0	1.033510

FIGURE 4.18 – Improved model : Real Values and Predicted Values

4.2.3.4 Model Evaluation

Here too, we calculated key evaluation metrics, including Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE). We visualized these metrics using a bar chart to provide a clear view of the model's performance. The fig_metrics.update_layout method was employed to title this plot "Evaluation Metrics" and label the x-axis as "Metric" and the y-axis as "Value". This plot was saved as "base_metrics_evaluation.html". These visualizations and metrics provide a comprehensive overview of the model's accuracy and reliability, facilitating an in-depth evaluation of its predictive capabilities.

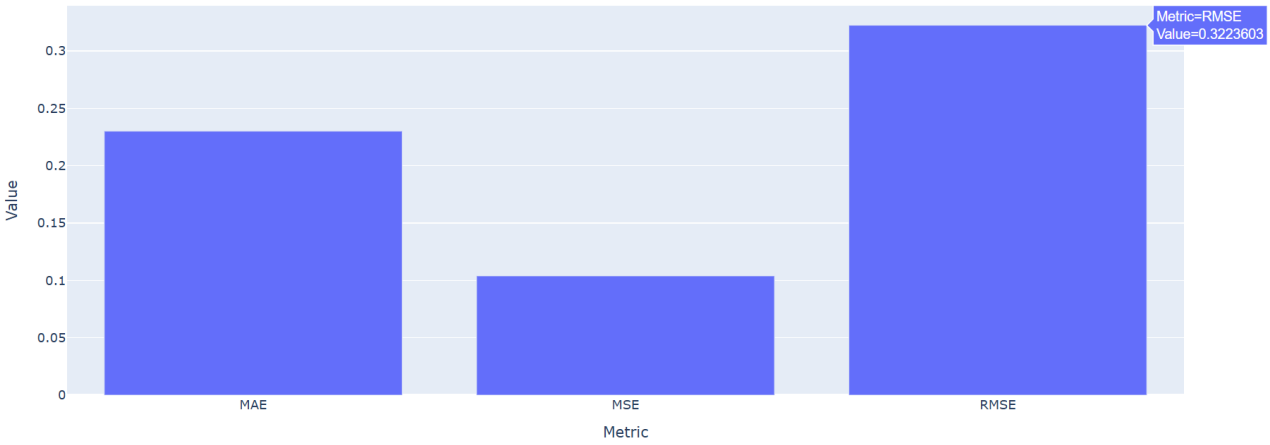


FIGURE 4.19 – Improved model : Basic Model Evaluation based on RMSE, MAE, MSE values

The Improved LSTM model shows a significant enhancement over the Basic LSTM model it demonstrates a notable reduction in prediction errors. This approve that optimizing hyperparameters, adjusting the network architecture, and adding dropout layers—have positively impacted the model's performance. The lower error metrics indicate more accurate and reliable predictions, showcasing the effectiveness of the enhancements applied to the LSTM model.

Model	**RMSE**	**MAE**	**MSE**
Basic LSTM	0.397	0.31	0.15
Improved LSTM	0.316	0.22	0.1

TABLE 4.1 – Comparison of Basic and Improved LSTM Models Performance

4.2.4 Combined LSTM Model with XGBoost

4.2.4.1 Model definition

The implemented script leverages various machine learning techniques for traffic prediction and management. It combines LSTM (Long Short-Term Memory) neural networks and XGBoost models to forecast traffic situations based on time, date, and vehicle counts. The LSTM architecture is designed with multiple layers and dropout regularization for enhanced prediction accuracy. The first two LSTM layers have 100 units, while the third LSTM layer has 50 units. The Dropout layers have a dropout rate of 30%. A final Dense layer with a single unit is used for output. The model is compiled with the Adam optimizer and the Mean Squared Error (MSE) loss function.

4.2.4.2 Model training and Prediction

The model training and prediction process involves the strategic design of an LSTM network, followed by combining its output with an XGBoost model to enhance predictive performance. The LSTM network consists of three main LSTM layers, each with 100 units in the first two layers and 50 units in the third. Each LSTM layer is followed by a Dropout layer with a rate of 0.3 to prevent overfitting. The final output layer is a Dense layer with a single unit, making it suitable for regression tasks. The LSTM model captures temporal dependencies in the data, while the Dropout layers add regularization. After training, the LSTM model's predictions are combined with those from an XGBoost model, which is trained on the same features but operates independently. The combined predictions, obtained by averaging the outputs of both models, leverage the temporal strengths of LSTM and the predictive power of XGBoost, resulting in improved evaluation metrics. This ensemble approach ensures that the complementary strengths of both models are utilized for better traffic situation forecasting.

```
xgb_model = xgb.XGBRegressor()  # Initialize the XGBoost model
xgb_model.fit(X_train.squeeze(), y_train)  # Train the XGBoost model on the actual labels

# Prediction on test data with the XGBoost model
predictions_xgb = xgb_model.predict(X_test.squeeze())
```

FIGURE 4.20 – Code of our combined model training and prediction

4.2.4.3 Model Testing

After training. The test data undergoes the same preprocessing steps as the training data, including scaling the features with the same MinMaxScaler and reshaping to match the input shape required by the LSTM model. The LSTM model then generates predictions for the test dataset, capturing the temporal dependencies it learned during training. Simultaneously, the XGBoost model, which was also trained on the same features but operates independently, makes its own predictions. To enhance predictive accuracy, the outputs from both models are averaged, combining the temporal strengths of the LSTM with the predictive power of XGBoost. This ensemble approach is evaluated using metrics such as (MAE), (MSE), and (RMSE). The combined model typically shows improved performance, demonstrating its effectiveness in accurately forecasting traffic situations by leveraging the strengths of both LSTM and XGBoost.

A line plot to compare the real and predicted traffic situations by our combined model :

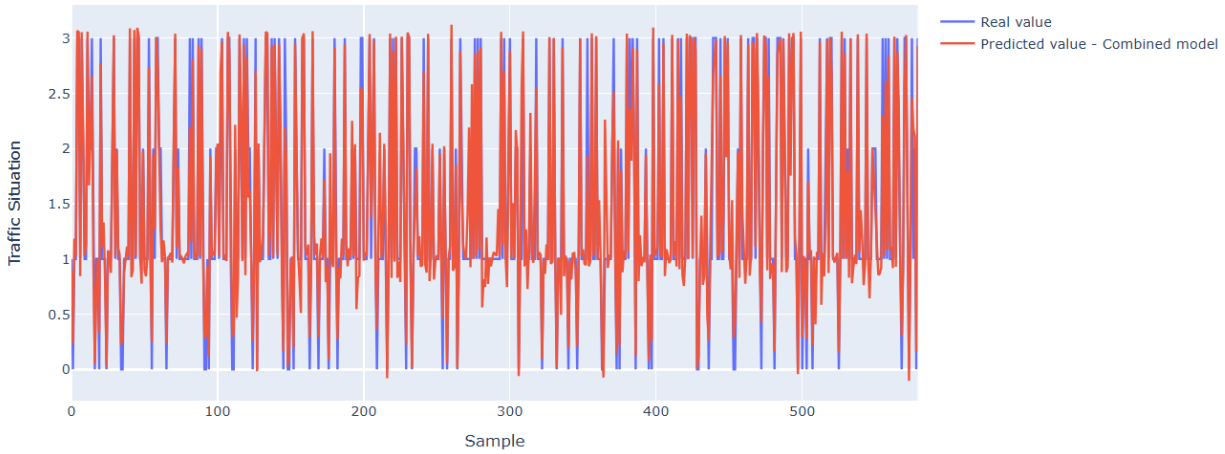


FIGURE 4.21 – Combined Model : Comparison of Real and Predicted Values

Here, a comparison between the actual traffic situation values and the predicted values generated by the combined LSTM and XGBoost model after averaging results.

	real values	Predicted Values
155	1.0	1.068664
132	2.0	1.932298
297	1.0	1.027954
197	1.0	0.844515
34	0.0	0.227470
490	1.0	0.937688
94	0.0	0.150836
58	3.0	2.998081
263	2.0	1.864565
227	2.0	2.031373

FIGURE 4.22 – Combined Model : Real Values and Predicted Values

4.2.4.4 Model Evaluation

Evaluation metrics include MSE, MAE, and RMSE (Root Mean Squared Error) calculated on the test dataset to assess prediction accuracy. Visualizations such as line plots and bar charts illustrate the comparison between actual and predicted traffic situations, highlighting the effectiveness of the combined LSTM and XGBoost approach in traffic prediction tasks.

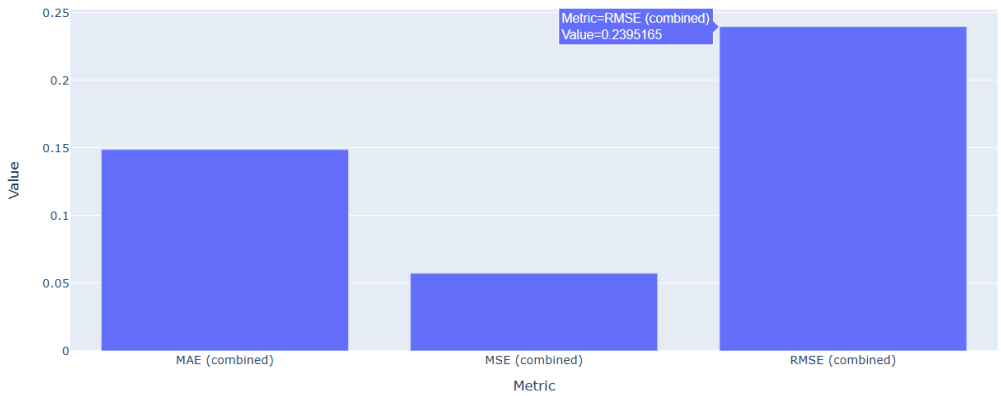


FIGURE 4.23 – Combined Model : Model Evaluation based on RMSE, MAE, MSE values

Moving to the Combined LSTM + XGBoost model outperformed the Basic LSTM and Improved LSTM models by achieving significantly lower RMSE, MAE, and MSE values. This improvement highlights the synergistic benefits of integrating LSTM for capturing temporal patterns and XGBoost for handling nonlinear relationships in traffic data. The combined approach resulted in more accurate predictions, making it a robust solution for enhancing short-term traffic forecasting accuracy.

4.2.5 Conclusion

The comparison of three LSTM models demonstrates a clear progression towards enhanced predictive accuracy in traffic situation forecasting.

Model	**RMSE**	**MAE**	**MSE**
Basic LSTM	0.397	0.31	0.15
Improved LSTM	0.316	0.22	0.1
Combined LSTM + XGBoost	0.23	0.14	0.05

Table 4.2 – Comparison of LSTM Models Performance

This integrated approach not only outperformed standalone LSTM variants but also capitalized on LSTM's ability to capture temporal dependencies and XGBoost's strengths in handling complex relationships and improving model stability through ensemble learning.

These results underscore the efficacy of combining Results of LSTM with those made by nXGBoost for achieving superior predictive performance in traffic forecasting applications, offering more accurate insights crucial for effective decision-making in urban transportation management and planning.

Chapter 5

Deployement

In this chapter, we will explore the implementation of a system that prioritizes vehicles equipped with cameras by simulating a traffic control mechanism using Arduino. This involves the detection of camera-equipped vehicles and dynamically adjusting traffic lights to green, ensuring a smoother and more efficient traffic flow for these priority vehicles.

We will begin by discussing the necessary hardware components, followed by the software implementation and coding. Finally, we will present and analyze the results of our simulation. This project aims to demonstrate how intelligent traffic systems can be enhanced through the integration of modern technology to improve urban mobility and safety.

5.1 Hardware Components

Detailed Overview of the Arduino-Based Traffic Light Simulation

In our Arduino-based traffic light simulation, we integrated various components to ensure precise functionality and control :

— **Arduino Uno Microcontroller** : Acts as the central controller, responsible for managing the sequencing and timing of the traffic light LEDs. It processes input signals from connected devices to dynamically adjust traffic light states.

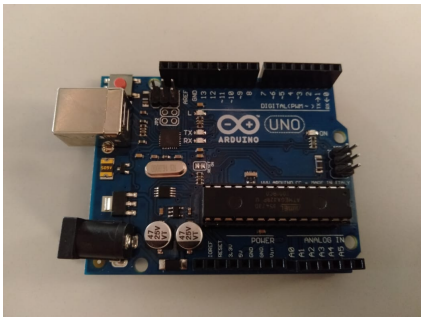


FIGURE 5.1 – Arduino Uno

— **USB Cable** : Links the Arduino Uno to a computer, providing both power and a communication interface for programming and data exchange. This connection is essential for real-time adjustments and updates to the traffic light system.

FIGURE 5.2 – USB Cable for Arduino UNO

— **Jumper Wires** : Serve as flexible connectors between the Arduino Uno, breadboards, and other components. These wires enable easy prototyping and circuit adjustments without the need for soldering, facilitating rapid development and testing.

FIGURE 5.3 – Jumper Wires

— **Resistors** : Play a crucial role in the circuit by limiting current flow through the traffic light LEDs. This protection ensures the longevity and proper operation of the LEDs, preventing them from burning out due to excessive current.

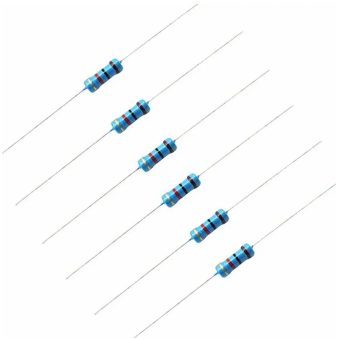


FIGURE 5.4 – Resistors

— **LEDs** : Represent the traffic light signals, with different colors (red, yellow, green) indicating various states of the traffic flow. LEDs provide clear visual feedback on the current traffic light sequence managed by the Arduino Uno.

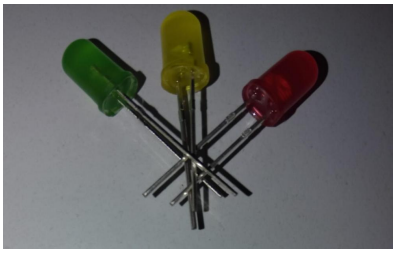


FIGURE 5.5 – LEDs

— **Breadboards** : Act as the platform for assembling and testing the circuitry. They allow for the temporary connection of components and facilitate quick modifications during the development process. Breadboards are instrumental in prototyping before finalizing the circuit design.

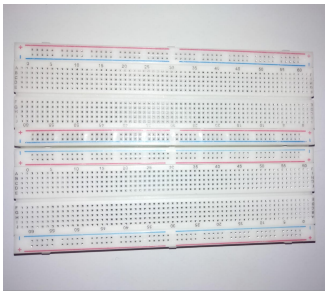


FIGURE 5.6 – Breadboards

— **Webcam Integration** : A webcam connected to a PC captures continuous video footage, which is processed using the YOLO (You Only Look Once) object detection algorithm. This system identifies emergency vehicles within the video stream based on predefined characteristics. Once an emergency vehicle is detected, the information is communicated to the Arduino Uno via serial communication protocols.

FIGURE 5.7 – Webcam

— **PC Internal Microphone Integration** : The PC's internal microphone monitors ambient audio for specific sound patterns associated with emergency vehicle sirens. Real-time audio analysis software processes these signals to detect sirens. Upon detection, the PC sends a signal to the Arduino Uno, prompting it to adjust the traffic lights to green, prioritizing the passage of emergency vehicles.

— **Small-Scale Replica of a Moroccan Police Car** : Integrating a small-scale replica of a Moroccan police car visually demonstrates the system's capability to detect and respond to emergency vehicles. This physical model enhances the simulation by providing a tangible representation of how the traffic lights adjust in real-time, showcasing the practical application of the integrated detection and control system.

FIGURE 5.8 – Small-Scale Replica of a Moroccan Police Car

This comprehensive setup illustrates the synergy between hardware components, real-time data processing, and microcontroller programming in enhancing traffic management systems through intelligent control mechanisms.

5.2 Software Implementation

The implemented Python script integrates OpenCV for video capture and object detection using the YOLOv8 model, specifically trained to identify ambulance, fire truck, and police vehicles. It utilizes a separate thread for real-time audio detection through PyAudio, monitoring for emergency vehicle sirens. Upon detecting

an emergency vehicle both visually and audibly, it sends a signal to an Arduino Uno via serial communication to adjust traffic lights, prioritizing emergency vehicle passage. The Tkinter GUI displays the processed video feed with overlaid detection results, showcasing the system's capability to enhance traffic management by dynamically responding to emergency situations.
In the implemented Python script, two main detection mechanisms are employed : visual detection using YOLOv8 and audio detection using PyAudio.

Visual Detection

The script utilizes the YOLOv5 model for real-time object detection of emergency vehicles including ambulances, fire trucks, and police vehicles. The model processes frames captured from a connected camera to identify these vehicles based on visual characteristics such as shape, color, and spatial context within the image. When a vehicle is detected with confidence exceeding 60%, a bounding box is drawn around it on the frame. If the detected vehicle corresponds to one of the predefined emergency vehicle classes (indices 0, 1, or 2), a message indicating "Emergency vehicle detected!" is printed, and the `visual_detected` flag is set to true.

Audio Detection

The audio detection function continuously records audio input from the PC's internal microphone using PyAudio. Audio data is processed in chunks, and the function `detecter_sirene` calculates the average power of the signal. It compares this average power against a predefined threshold (`SEUIL = 5000`) to detect the presence of a siren sound. When a siren sound is detected, the `audio_detected` flag is set to true, and a corresponding message "Siren detected!" is printed.

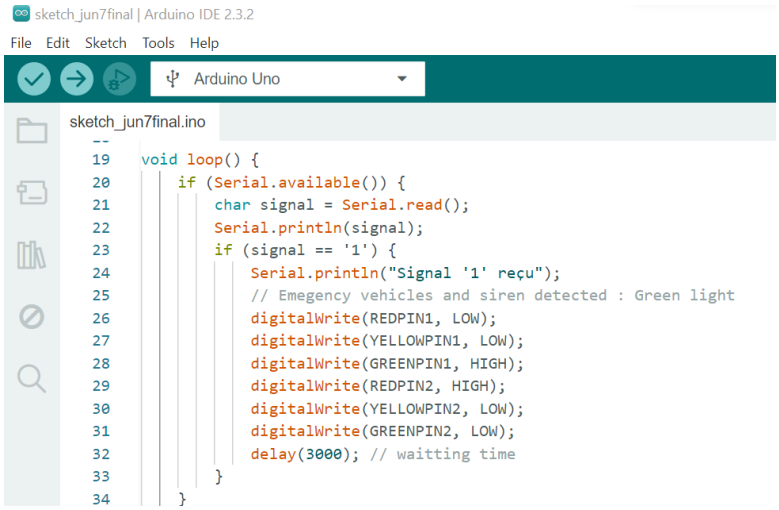
Integration and Response

The system integrates both visual and audio detection mechanisms to enhance traffic management. The following Python script demonstrates the integration of real-time visual and audio detection for emergency vehicles using OpenCV, YOLOv8, PyAudio, and Arduino. The script captures video from a camera, detects emergency vehicles visually with YOLOv8, and monitors for sirens using audio detection. When both visual and audio detections confirm the presence of an emergency vehicle with a siren, a signal (`b'1'`) is sent to an Arduino Uno via serial communication (`arduino.write(b'1')`). This signal instructs the Arduino to adjust traffic lights to prioritize the passage of the emergency vehicle. If no emergency vehicle with a siren is detected, a signal (`b'0'`) is sent (`arduino.write(b'0')`) to maintain normal traffic flow.

```python
def update_frame():
    global audio_detected
    ret, frame = cap.read()
    frame = cv2.resize(frame, (640, 480))
    result = model(frame, stream=True)
    visual_detected = False

    for info in result:
        boxes = info.boxes
        for box in boxes:
            confidence = box.conf[0]
            confidence = math.ceil(confidence * 100)
            Class = int(box.cls[0])
            if confidence > 60:
                x1, y1, x2, y2 = box.xyxy[0]
                x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 5)
                cvzone.putTextRect(frame, f'{classnames[Class]} {confidence}%', [x1 + 8, y1 + 100], scale=1.5, thickness=2)
                if Class in [0, 1, 2]:
                    print("Emergency vehicle detected  !")
                    visual_detected = True

    print(f"audio_detected: {audio_detected}, visual_detected: {visual_detected}")

    if visual_detected and audio_detected:
        print("Emergency vehicle with siren detected ! Sending the signal to the Arduino...")
        arduino.write(b'1')
        time.sleep(0.1)
        visual_detected = False
        audio_detected = False
    else:
        arduino.write(b'0')

    cv2image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGBA)
    img = Image.fromarray(cv2image)
    imgtk = ImageTk.PhotoImage(image=img)
    label.imgtk = imgtk
    label.configure(image=imgtk)
    label.after(10, update_frame)
```

Arduino Uno Control Logic for Traffic Light Adjustment

The void loop() function in our Arduino implementation serves as the central control mechanism for dynamically adjusting traffic lights based on real-time signals received via serial communication. This function continuously monitors incoming data from a connected computer, typically generated by a Python script that detects emergency vehicles using visual and audio cues. Upon receiving a specific signal (typically '1'), indicating the presence of an emergency vehicle with sirens detected, Arduino executes commands to switch the traffic lights. It transitions the relevant lights to green, allowing priority passage for emergency vehicles while halting cross traffic momentarily for safe traversal.



Here you can Find the Software Implementation codes : Python script that detects emergency vehicles using visual and audio cues and Arduino Uno Control Logic Code : Link of Software Implementation

5.3 Practical Experimentation

Step 1 : Using all the integrated materials, and YOLO object detection algorithm the system effectively prioritizes emergency vehicles during critical situations. When an emergency vehicle, identified by sirens and visual cues, approaches, the Arduino Uno swiftly adjusts the traffic lights to grant it a clear path.

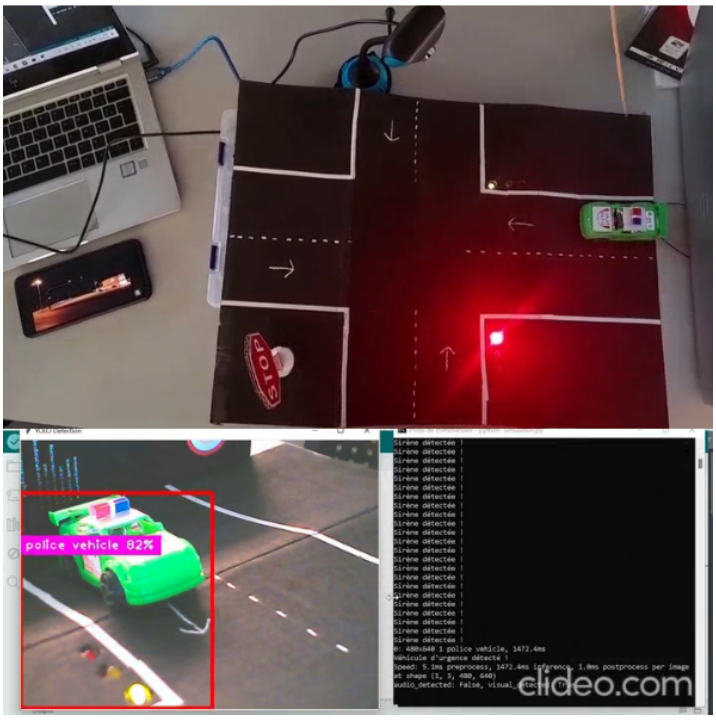


FIGURE 5.9 – Emergency Vehicle Detection and Priority Response

Step 2 : Once the emergency vehicle passes through the intersection and the siren stops, the system verifies the road's safety before proceeding.

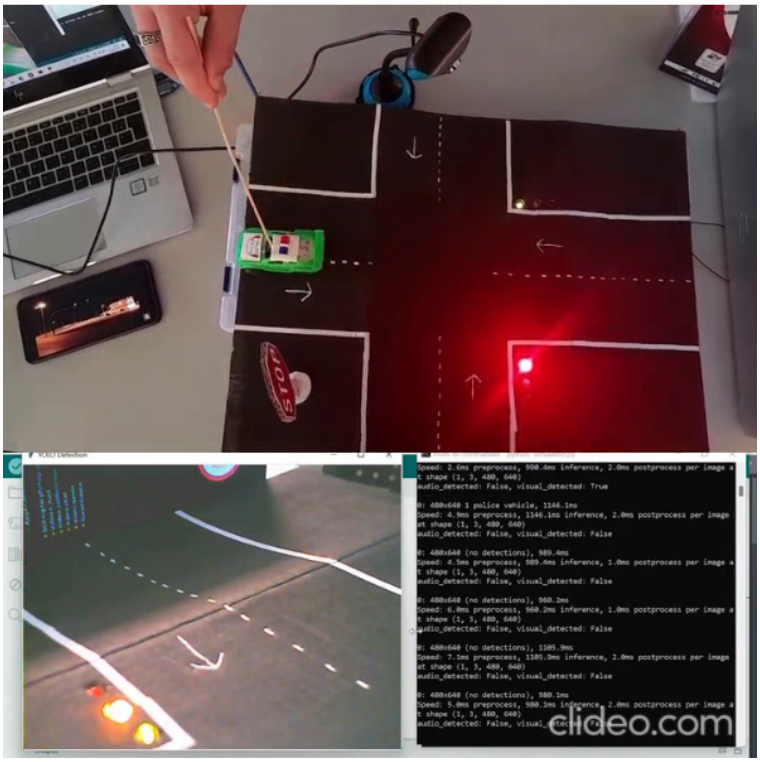


FIGURE 5.10 – Road Clearance Verification

Step 3 : After confirming that the emergency vehicle has passed and there is no siren , the traffic light returns to its regular mode to resume normal traffic operations.

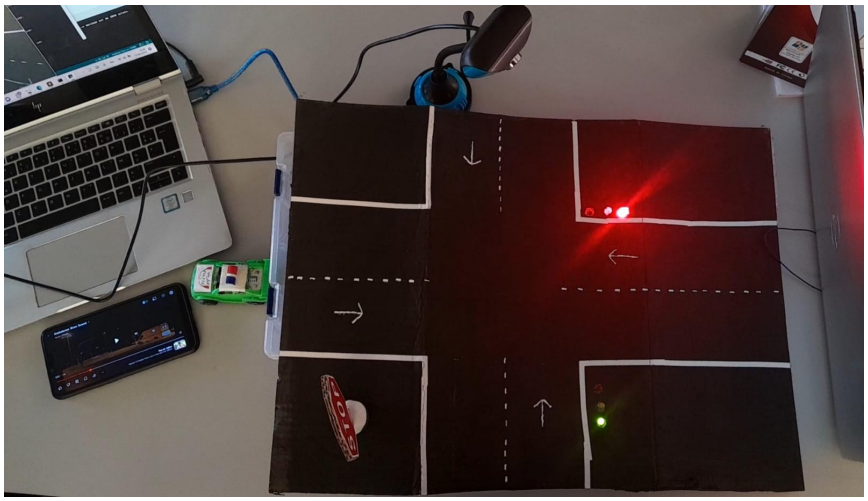


FIGURE 5.11 – Transition to Regular Traffic Lights Mode

Here You can Find the Video of the Simulation in Real-Time : Link of Real-Time Simulation

5.4 Conclusion

In summary, we have successfully developed and implemented an intelligent traffic management system that seamlessly integrates hardware components with advanced software algorithms. Our Arduino-based traffic light controller, coupled with visual and audio detection mechanisms, effectively prioritizes emergency vehicles

in real-time traffic scenarios. The system's ability to swiftly adjust traffic signals upon detecting emergency vehicles, and then revert to normal operations once the situation is resolved, demonstrates its practical utility in enhancing both emergency response times and overall traffic flow efficiency.

Chapter 6

Conclusion

6.1 Summary of the Project's Objectives and Accomplishments

This project successfully developed an intelligent traffic management system that prioritizes emergency vehicles using the YOLOv8 object detection algorithm and predicts traffic congestion through LSTM and XGBoost models. The results demonstrated that the system has the potential to significantly improve emergency response times and overall traffic flow. Despite the challenges encountered, the integration of machine learning and hardware components proved effective in creating a robust and reliable solution.

The future work will focus on addressing the identified problems, integrating more real-time data sources, and expanding the system's capabilities to handle a broader range of scenarios. Overall, this project lays a solid foundation for the development of advanced intelligent traffic management systems, promising substantial benefits for urban mobility and public safety.

6.2 Difficulties Limitations

Throughout the development and implementation of this project, several challenges and limitations were encountered. The inconsistency in data quality affected the accuracy of both the vehicle detection and traffic prediction models. Synchronizing different hardware components, such as Arduino boards and cameras, presented difficulties that needed careful management to ensure reliable operation. Real-world testing revealed occasional delays in the system's response time , which could potentially impact its effectiveness in critical situations. Additionally, there were challenges related to the scarcity of Moroccan-specific data and external sources that could assist in our research. Furthermore, both the internal and external cameras exhibited slow detection speeds and the low quality and the positioning of our camera for a perfect view, which sometimes results in very approximate detection

6.3 Future Work

In the realm of future enhancements, our intelligent traffic management system will focus on integrating real-time data sources such as live traffic feeds and emergency dispatch information. This integration aims to bolster the system's agility and accuracy in responding to dynamic traffic conditions and emergency incidents promptly. Expanding the system's capabilities beyond its current focus on emergency vehicle detection and traffic congestion prediction presents an opportunity to handle a broader range of emergency scenarios, including natural disasters and public events. Additionally, exploring advanced machine learning techniques such as reinforcement learning holds promise for refining traffic prediction models, enabling the system to adapt and optimize traffic flow in real-time based on evolving environmental and operational conditions. Establishing a comprehensive simulation environment will be instrumental in testing various system configurations and scenarios, ensuring robust performance and facilitating iterative improvements. Deploying the system in larger urban environments will provide critical insights into scalability and effectiveness, guiding further enhancements and optimizations tailored to diverse urban settings.Lastly, incorporating future versions of YOLO for enhanced object detection capabilities promises to further elevate the system's precision and efficiency in vehicle detection within complex traffic environments.

6.4 General Conclusion

This project represents a significant step forward in the development of intelligent traffic management systems, addressing two critical aspects of urban mobility : emergency vehicle detection and traffic congestion prediction. By leveraging cutting-edge technologies and innovative approaches, we have created a comprehensive solution that has the potential to transform urban traffic management and emergency response. Our work on emergency vehicle detection, utilizing the advanced YOLOv8 algorithm, demonstrates the power of modern object detection techniques in real-world applications. The ability to accurately and swiftly identify ambulances, police cars, and fire trucks in complex traffic scenarios is crucial for ensuring rapid response times and potentially saving lives. This system not only enhances road safety but also contributes to the overall efficiency of emergency services in urban environments.

The traffic congestion prediction component of our project, combining LSTM networks with XGBoost, showcases the potential of hybrid machine learning approaches in tackling complex, time-dependent problems. By accurately forecasting traffic conditions, this system enables proactive traffic management, potentially reducing commute times and improving the overall quality of urban life. The integration of this predictive capability with the emergency vehicle detection system creates a synergistic solution that can dynamically adjust traffic flow to prioritize emergency responses while managing general congestion.

Furthermore, the incorporation of physical hardware components, such as Arduino boards, cameras, and sound detectors, bridges the gap between theoretical models and practical implementation. This integration demonstrates the feasibility of deploying such advanced systems in real-world settings, paving the way for future smart city initiatives. In the broader context of urban development and smart city technologies, our project aligns with global efforts to create more efficient, safer, and more livable urban environments. As cities worldwide grapple with increasing population densities and the challenges of modernizing infrastructure, solutions like ours offer a glimpse into the future of urban mobility management.

While our project has achieved significant milestones, it also opens up numerous avenues for future research and development. The potential for integrating real-time data sources, expanding to additional emergency scenarios, and exploring more advanced machine learning techniques suggests that this field of study remains rich with possibilities. In conclusion, our work not only contributes to the academic understanding of traffic management and emergency response systems but also offers practical solutions to real-world challenges. As we look to the future, the continued development and refinement of such systems will play a crucial role in shaping smarter, safer, and more efficient cities for generations to come.

Bibliography

[1] Les réseaux de neurones récurrents | Reccurent neural network - Deep learning - - Kongakura. https ://kongakura.fr/article/Les-reseaux-de-neurones-reccurent.

[2] Bassant M. Elbagoury, Luige Vladareanu, Victor Vlădăreanu, Abdel Badeeh Salem, Ana-Maria Travediu, and Mohamed Ismail Roushdy. A Hybrid Stacked CNN and Residual Feedback GMDH-LSTM Deep Learning Model for Stroke Prediction Applied on Mobile AI Smart Hospital Platform. *Sensors*, 23(7) :3500, January 2023.

[3] Autoencoders in Deep Learning : Tutorial & Use Cases [2023]. https ://www.v7labs.com/blog/autoencoders-guide, https ://www.v7labs.com/blog/autoencoders-guide.

[4] Cnn Architecture Layers Cnn Architecture For Isolated. https ://www.vrogue.co/post/cnn-architecture-layers-cnn-architecture-for-isolated-2-and-3-digit.

[5] A Transfer-Learning-Based Approach for Emergency Vehicle Detection. *Eurasian Journal of Science and Engineering*, 8(1), 2022.

[6] Brief summary of YOLOv8 model structure. https ://github.com/ultralytics/ultralytics/issues/189.

[7] Road Safety in Morocco | Traffic accidents, crash, fatalities & injury statistics | GRSF. https ://www.globalroadsafetyfacility.org/country/morocco.

[8] Kaabeche Oussama. EXTRACTION DES ILOTS PAR DEEP LEARNING.

[9] What is YOLOv8 ? The Ultimate Guide. [2024]. https ://blog.roboflow.com/whats-new-in-yolov8/.

[10] LSTM network : A deep learning approach for short-term traffic forecast - Zhao - 2017 - IET Intelligent Transport Systems - Wiley Online Library. https ://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/iet-its.2016.0208.

[11] Pv4 Image Dataset. https ://universe.roboflow.com/national-school-of-applied-science-zh1i1/pv4/dataset/1.

[12] Traffic Prediction (100 % test accuracy). https ://kaggle.com/code/munnafkoilakuntla/traffic-prediction-100-test-accuracy.

[13] Robert E. Banfield, Lawrence O. Hall, Kevin W. Bowyer, and W.P. Kegelmeyer. A Comparison of Decision Tree Ensemble Creation Techniques. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(1) :173–180, January 2007.

[14] Research on Traffic Congestion Prediction Based on XGBoost. *Frontiers in Traffic and Transportation Engineering*, 4(1), 2024.

[15] Precision and recall. *Wikipedia*, June 2024.

[16] IEEE Xplore - Temporarily Unavailable. https ://ieeexplore.ieee.org/abstract/document/4016560/.