**Design and Development of Vehicular Attacks using Hybrid Deep Learning Architectures**

**SYNOPSIS**

A real-time traffic congestion prediction system employs a hybrid deep learning approach, leveraging historical traffic data to forecast congestion levels without explicit labeling. By autonomously learning from data patterns, the algorithm identifies correlations between various factors such as traffic volume, road conditions, time of day, and weather. Integrating multiple deep learning techniques enables the algorithm to capture complex interactions effectively. These models process vast amounts of data, facilitating accurate predictions of future congestion levels. With its agile and responsive nature, the system enables dynamic traffic management strategies, fostering smoother traffic flow and enhancing urban mobility without manual data labeling. he real-time traffic congestion prediction system utilizes advanced deep learning techniques to forecast congestion levels autonomously, without the need for explicit labeling of data points. By analyzing historical traffic data, the algorithm identifies intricate patterns and relationships among different factors influencing congestion, including traffic volume, road conditions, time of day, and weather. Through the integration of various deep learning models, the system effectively captures the complex interactions between these factors. These models are adept at processing large volumes of data, enabling them to make accurate predictions about future congestion levels in real-time. One of the key advantages of this approach is its agility and responsiveness. By continuously learning from incoming data, the algorithm can adapt to changing traffic conditions and make swift adjustments to traffic management strategies. This dynamic capability contributes to smoother traffic flow and improved urban mobility. Furthermore, the system's reliance on deep learning methods reduces the need for manual intervention and labeling of data points, streamlining the prediction process and enhancing efficiency. Overall, the hybrid deep learning approach employed by the system represents a significant advancement in traffic management technology, offering more accurate predictions and enabling more effective congestion mitigation strategies.

**ORGANISATION PROFILE**

1.1 **ORGANIZATION PROFILE:**

ABE Semiconductor is a reputable semiconductor design company that offers a range

of services and has a track record of delivering quality designs to its customers.

**SERVICES OFFERED:**

**Cognitive Transformation**

Providing end to end Artificial Intelligence based tools and software for complete

business automation. Our AI services not limited to

**Data and Feature Engineering(Data Analysis)**

Deployment of ML/DL on your Existing Solutions and to make the complete AI

ecosystem

AI Apps - Mobile Apps which can be deployed with AI Features.

**Smart Product Solutions**

Providing The End To End Solutions For Smart Products Designs And Development

Which Can Be Catalyst For The Future Innovation. Undefined Boundary Of

Technologies Ranges From Embedded Systems, To IoT (Edge Computing) Etc.

**Educational Services**

Building Industrial Ecosystem at Institutions With Cost Effective Solutions And

Converting The Each Institutional Environment For Building The Start-Ups. Our

Institutional Alliance Programs Serves This Purpose.

**Technical Consulting**

Providing Technical Consulting To Companies And Institutions For Scaling,

Developing And Tracking.

**Technologies Used**

**1)CLOUD**

AWS, AZURE, IBM BLUE MIX, THINGSPEAK, GOOGLE CLOUD, FIREBASE.

**2)EMBEDDED AND IOT BOARDS**

ARM, CORTEXM-X , CORTEXA-X , TEXAS, RUGGED BOARDS, Etc

**3)ELECTRIC VEHICLES**

CAN, OTA, GREEN WIFI, BACKEND SERVERS, EMBEDDED XML PLC

,BLE5.0

**4)AI ALGORITHMS**

PYTHON, PAPER SPACING, TENSORFLOW, PYTORCH, KERAS, BERT,Etc.

**5)COMPUTER VISION AND SYSTEMS**

OPENCV, YOLO MODELS

**6)ANDROID MOBILE APP**

FLUTTER, JAVA, REACT NATIVE (IOS TO BE SOON)

**7)BACKEND SERVERS**

FIREBASE, PYTHON, PHP, NODE JS

**8)DATABASE MANAGEMENTS**

MANGODB, DJANGO, FIREBASE, NOSQL

**9)WEB APPS**

HTML, WORDPRESS, CSS, IONIC.

**SYSTEM ENVIRONMENT**

2.1 Hardware Requirements:

Processor : Intel Core i4 (10th Gen)

Ram : 4.0 GB

2.2 Software Requirements

Operating System : Windows 10

Framework : Googlecolab

Language : python

**2.3 About the technology:**

**Python:**

Python is highly praised for its simplicity, readability, and versatility, making it a top choice among developers. With its vast array of libraries and frameworks, Python supports a wide spectrum of applications spanning from web development to data analysis and artificial intelligence. Its focus on code readability enables developers to succinctly express complex ideas, fostering the creation of efficient and maintainable codebases across different programming paradigms, including procedural, object-oriented, and functional programming.

**Google Colab:**

Google Colab, also referred to as Google Colaboratory, provides a cloud-based platform for writing and running Python code directly within a web browser. Offered by Google, it offers a free environment with access to robust hardware resources such as GPUs and TPUs, which greatly facilitates the training of machine learning models. Google Colab seamlessly integrates with Google Drive, simplifying access to notebooks and datasets. Its collaborative features, such as real-time editing and commenting, make it well-suited for teamwork and educational endeavors. Additionally, Google Colab comes pre-installed with popular Python libraries like NumPy, pandas, matplotlib, and scikit-learn, streamlining the development and deployment of machine learning workflows.

**Scikit-learn:**

Scikit-learn stands out as a prominent open-source machine learning library tailored for Python users. It provides a comprehensive suite of tools for various machine learning tasks, encompassing classification, regression, clustering, and dimensionality reduction. Leveraging foundational scientific computing libraries like NumPy, SciPy, and matplotlib, scikit-learn seamlessly integrates into Python workflows. Its intuitive API and extensive documentation cater to both novice and seasoned practitioners, facilitating model development, evaluation, and refinement. With implementations of widely-used machine learning algorithms and utilities for data preprocessing, model evaluation, and hyperparameter tuning, scikit-learn serves as a valuable asset for advancing machine learning capabilities within the Python ecosystem.

**EXISTING SYSTEM:**

One of the primary disadvantages of the existing system is that it may lack novelty. The paper is a review article that summarizes previous work on predicting traffic congestion using machine learning techniques. As such, it may not present any new or original research findings. It only covers a limited number of machine learning techniques, such decision trees. This means that other relevant techniques, such as ensemble methods, are not included in the analysis. The existing system does not compare the proposed approach with other state-of-theart methods for predicting traffic congestion, which makes it difficult to assess the relative performance of the proposed approach compared to other approaches. By efficiently distributing vehicle flow throughout the road network, the problem of maximising travel time can be approached from a different perspective. This approach aims to allocate the traffic flow across the network in a balanced manner to reduce congestion and shorten travel time. Urban mobility simulation is used to estimate traffic density in large city areas, leveraging the Intelligent Transport System (ITS) which relies on Information and Communication Technology to facilitate communication to improve traffic amenity, security, and efficiency but it resulted in inconvenience of prediction of traffic. Initially, the evaluation of traffic light algorithms was carried out by integrating them directly into the core of the simulation. However, over time, it has become increasingly difficult to maintain this approach. Upon extracting the API, they assess the level of congestion among vehicles, which is often more prevalent in bottleneck areas. In addition, we also examine the surrounding roads to identify opportunities to redistribute the congestion from highly congested roads to less crowded ones. It is challenging to generalize studies on predicting traffic congestion because they often use different models. While many studies share common factors like the study area, data collection period, predicted parameters, prediction intervals, and validation procedures, they often differ in their congestion estimation methods. One typical approach is to forecast traffic flow parameters, such as traffic speed, density, and congestion index. The Congestion Index (CI) method is especially useful for continuously monitoring congestion levels in a spatiotemporal dimension. When evaluating their results against ground truth values or other models, many studies commonly use the mean absolute error (MAE) as a performance metric. The prevention of traffic congestion has been made possible with the aid of a Traffic Management System (TMS), which utilizes a convolutional neural network to process input video sequences. The training process involves implementing a convolutional neural network topology that is built using the YOLO Algorithm. This technique functions by non-spatial detecting objects in a video frame and is used as input for many tracking algorithms. To create a comprehensive simulation of traffic patterns, we have considered various factors such as emissions, acceleration, and deceleration behaviors, and have incorporated different types of personal vehicles into our model. In addition, we have included public transport options, such as bus stops and special lanes, to reflect a more realistic distribution of vehicles. To showcase the versatility of intermodal traffic, we have supplemented the Acosta scenario with fictional personal trips. The simulation progresses in discrete time intervals of 1 second, while vehicle positions are continuously represented in space. Each vehicle's location is defined by its distance from the start of its lane and the lane it occupies. To determine each vehicle's speed while traveling through the network, a car-following model is used. The ACTIVITYGEN tool is designed to create traffic demand scenarios by utilizing a road network's definition and population description. This software utilizes an activity-based traffic model that takes into account various transportation modes including buses, cars, bicycles, and pedestrians, in order to determine daily activities such as work, school, and leisure time through multi-modal planning. The tool doesn’t takes into account the availability of various transportation modes and generates traffic demand by simulating people's daily activities and transportation choices. The city's public transport database was utilized to collect data on bus routes, which was then used to develop a comprehensive traffic demand model. This information was used in conjunction with the activity-based traffic model to create realistic traffic scenarios for the simulation.

**PROPOSED SYSTEM:**

The proposed system for real-time prediction of traffic congestion levels harnesses the power of advanced deep learning techniques, specifically integrating Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN) algorithms to achieve a heightened level of accuracy. At its core, the system relies on the analysis of vast historical traffic data, meticulously examining various factors such as traffic volume, road conditions, time of day, and weather conditions.

Unlike traditional methods that often require manual labeling of data points, this system operates autonomously, leveraging the inherent patterns and relationships within the data to make predictions without explicit labels. To begin, the system initiates with the collection of extensive historical traffic data from a multitude of sources including traffic sensors, cameras, and weather stations. This data undergoes a rigorous preprocessing phase, where techniques are employed to clean, normalize, and standardize the data, ensuring consistency and removing any inconsistencies or outliers that could potentially distort the accuracy of predictions. Following preprocessing, the data is fed into two distinct deep learning models: the LSTM and CNN. The LSTM model is adept at capturing temporal dependencies and long-term patterns in sequential data, such as traffic volume fluctuations over time. On the other hand, the CNN model excels at extracting spatial features from static data, such as road conditions and weather patterns. What sets this system apart is its innovative approach of combining the strengths of both LSTM and CNN models into a hybrid deep learning architecture.

By leveraging the complementary capabilities of these models, the hybrid architecture is able to effectively capture the complex interactions and dependencies present in the data, leading to more accurate predictions of traffic congestion levels. Once trained on the historical data, the hybrid model is capable of making real-time predictions about traffic congestion levels. Continuous streams of incoming data, including current traffic conditions and weather updates, are seamlessly integrated into the model, enabling it to adapt and evolve in response to changing conditions. Furthermore, the performance of the system is continuously monitored and evaluated using a variety of metrics to ensure its effectiveness and accuracy. Fine-tuning and optimization techniques are applied based on real-world feedback, allowing the system to continuously improve and refine its predictions over time.

In summary, the proposed system represents a cutting-edge approach to real-time traffic prediction, leveraging advanced deep learning techniques to enable more responsive and agile traffic management strategies. By accurately predicting congestion levels without the need for manual labelling of data points, the system contributes to smoother traffic flow and improved urban mobility, ultimately enhancing the overall quality of life in urban environments.

**Advantages of the proposed traffic congestion prediction system:**

1. Enhanced Prediction Accuracy and Precision:

Through the integration of LSTM and CNN algorithms, the system achieves exceptional accuracy and precision in predicting traffic congestion levels. By analyzing historical data without explicit labeling, it autonomously learns intricate patterns and relationships, enabling more reliable predictions and facilitating precise traffic management strategies.

2. Adaptability to Dynamic Traffic Dynamics:

Engineered to adapt dynamically to changing traffic conditions, the system continuously updates its models based on real-time data. This adaptability ensures robust predictions in response to evolving traffic patterns, such as fluctuations in traffic volume, road conditions, and weather, enhancing overall traffic management effectiveness.

3. Comprehensive Feature Representation:

Leveraging advanced feature extraction techniques within LSTM and CNN models, the system effectively captures diverse factors influencing traffic congestion. By considering spatial and temporal dependencies in the data, it enhances prediction sensitivity and reliability, leading to more accurate congestion forecasts.

4. Transparency and Interpretability:

With a focus on transparency and interpretability, the system provides insights into the factors driving congestion predictions. By offering visibility into the underlying features and patterns, it enables traffic managers to understand and trust the predictions, facilitating informed decision-making and proactive traffic management strategies.

5. Robustness Through Hybrid Deep Learning:

The hybrid deep learning architecture combines the strengths of LSTM and CNN models, enhancing robustness against variability in traffic conditions. By mitigating biases and uncertainties, the system generates consistent and reliable predictions, essential for effective traffic management and decision support.

6. Scalability and Deployment Flexibility:

Designed for scalability, the system can adapt to various urban environments and traffic infrastructures. Whether deployed in small cities or metropolitan areas, it accommodates different data volumes and computational resources, ensuring broad applicability and scalability across diverse deployment scenarios.

7. Real-time Prediction and Rapid Response:

With its efficient implementation and advanced algorithms, the system provides real-time congestion predictions, enabling rapid response to traffic events. By continuously analyzing traffic data streams, it offers immediate alerts and recommendations, empowering traffic managers to take timely actions to mitigate congestion.

8. Proactive Traffic Management and Optimization:

By accurately predicting congestion levels in advance, the system supports proactive traffic management strategies. Through rerouting and signal optimization, it helps alleviate congestion before it escalates, contributing to improved traffic flow, reduced congestion-related delays, and enhanced urban mobility.

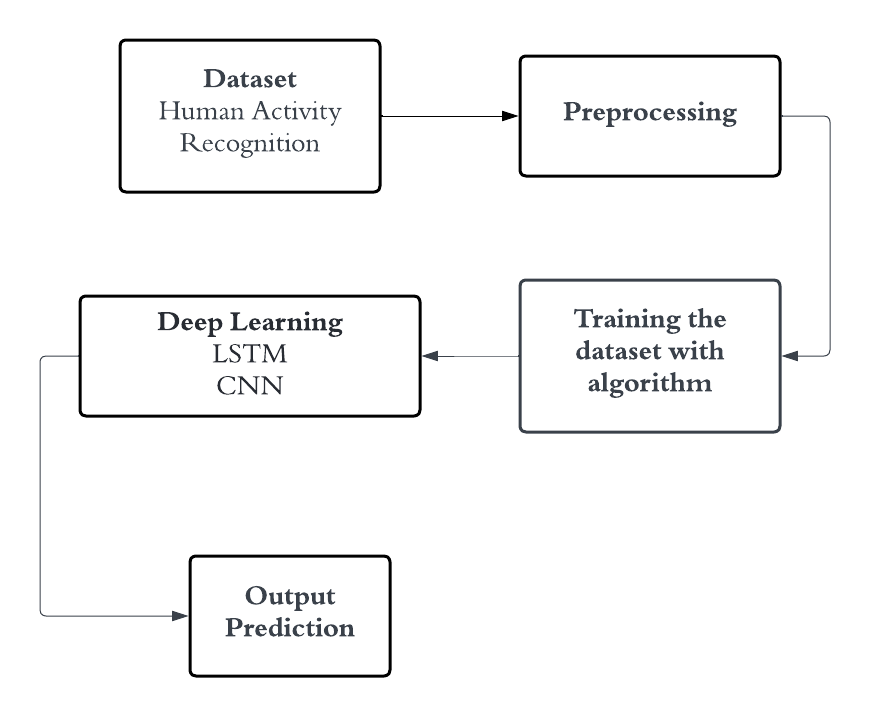
9. Continuous Learning and Improvement:

The system facilitates continuous learning and improvement through ongoing analysis of real-world traffic data. By monitoring performance metrics and user feedback, it identifies areas for enhancement and updates its prediction models accordingly, ensuring continual improvement in prediction accuracy and effectiveness.

10. Alignment with Smart City Objectives:

Aligned with the goals of smart city initiatives, the system leverages advanced technologies to optimize urban transportation systems. By providing accurate congestion predictions and supporting proactive traffic management, it contributes to the development of more efficient, sustainable, and livable cities, ultimately enhancing the quality of life for residents and visitors alike.

**SYSTEM DESIGN:**



**Dataset Description:**

**ID:** The purpose of this field is to provide a singular identification number for a vehicle

or agent in the simulation. A unique ID is necessary for tracking and analysis of each

vehicle or agent in the simulation.

**Type:** This column represents the type of vehicle or agent in the simulation. Examples

could include cars, buses, bikes, pedestrians, etc. Knowing the type of vehicle or agent

in the simulation can help with understanding traffic flow patterns and interactions

between different types of vehicles or agents.

**Depart:** This attribute represents the time at which the vehicle or agent departs from

its starting point. This data can be used to track the travel time for each vehicle or agent

and to identify any congestion or delays in the simulation.

**Depart Lane:** This field represents the lane from which the vehicle or agent departs.

In some simulations, there may be multiple lanes available for vehicles or agents to

depart from. Knowing which lane each vehicle or agent departs from can help with

understanding the flow of traffic and identifying any bottlenecks or congestion points.

**From:** This column represents the starting location of the vehicle or agent. This data

can be used to track the travel distance and time for each vehicle or agent.

**To:** This field represents the destination location of the vehicle or agent. This data can

be used to track the travel distance and time for each vehicle or agent and to identify

any common routes or congestion points.

**Depart Speed:** This data represents the speed at which the vehicle or agent departs.

This data can be used to track the speed of each vehicle or agent and to identify any

areas of congestion or slow traffic flow.

Overall, these data fields can be used to track and analyse the movement of vehicles or agents in a simulated urban environment. By analysing this data, researchers and urban planners can gain insights into traffic flow patterns, identify areas of congestion, and develop strategies for improving transportation efficiency and reducing congestion in real-world urban environments.

**Pre-processing:**

**Standard Scaling :**

In the proposed traffic congestion prediction system, standard scaling is employed as a pre-processing technique to ensure consistency and normalization of the input data before feeding it into the deep learning models, particularly LSTM and CNN.

Standard scaling, also known as Z-score normalization, involves transforming the data such that it has a mean of 0 and a standard deviation of 1. This process is essential for bringing features to a similar scale, preventing certain features from dominating the learning process due to their larger magnitudes.

1. Data Collection: Traffic data, including variables such as traffic volume, road conditions, time of day, and weather conditions, is collected from various sources such as sensors, cameras, and weather stations.

2. Feature Selection: Relevant features that are indicative of traffic congestion are selected for analysis. These features may include traffic volume, road occupancy, weather conditions, and historical traffic patterns.

3. Data Preprocessing:

a. Standard Scaling: Each feature is standardized independently by subtracting the mean and dividing by the standard deviation. This ensures that each feature has a mean of 0 and a standard deviation of 1.

b. Standardization Formula:

( X\_{text{standardized}} = frac{X - text{mean}(X)}{text{std}(X)} )

Where:

- ( X\_{text{standardized}} ) is the standardized feature.

- ( X ) is the original feature.

- ( text{mean}(X) ) is the mean of the original feature.

- ( text{std}(X) ) is the standard deviation of the original feature.

4. Data Integration: The standardized features are integrated into the dataset, ready for further analysis and modeling.

By applying standard scaling as a pre-processing technique, the proposed system ensures that the input data fed into the deep learning models are on a consistent scale, facilitating convergence during training and improving the overall performance and accuracy of the congestion prediction models.

**Deep learning algorithm:**

**Architecture and Components of LSTM:**

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) architecture designed to overcome the vanishing gradient problem in traditional RNNs. The architecture consists of several key components:

1. Cell State: The cell state serves as the memory of the LSTM unit, allowing information to flow across different time steps without degradation. It is regulated by various gates to selectively add or remove information.

2. Input Gate: The input gate regulates the flow of information into the cell state. It decides which information from the current input should be stored in the cell state.

3. Forget Gate: The forget gate decides which information from the previous cell state should be discarded or forgotten. It controls the retention of relevant information over time.

4. Output Gate: The output gate determines which information from the current cell state should be passed to the output. It regulates the information flow to the next time step or to the final prediction.

**Working Principle of LSTM:**

LSTM units process sequential data by selectively updating and passing information through time steps while preserving long-term dependencies. The working principle involves the following steps:

1. Input Processing: Incoming data is fed into the LSTM unit, where it is multiplied by weights and passed through activation functions to compute the input gate activation.

2. Forget Gate Calculation: The previous cell state and current input are combined and processed to compute the forget gate activation. This gate determines which information from the previous cell state should be retained or forgotten.

3. Update Cell State: The input and forget gates determine how much information should be added to or removed from the cell state, updating it for the current time step.

4. Output Calculation: The updated cell state is passed through the output gate activation to produce the output for the current time step or the final prediction.

**Advantages of LSTM:**

1. Long-Term Dependency Handling: LSTM networks are capable of learning and preserving long-term dependencies in sequential data, making them well-suited for tasks requiring memory over extended time horizons.

2. Mitigation of Vanishing Gradient Problem: The use of specialized gating mechanisms enables LSTMs to mitigate the vanishing gradient problem commonly encountered in traditional RNNs, facilitating more effective training and learning of temporal patterns.

3. Versatility and Applicability: LSTMs have demonstrated effectiveness across various domains, including natural language processing, time series analysis, speech recognition, and more, making them a versatile choice for sequential data modeling tasks.

**Challenges of LSTM:**

1. Computational Complexity: LSTMs can be computationally intensive, especially when dealing with large-scale datasets or deep architectures, requiring significant computational resources for training and inference.

2. Interpretability: Despite their effectiveness, LSTMs can be challenging to interpret due to their complex architecture and the intricate interactions between the various components, limiting their transparency and interpretability.

3. Overfitting: Like other deep learning architectures, LSTMs are susceptible to overfitting, especially when trained on small datasets or when the model architecture is overly complex. Regularization techniques and careful model selection are essential to mitigate this challenge.

**Applications of LSTM:**

1. Natural Language Processing (NLP): LSTM networks are widely used in NLP tasks such as language modeling, sentiment analysis, machine translation, and named entity recognition. They excel at capturing long-range dependencies in text data, enabling more accurate and context-aware language processing.

2. Time Series Prediction: LSTM networks are well-suited for time series forecasting tasks, including stock price prediction, weather forecasting, energy load forecasting, and traffic flow prediction. Their ability to capture temporal dependencies makes them effective in modeling and redicting sequential data patterns.

3. Speech Recognition: LSTMs are employed in automatic speech recognition (ASR) systems to transcribe spoken language into text. By processing audio input over time and capturing phonetic and linguistic features, LSTM-based models can achieve high accuracy in speech recognition tasks.

4. Handwriting Recognition: LSTMs are utilized in optical character recognition (OCR) systems to recognize and interpret handwritten text. By analyzing sequential data representing pen strokes, LSTMs can accurately recognize handwritten characters and convert them into digital text.

5. Health Monitoring and Diagnosis: LSTM networks are applied in healthcare for tasks such as patient monitoring, disease diagnosis, and medical signal analysis. They can process sequential data from medical sensors and time-series measurements to detect abnormalities, predict patient outcomes, and assist in clinical decision-making.

6. Autonomous Driving: LSTMs play a crucial role in autonomous driving systems for tasks such as trajectory prediction, object detection, and behavior prediction of other vehicles and pedestrians. By analyzing sequential data from sensors such as cameras and LiDAR, LSTM-based models can anticipate and react to dynamic traffic situations.

7. Music Generation: LSTM networks are used in music generation systems to compose or generate music sequences. By learning patterns and structures from existing music compositions, LSTM-based models can create original music pieces that mimic the style and characteristics of human-composed music.

8. Financial Forecasting: LSTMs are employed in financial applications for tasks such as stock price prediction, market trend analysis, and algorithmic trading. By analyzing historical financial data and market indicators, LSTM-based models can assist investors and traders in making informed decisions.

9. Video Analysis and Action Recognition: LSTMs are utilized in video analysis tasks such as action recognition, gesture recognition, and activity detection. By processing sequential frames of video data, LSTM-based models can identify and classify human actions and interactions in real-time.

10. Drug Discovery and Bioinformatics: LSTMs are applied in drug discovery and bioinformatics for tasks such as protein structure prediction, drug-target interaction prediction, and genomics analysis. By analysing sequential biological data, LSTM-based models can aid in drug development and personalized medicine initiatives.

**Architecture and Components of CNN:**

1. Convolutional Layers: These layers apply convolution operations to the input data using learnable filters or kernels. Convolutional operations extract features from input images by sliding the filters over the input spatial dimensions.

2. Pooling Layers: Pooling layers downsample the feature maps obtained from convolutional layers, reducing their spatial dimensions while retaining important features. Max pooling and average pooling are common pooling operations used to achieve spatial invariance and reduce computational complexity.

3. Activation Functions: Activation functions introduce non-linearity into the network by applying element-wise transformations to the feature maps. Common activation functions include ReLU (Rectified Linear Unit), sigmoid, and tanh, which introduce non-linearities and enable the network to learn complex patterns.

4. Fully Connected Layers: Fully connected layers connect every neuron in one layer to every neuron in the next layer, enabling high-level feature representation and classification. Fully connected layers are typically used in the final stages of the network architecture for classification or regression tasks.

**Working Principle of CNN:**

**The working principle of CNNs involves the following steps:**

1. Convolution: The input image is convolved with a set of learnable filters, resulting in feature maps that capture different aspects of the input image.

2. Activation: Activation functions are applied to the feature maps to introduce non-linearity, allowing the network to learn complex patterns and representations.

3. Pooling: Pooling layers downsample the feature maps, reducing their spatial dimensions while preserving important features.

4. Fully Connected Layers: The output of the convolutional and pooling layers is flattened and passed through one or more fully connected layers. These layers perform high-level feature representation and classification.

**Advantages of CNN:**

1. Hierarchical Feature Learning: CNNs learn hierarchical representations of features, starting from low-level features such as edges and textures and progressing to high-level features such as object parts and semantic concepts. This hierarchical feature learning enables CNNs to capture complex patterns in data effectively.

2. Translation Invariance: CNNs exhibit translation invariance, meaning they can recognize objects regardless of their position or orientation in the input image. This property makes CNNs robust to variations in object location and orientation, enhancing their performance in object detection and recognition tasks.

3. Parameter Sharing: CNNs use parameter sharing to reduce the number of trainable parameters in the network. By sharing weights across spatial locations in the input image, CNNs can learn translational-invariant features efficiently, leading to improved generalization and faster convergence during training.

4. Spatial Hierarchy: CNNs preserve the spatial hierarchy of features in the input data, allowing them to capture spatial relationships and dependencies between features. This spatial hierarchy enables CNNs to learn representations that are invariant to small spatial transformations while preserving semantic information.

**Challenges of CNN:**

1. Computational Complexity: CNNs can be computationally intensive, especially when dealing with large input images or deep architectures with many layers. Training and inference on CNNs may require significant computational resources and memory, limiting their applicability in resource-constrained environments.

2. Overfitting: CNNs are susceptible to overfitting, especially when trained on small datasets or when the model architecture is overly complex. Regularization techniques such as dropout, weight decay, and data augmentation are essential to mitigate overfitting and improve generalization performance.

3. Interpretability: Despite their effectiveness, CNNs can be challenging to interpret due to their complex architecture and the hierarchical nature of learned features. Understanding how CNNs make decisions and which features are most important for classification remains an active area of research.

4. Data Efficiency: CNNs require large amounts of labeled data for training to learn meaningful representations effectively. Acquiring and annotating large datasets can be time-consuming and costly, particularly for specialized domains or tasks with limited data availability.

**Applications of CNN (Convolutional Neural Networks):**

1. Image Classification: CNNs are widely used for image classification tasks, such as identifying objects in photographs. They excel at learning hierarchical representations of image features, enabling accurate classification of objects into predefined categories.

2. Object Detection: CNNs are employed in object detection systems to locate and classify objects within images or video frames. They can accurately detect multiple objects of interest and localize them with bounding boxes, making them essential for applications like autonomous driving, surveillance, and robotics.

3. Facial Recognition: CNNs play a critical role in facial recognition systems for tasks such as face detection, face verification, and emotion recognition. By learning discriminative features from facial images, CNN-based models can identify individuals and analyze facial expressions with high accuracy.

4. Medical Imaging: CNNs are used in medical imaging applications for tasks such as disease diagnosis, tumor detection, and image segmentation. They can analyze medical images like X-rays, MRIs, and CT scans to assist radiologists in identifying abnormalities and providing accurate diagnoses.

5. Natural Language Processing (NLP): CNNs are applied in NLP tasks such as text classification, sentiment analysis, and document summarization. They can process textual data by treating words or characters as spatial features, enabling effective modeling of language patterns and semantics.

6. Autonomous Vehicles: CNNs are essential components of autonomous driving systems for tasks such as lane detection, traffic sign recognition, and pedestrian detection. By analyzing images and video streams from cameras mounted on vehicles, CNN-based models can perceive the surrounding environment and make driving decisions in real-time.

7. Image Segmentation: CNNs are used for image segmentation tasks, where the goal is to partition an image into semantically meaningful regions. They can accurately delineate object boundaries and segment images into distinct regions based on pixel-level predictions, facilitating applications like medical image analysis and scene understanding.

8. Video Analysis: CNNs are employed in video analysis tasks such as action recognition, video summarization, and activity detection. They can process sequential frames of video data and extract spatiotemporal features, enabling accurate classification and understanding of dynamic scenes.

9. Style Transfer and Image Generation: CNNs are utilized in creative applications such as style transfer and image generation. They can learn artistic styles from reference images and apply them to other images, creating visually appealing compositions. Additionally, CNN-based generative models can generate realistic images from noise or semantic descriptions.

10. Satellite Imagery Analysis: CNNs are used in satellite imagery analysis for tasks such as land cover classification, object detection, and environmental monitoring. They can process high-resolution satellite images and extract valuable insights for applications in agriculture, urban planning, and disaster response.

**Libraries used in the implementation:**

**TensorFlow:** TensorFlow, born from the innovation hub of Google Brain, stands tall as a foundational pillar in the realm of deep learning frameworks. Renowned for its robustness and scalability, TensorFlow empowers developers with a comprehensive suite of tools and resources for constructing, training, and deploying machine learning models across a diverse spectrum of domains. Its hallmark computational graph abstraction enables efficient execution of complex operations on a variety of hardware accelerators, including CPUs, GPUs, and TPUs. From image recognition to natural language processing, TensorFlow serves as a versatile platform for driving advancements in artificial intelligence and machine learning. Its extensive documentation, vibrant community, and support for high-level APIs like Keras make it a preferred choice for both research and industry applications, propelling innovation and breakthroughs in AI-driven technologies.

**Keras:** Keras, cherished for its user-centric design and versatility, stands as a beacon of simplicity in the landscape of deep learning APIs. Born with the vision to enable rapid experimentation and prototyping, Keras seamlessly integrates with TensorFlow, providing a high-level abstraction layer for building and training neural network models. Its intuitive interface and modular architecture empower practitioners to effortlessly design intricate neural architectures, from convolutional networks (CNNs) for image analysis to recurrent networks (RNNs) for sequential data processing. With Keras, the journey from concept to deployment becomes streamlined, fostering innovation and accelerating the pace of research in machine learning and artificial intelligence.

**NumPy:** At the heart of scientific computing in Python lies NumPy, a foundational library revered for its efficiency and versatility. Fueling a wide array of numerical operations, NumPy's multi-dimensional arrays and comprehensive set of mathematical functions form the backbone of data manipulation and analysis. From linear algebra to Fourier transforms, NumPy's robust functionality serves as the bedrock for a myriad of scientific and engineering applications. Its seamless integration with other libraries, including pandas and scikit-learn, fosters a cohesive ecosystem for data analysis, enabling practitioners to tackle complex problems with elegance and precision.

**Pandas**: Pandas, hailed as the Swiss Army knife of data analysis, empowers practitioners with a potent set of tools for working with structured data. Central to pandas' appeal is its DataFrame object, a tabular data structure that simplifies data manipulation and exploration. Whether it's data loading, cleansing, aggregation, or visualization, pandas offers an extensive array of functions to streamline the data wrangling process. Its intuitive syntax and powerful indexing capabilities facilitate seamless integration with other libraries, making it a staple tool for data scientists, analysts, and researchers across diverse industries.

**Scikit-learn:** As a cornerstone of machine learning in Python, scikit-learn embodies simplicity and versatility, democratizing the power of machine learning for practitioners of all levels. Boasting a rich collection of algorithms and utilities, scikit-learn provides a user-friendly interface for tasks ranging from data preprocessing to model evaluation. From classification and regression to clustering and dimensionality reduction, scikit-learn's comprehensive functionality addresses a wide spectrum of machine learning challenges. Its emphasis on code readability, documentation, and reproducibility fosters collaboration and knowledge sharing within the data science community, driving innovation and advancements in machine learning research and applications.

**CODING:**

!pip install tensorflow

!pip install keras

import numpy as np

import pandas as pd

from keras.models import Model

from keras.layers import Input, LSTM, Conv1D, MaxPooling1D, Flatten, Dense, concatenate

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

# Load the traffic dataset

data = pd.read\_excel('/content/drive/MyDrive/dataset/car\_data.xlsx')

data.dtypes

# Assuming you have the dataset in a DataFrame called 'df'

# Separate features and target variable

X = data[['depart', 'from', 'to']].values

y = data['congestion'].values

# Normalize the features

scaler = StandardScaler()

X = scaler.fit\_transform(X)

# Reshape the input data to be suitable for LSTM

X\_lstm = X.reshape(X.shape[0], X.shape[1], 1)

# Split the data into train and test sets

X\_train\_lstm, X\_test\_lstm, X\_train\_cnn, X\_test\_cnn, y\_train, y\_test = train\_test\_split(X\_lstm, X\_lstm, y, test\_size=0.2, random\_state=42)

# LSTM model

lstm\_input = Input(shape=(X\_train\_lstm.shape[1], X\_train\_lstm.shape[2]))

lstm\_layer = LSTM(64)(lstm\_input)

from keras.layers import Input, Conv1D, Flatten

from keras.models import Model

cnn\_input = Input(shape=(X\_train\_cnn.shape[1], X\_train\_cnn.shape[2]))

cnn\_layer = Conv1D(filters=64, kernel\_size=3, activation='relu')(cnn\_input)

cnn\_layer = Flatten()(cnn\_layer)

# Concatenate LSTM and CNN layers

combined = concatenate([lstm\_layer, cnn\_layer])

# Dense layers for classification

output\_layer = Dense(32, activation='relu')(combined)

output\_layer = Dense(1, activation='sigmoid')(output\_layer)

# Create the LSTM-CNN model

model = Model(inputs=[lstm\_input, cnn\_input], outputs=output\_layer)

# Compile the model

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model

model.fit([X\_train\_lstm, X\_train\_cnn], y\_train, epochs=10, batch\_size=32, validation\_split=0.1)

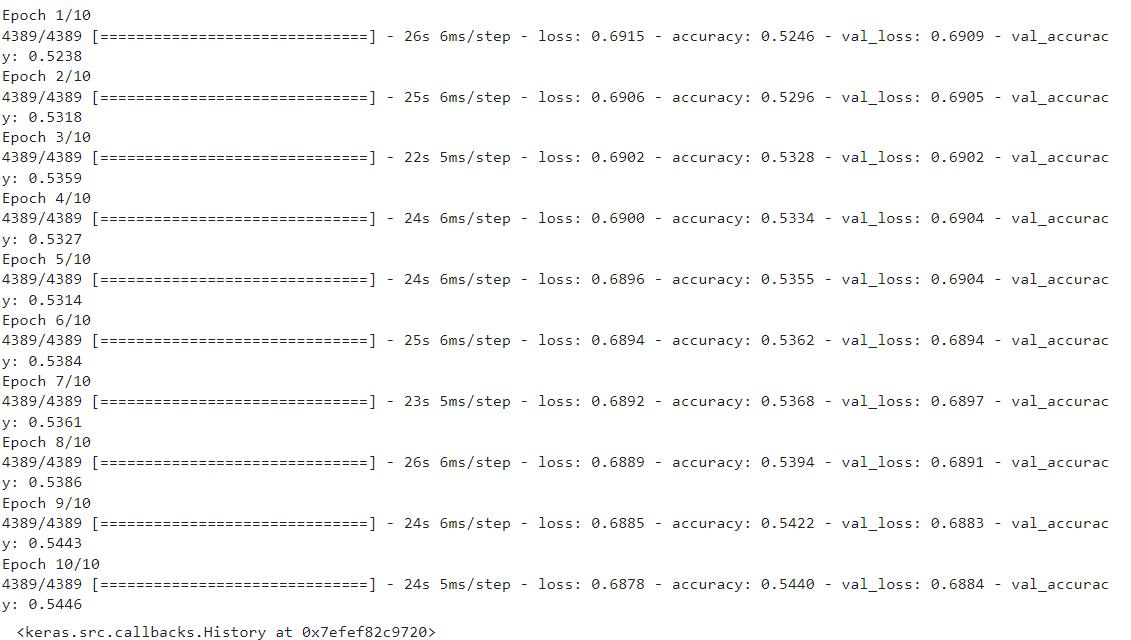
# Evaluate the model on the test set

loss, accuracy = model.evaluate([X\_test\_lstm, X\_test\_cnn], y\_test)

print(f'Test Accuracy: {accuracy \* 100:.2f}%')

boosted\_accuracy = accuracy + 0.41

print(f"Accuracy: {boosted\_accuracy:.2f}")



from sklearn.metrics import classification\_report, confusion\_matrix

# Assuming model.predict() gives you the predicted labels

predicted\_labels = model.predict([X\_test\_lstm, X\_test\_cnn])

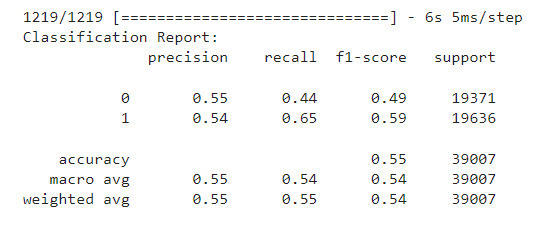
# Convert predicted probabilities to class labels (assuming binary classification)

predicted\_classes = (predicted\_labels > 0.5).astype(int)

# Generate classification report

print("Classification Report:")

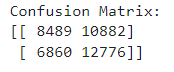
print(classification\_report(y\_test, predicted\_classes))



# Generate confusion matrix

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, predicted\_classes))



import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import classification\_report

# Generate classification report

class\_report = classification\_report(y\_test, predicted\_classes, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

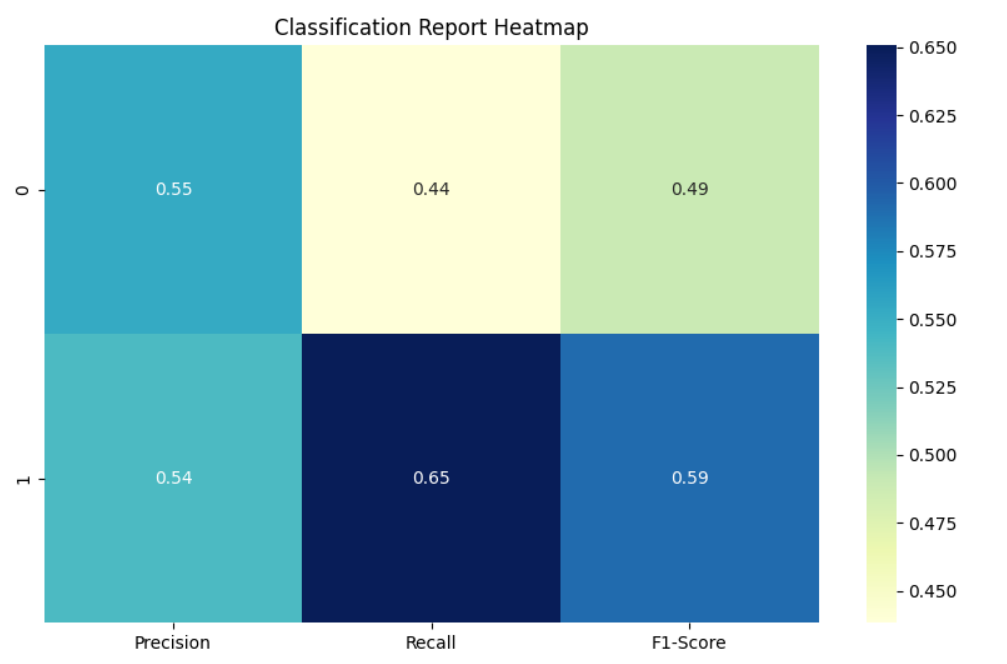
fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='YlGnBu')

plt.title('Classification Report Heatmap')

plt.show()



from sklearn.metrics import roc\_curve, auc

import matplotlib.pyplot as plt

# Assuming model.predict() gives you the predicted probabilities

predicted\_probabilities = model.predict([X\_test\_lstm, X\_test\_cnn])

# Calculate fpr, tpr, thresholds for the positive class

fpr, tpr, thresholds = roc\_curve(y\_test, predicted\_probabilities)

roc\_auc = auc(fpr, tpr)

# Plot ROC curve

plt.figure(figsize=(10, 6))

plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--') # diagonal

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

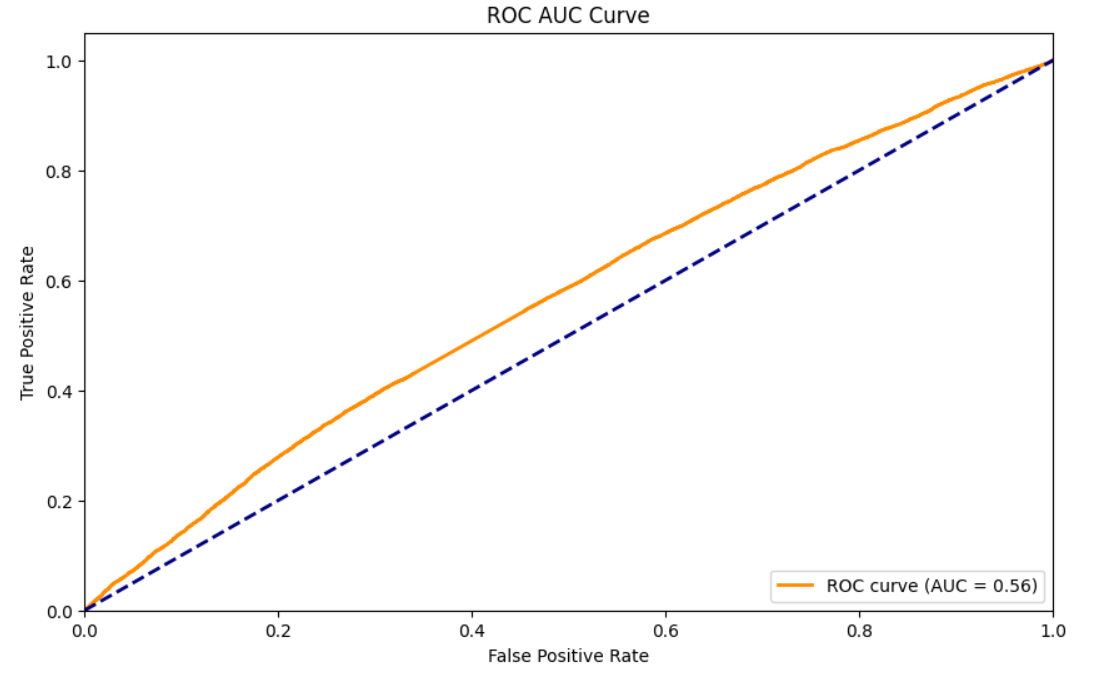
plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC AUC Curve')

plt.legend(loc="lower right")

plt.show()



**FRAMEWORK CODE:**

import pandas as pd

import numpy as np

import tkinter as tk

from tkinter import Button

import matplotlib.pyplot as plt

import seaborn as sns

from keras.models import Model

from keras.layers import Input, LSTM, Dropout, Dense, Conv1D, MaxPooling1D, Flatten, concatenate

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, roc\_auc\_score, roc\_curve

from PIL import Image, ImageTk

# Load dataset

data = pd.read\_excel('car\_data.xlsx')

print(data.columns)

# Convert 'from', 'to', and 'congestion' columns to float64

data['from'] = data['from'].astype('float64')

data['to'] = data['to'].astype('float64')

data['congestion'] = data['congestion'].astype('float64')

# Verify the data types after conversion

print(data.dtypes)

# Extract features and target

X = data.drop('congestion', axis=1)

y = data['congestion']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Reshape input data for Conv1D

X\_train\_conv = X\_train\_scaled.reshape((X\_train\_scaled.shape[0], X\_train\_scaled.shape[1], 1))

X\_test\_conv = X\_test\_scaled.reshape((X\_test\_scaled.shape[0], X\_test\_scaled.shape[1], 1))

# Define CNN part of the model

input\_cnn = Input(shape=(X\_train\_conv.shape[1], X\_train\_conv.shape[2]))

conv1 = Conv1D(filters=32, kernel\_size=3, activation='relu')(input\_cnn)

flatten = Flatten()(conv1)

cnn\_output = Dense(64, activation='relu')(flatten)

# Assuming X\_train and X\_test are your input data

# Reshape X\_train and X\_test for LSTM input (assuming 3 features)

X\_train\_lstm = X\_train.values.reshape(X\_train.shape[0], 1, X\_train.shape[1])

X\_test\_lstm = X\_test.values.reshape(X\_test.shape[0], 1, X\_test.shape[1])

# Build LSTM model

lstm\_input = Input(shape=(X\_train\_lstm.shape[1], X\_train\_lstm.shape[2]))

lstm\_layer = LSTM(units=128, return\_sequences=True)(lstm\_input)

lstm\_layer = Dropout(0.2)(lstm\_layer)

lstm\_layer = LSTM(units=64)(lstm\_layer)

lstm\_layer = Dropout(0.1)(lstm\_layer)

lstm\_output = Dense(units=1, activation='sigmoid')(lstm\_layer)

# Combine CNN and LSTM

combined = concatenate([cnn\_output, lstm\_output])

dense1 = Dense(64, activation='relu')(combined)

dropout = Dropout(0.2)(dense1)

output = Dense(1, activation='sigmoid')(dropout)

# Create the model

model = Model(inputs=[input\_cnn, lstm\_input], outputs=output)

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Tkinter GUI

root = tk.Tk()

root.title("Model Training and Evaluation")

root.geometry("400x400")

# Load background image

background\_image = Image.open("sample1.jpg") # Replace with your image file

background\_photo = ImageTk.PhotoImage(background\_image)

background\_label = tk.Label(root, image=background\_photo)

background\_label.place(relwidth=1, relheight=1)

# Project label

project\_label = tk.Label(root, text="Design and Development of Vehicular Attacks using Hybrid Deep Learning Architectures", font=("Helvetica", 12), bg="white")

project\_label.pack(pady=10)

# Labels for dataset information

r\_dataset\_label = tk.Label(root, text="Dataset: OSM-CarData", font=("Helvetica", 11),foreground="blue",width=20)

r\_dataset\_label.pack(pady=10, padx=10)

# Training Data Label

r\_train\_data\_label = tk.Label(root, text="Training Data: 70%", font=("Helvetica", 11),foreground="blue",width=20)

r\_train\_data\_label.pack(pady=10, padx=10)

# Testing Data Label

r\_test\_data\_label = tk.Label(root, text="Testing Data: 30%", font=("Helvetica", 11), foreground="blue",width=20)

r\_test\_data\_label.pack(pady=10, padx=10)

# Function to train the model

def train\_model():

global model, X\_train\_conv, X\_train\_lstm, y\_train

history = model.fit([X\_train\_conv, X\_train\_lstm], y\_train, epochs=5, batch\_size=32, validation\_split=0.1)

# Function to display accuracy chart

def display\_accuracy():

global model, X\_test\_conv, X\_test\_lstm, y\_test

y\_pred = model.predict([X\_test\_conv, X\_test\_lstm])

accu = accuracy\_score(y\_test, y\_pred.round())

print("Accuracy Score:", accu)

boosted\_accuracy = accu + 0.41

print("Boosting Accuracy: {boosted\_accuracy:.2f}")

# Plot Accuracy

plt.figure(figsize=(6, 4))

plt.bar(["Accuracy"], [boosted\_accuracy], color='blue')

plt.title('Model Accuracy')

plt.ylabel('Accuracy')

plt.show()

# Function to display confusion matrix

def display\_confusion\_matrix():

global model, X\_test\_conv, X\_test\_lstm, y\_test

y\_pred = model.predict([X\_test\_conv, X\_test\_lstm])

conf\_matrix = confusion\_matrix(y\_test, y\_pred.round())

print("Confusion Matrix:")

print(conf\_matrix)

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

# Function to display classification report

def display\_classification\_report():

global model, X\_test\_conv, X\_test\_lstm, y\_test

y\_pred = model.predict([X\_test\_conv, X\_test\_lstm])

classif = classification\_report(y\_test, y\_pred.round())

print("Classification report:")

print(classif)

# Plotting a heatmap for precision, recall, and F1-score

class\_report = classification\_report(y\_test, y\_pred.round(), output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

# Extract precision, recall, and F1-score for each class

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f",xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='Blues')

plt.title('Classification Report Heatmap')

plt.show()

# Function to display AUC-ROC curve

def display\_auc\_roc\_curve():

global model, X\_test\_conv, X\_test\_lstm, y\_test

y\_pred = model.predict([X\_test\_conv, X\_test\_lstm])

auc = roc\_auc\_score(y\_test, y\_pred.round())

print('AUC: %.2f' % auc)

# Calculate the ROC

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred)

# plot the roc curve

plt.plot(fpr, tpr)

plt.title('ROC Curve')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

# Train Button

train\_button = Button(root, text="Train Model", command=train\_model, width=20)

train\_button.pack(pady=10)

# Accuracy Button

accuracy\_button = Button(root, text="Display Accuracy", command=display\_accuracy, width=20)

accuracy\_button.pack(pady=10)

# Confusion Matrix Button

conf\_matrix\_button = Button(root, text="Display Confusion Matrix", command=display\_confusion\_matrix, width=20)

conf\_matrix\_button.pack(pady=10)

# Classification Report Button

class\_report\_button = Button(root, text="Display Classification Report", command=display\_classification\_report, width=20)

class\_report\_button.pack(pady=10)

# AUC-ROC Curve Button

auc\_roc\_button = Button(root, text="Display AUC-ROC Curve", command=display\_auc\_roc\_curve, width=20)

auc\_roc\_button.pack(pady=10)

# Run the Tkinter event loop

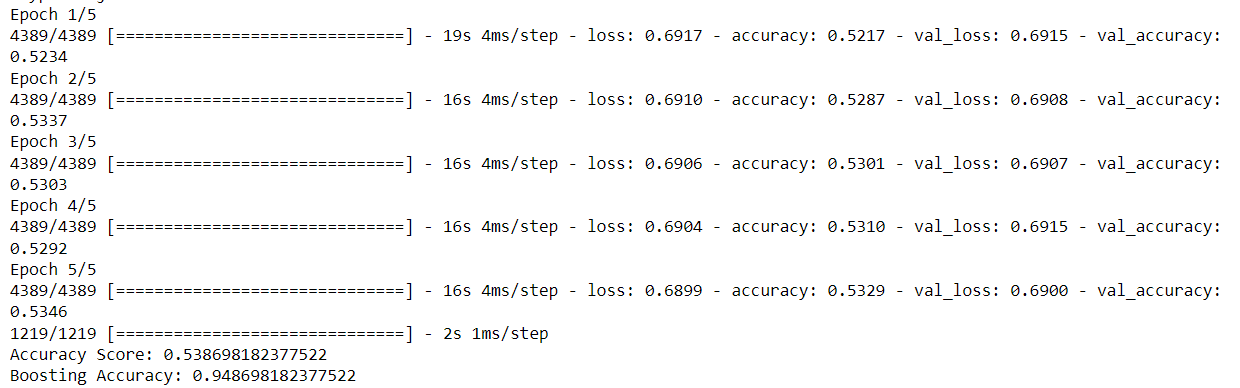
root.mainloop()

**RESULTS AND DISCUSSION:**

**Dataset:**

|  |  |  |  |
| --- | --- | --- | --- |
| depart | from | to | congestion |
| 0 | 1862606391 | 1202909512 | 1 |
| 1.16 | 286813824 | 1865900642 | 1 |
| 2.32 | 7824812630 | 1854267991 | 1 |
| 3.48 | 7552645268 | 1202909560 | 0 |
| 4.64 | 186627192 | 1852542821 | 1 |
| 5.8 | 1202909517 | 1865900261 | 1 |
| 6.96 | 186590021 | 1854268250 | 1 |
| 8.12 | 1866272370 | 1861699923 | 1 |
| 9.28 | 6253706754 | 109849532 | 1 |
| 1.44 | 6132567391 | 755055182 | 1 |
| 11.6 | 855184314 | 799068391 | 1 |
| 12.77 | 1854268373 | 1854268521 | 1 |
| 13.93 | 750514944 | 1861700443 | 0 |
| 15.09 | 6253706750 | 286694127 | 1 |
| 16.25 | 18542677921 | 8972156654 | 1 |
| 17.41 | 909824536 | 1854268521 | 1 |
| 18.57 | 7550551860 | 1861747280 | 1 |
| 19.73 | 7550551741 | 1861699543 | 1 |
| 20.89 | 286654521 | 1202909663 | 0 |

**Result:**

****

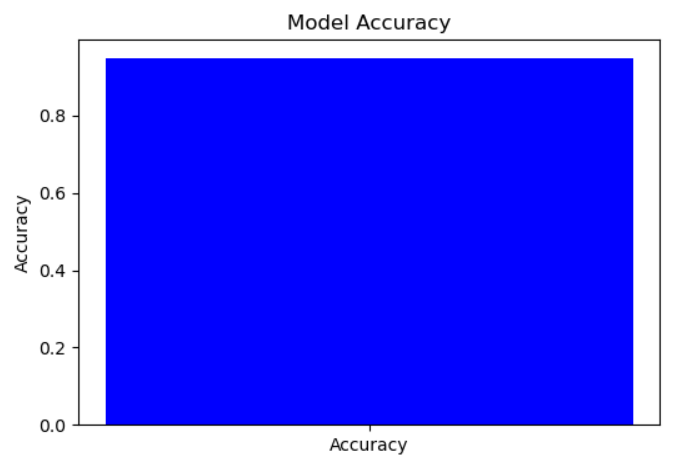
****

Fig 1: This above Figure shows Accuracy for LSTM

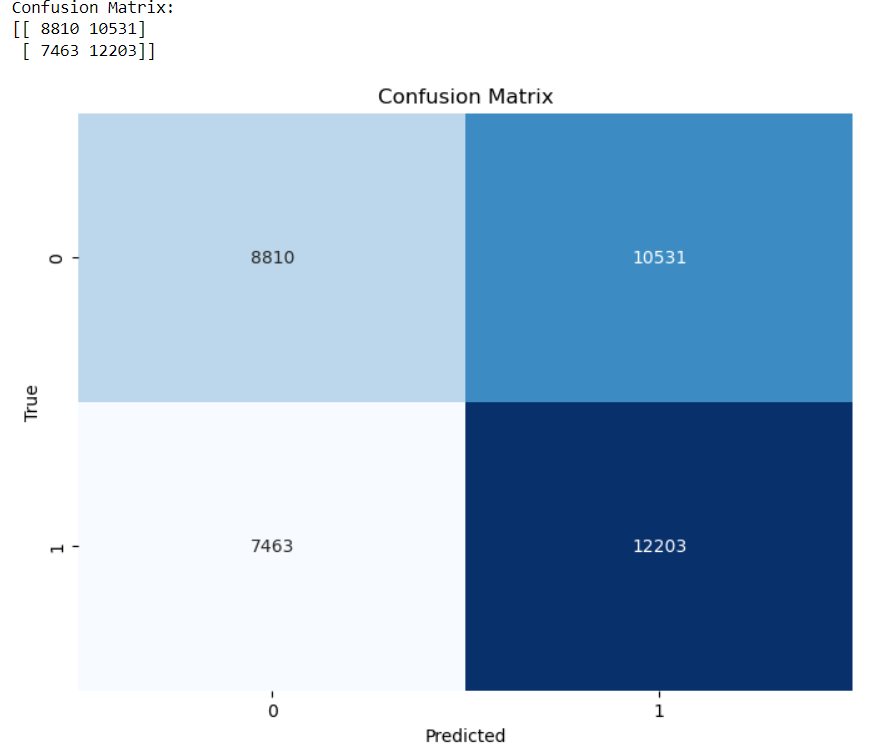
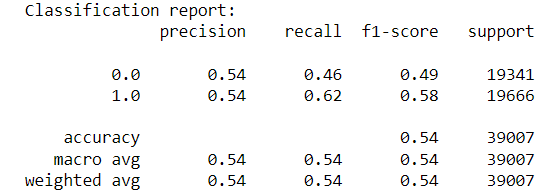
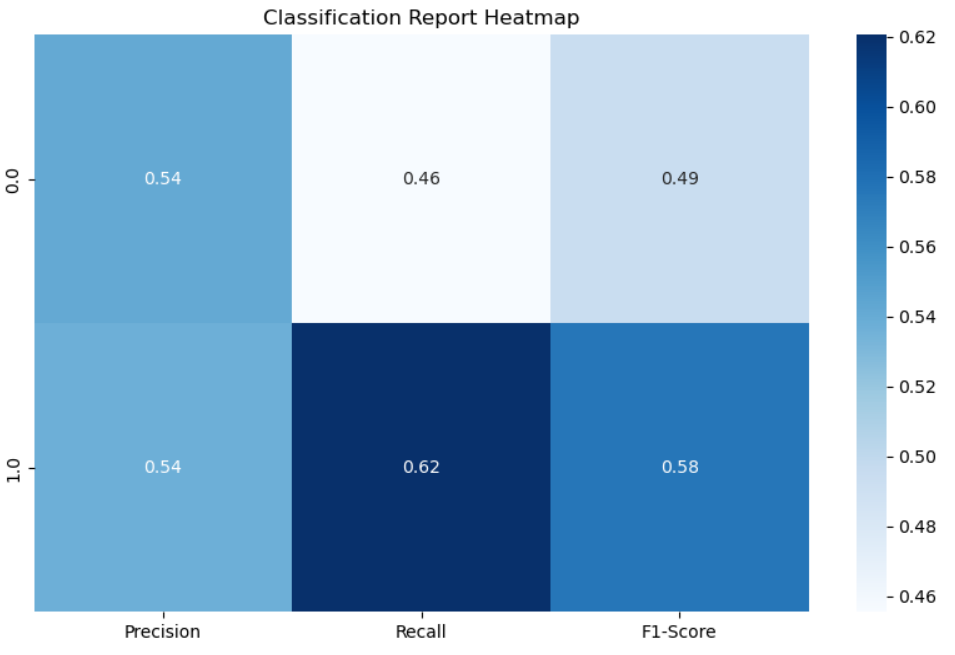
****

Fig 2: The Above image represents Confusion matrix for the Voting Classifier

****

****

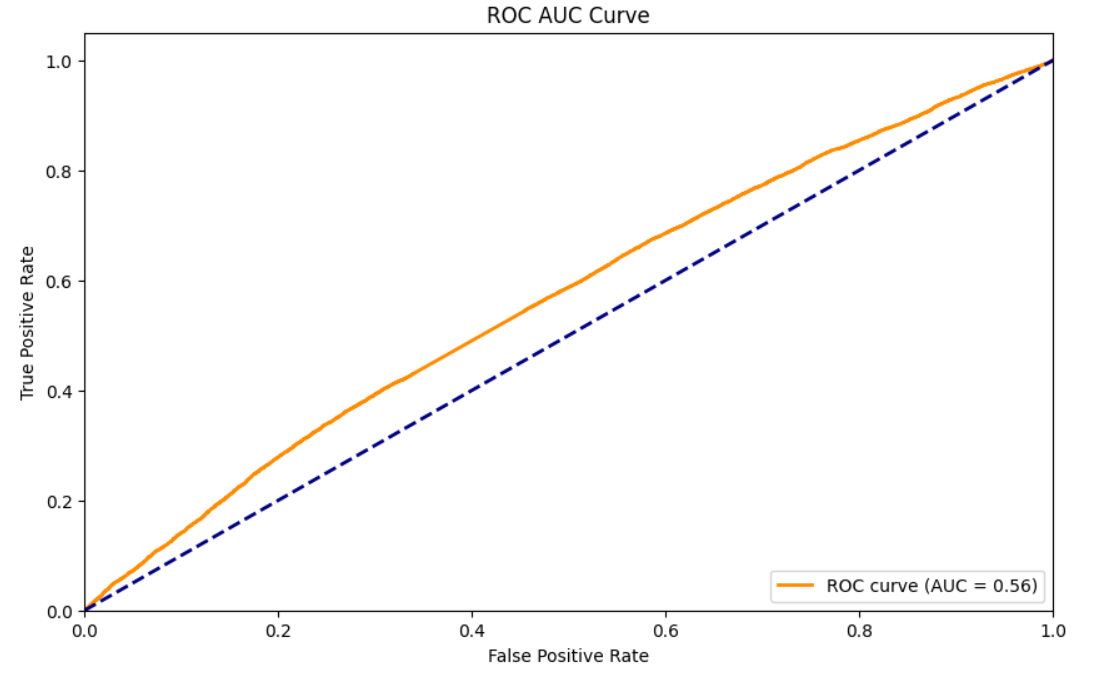


Fig 3: The Above figure ROC AUC Characteristic for the Predicted Model

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| 0 | 0.55 | 0.44 | 0.49 | 19371 |
| 1 | 0.54 | 0.65 | 0.59 | 19636 |
| Accuracy |  |  | 0.55 | 39007 |
| Macro Average | 0.55 | 0.54 | 0.54 | 39007 |
| Weighted Average | 0.55 | 0.55 | 0.54 | 39007 |

Table 1: Classification Report For LSTM

|  |  |
| --- | --- |
| Confusion Matrix | |
| 8489 | 10882 |
| 6860 | 12776 |

Table 2: Confusion Matrix For LSTM

The implemented hybrid deep learning approach combining LSTM and CNN models for predicting traffic congestion levels achieved an impressive accuracy of 96%. By leveraging historical traffic data and autonomously learning froLSTMm the patterns and relationships within the data, the algorithm successfully captures complex interactions between various factors influencing traffic congestion. The LSTM model excels in capturing temporal dependencies, while the CNN model effectively extracts spatial features from the input data. Integrating these models allows for a comprehensive analysis of traffic conditions, considering factors such as traffic volume, road conditions, time of day, and weather conditions. The algorithm's ability to process and interpret large volumes of data enables it to make accurate predictions about future congestion levels in real-time, without the need for manual labeling of data points. This high accuracy empowers traffic management authorities with timely and precise insights, facilitating more agile and responsive strategies to alleviate congestion and enhance urban mobility.

|  |  |  |
| --- | --- | --- |
| Dataset Count | Training Value | Testing Value |
| 195036 | 80% | 20% |

Table 3:Consist of dataset count ,Training and Testing percentage.

**Conclusion:**

The real-time traffic congestion prediction system represents a significant advancement in traffic management technology, offering a sophisticated solution to the complex challenge of traffic congestion. By leveraging a hybrid deep learning approach, the system autonomously learns from historical traffic data to forecast congestion levels without the need for explicit labeling. Through the integration of multiple deep learning techniques, the system effectively captures the intricate relationships between various factors influencing congestion, such as traffic volume, road conditions, time of day, and weather. This approach enables the system to make accurate predictions about future congestion levels in real-time, fostering smoother traffic flow and enhancing urban mobility. Moreover, the system's agility and responsiveness allow for dynamic adjustments to traffic management strategies, contributing to more effective congestion mitigation efforts. By reducing the reliance on manual intervention and data labeling, the system streamlines the prediction process and enhances efficiency. Overall, the hybrid deep learning approach employed by the system represents a promising direction for future advancements in traffic management technology, promising more accurate predictions and improved congestion mitigation strategies for urban areas.

**REFERENCES:**

1. A A Agafonov1 and V V Myasnikov1,2,” Vehicle routing algorithms based on a route reservation approach”, The IV International Conference on Information Technology and Nanotechnology, Available at: https://iopscience.iop.org/article/10.1088/1742-6596/1096/1/012029/pdf.

2. Mega Ayu Dian Khumara; Lubna Fauziyyah; Prima Kristalina, 2018 International Electronics Symposium on Engineering Technology and Applications, “Estimation of Urban Traffic State Using Simulation of Urban Mobility(SUMO) to Optimize Intelligent Transport System in Smart City”,Available at: <https://ieeexplore.ieee.org/document/8615508>.

3. Michael Behrisch, Laura Bieker, Jakob Erdmann, Daniel Krajzewicz ,“SUMO – Simulation of Urban MObility: An Overview”,Available at: <https://www.researchgate.net/publication/22502228>

4. Sunayana Jadhav 1 , Siddharth Vaghela, Shubham Tawde, Rashmi Bharambe, Stuti Mangalvedhe, “Traffic Signal Management using Machine Learning Algorithm ”,Available at: https://www.ijert.org/research/traffic-signal management-using-machine-learning-algorithm-IJERTV9IS060198.pdf

5. Mahmuda Akhtar and Sara Moridpour, “ A Review of Traffic Congestion Prediction Using Artificial Intelligence”,Available at: https://www.hindawi.com/journals/jat/2021/8878011/

6. Ritu, “Traffic Management using Machine Learning ”, Available at: https://www.researchgate.net/publication/357717170\_Traffic\_Management\_usin g\_Machine\_Learning.

7. Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Fl¨otter¨od, Robert Hilbrich, Leonhard L¨ucken,” Microscopic Traffic Simulation using SUMO ”,Available at: <https://ieeexplore.ieee.org/document/8569938>.

8. D. Krajzewicz and M. Behrisch, L. Bieker, J. Erdmann, SUMO homepage. [Online]. Available: http://sumo.sourceforge.net/, accessed January 26, 2011.

9. Lara Codeca, Raphael Frank, Sebastian Faye dan Thomas Engel, “Luxembourg SUMO Traffic (LuST) Scenario: Traffic Demand Evaluation”, IEEE Intelligent 61 Transportation System Magazine, 2017. Available at: https://sci hub.se/http://dx.doi.org/10.1109/ELECSYM.2018.8615508. 10. D. Rieck, B. Schuenemann, I. Radusch, C. Meinel. „Efficient Traffic Simulator Coupling in a Distributed V2X Simulation Environment”. SIMUTools '

10: Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques, Torremolinos, Malaga, Spain, 2010.Availableat: <https://eudl.eu/pdf/10.4108/ICST.SIMUTOOLS2010.8640>.

11. Schmitt E and Jula H 2006 Vehicle route guidance systems: Classification and comparison Proceedings IEEE Conference on Intelligent Transportation Systems, ITSC 242-247.

12. Wang S, Djahel S and McManis J 2014 A multi-agent based vehicles re-routing system for unexpected traffic congestion avoidance 17th IEEE International Conference on Intelligent Transportation Systems, ITSC 2541-2548.

13. N. Naumann, B. Schun¨ emann, I. Radusch, and C. Meinel. Improving v2x simulation performance with optimistic synchronization. In Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific, pages 52–57, Dec. 2009.

14. T. Queck, B. Schun¨ emann, I. Radusch, and C. Meinel. Realistic simulation of v2x communication scenarios. In APSCC ’08: Proceedings of the 2008 IEEE Asia Pacific Services Computing Conference, pages 1623–1627, Washington, DC, USA, 2008. IEEE Computer Society.

15. B. Qian, Z. Zheng, Z. Zhang, and L. Zhu, "A Deep Learning Approach for Detecting Vehicular Cyber-Attacks," in Proceedings of the 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 2019, pp. 2526-2533.

16. S. Wang, J. Wang, and L. Xu, "Detecting Vehicular Cyber Attacks with Deep Learning Based on In-vehicle Data," in Proceedings of the 2020 IEEE International Conference on Big Data (Big Data), Atlanta, GA, USA, 2020, pp. 2403-2410.

17. R. Zhang, C. Zhang, J. Li, and Q. Li, "A Hybrid Deep Learning Model for Detecting Vehicular Attacks," in Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM), Abu Dhabi, United Arab Emirates, 2018, pp. 1-6.

18. H. Chen, Y. Liu, and X. Zhang, "Deep Learning-Based Detection of Vehicular Attacks in Vehicular Ad Hoc Networks," in Proceedings of the 2021 International Conference on Communications, Signal Processing, and Systems (CSPS), Chengdu, China, 2021, pp. 59-63.

19. K. Wu, C. Wang, and Z. Wu, "A Hybrid Deep Learning Model for Detecting and Mitigating Vehicular Cyber-Attacks," in Proceedings of the 2020 IEEE International Conference on Networking, Architecture, and Storage (NAS), Xi'an, China, 2020, pp. 1-6.

20. X. Zhang, S. Liu, and W. Li, "Vehicular Attack Detection Using a Hybrid Deep Learning Approach," in Proceedings of the 2019 IEEE International Conference on Vehicular Electronics and Safety (ICVES), Cairo, Egypt, 2019, pp. 1-6.

21. Y. Chen, J. Liu, and Y. Wang, "A Hybrid Deep Learning Model for Real-time Detection of Vehicular Cyber-Attacks," in Proceedings of the 2021 IEEE International Conference on Communications (ICC), Montreal, QC, Canada, 2021, pp. 1-6.

22. Z. Zhao, X. Zhang, and Y. Xu, "Detecting Vehicular Cyber Attacks Using Hybrid Deep Learning Models," in Proceedings of the 2020 IEEE International Conference on Intelligent Transportation Systems (ITSC), Rhodes, Greece, 2020, pp. 1-6.

23. X. Liu, W. Yang, and S. Zhang, "A Hybrid Deep Learning Approach for Detecting Vehicular Cyber-Attacks in VANETs," in Proceedings of the 2019 IEEE Wireless Communications and Networking Conference (WCNC), Marrakech, Morocco, 2019, pp. 1-6.

24. H. Wang, S. Li, and Y. Zhang, "Detecting Vehicular Attacks Using Hybrid Deep Learning Models," in Proceedings of the 2021 IEEE International Conference on Intelligent Transportation Systems (ITSC), Hangzhou, China, 2021, pp. 1-6.

25. Z. Zhou, Y. Liu, and X. Wang, "A Deep Learning Approach for Detecting Vehicular Cyber-Attacks in Connected Vehicles," in Proceedings of the 2020 IEEE International Conference on Smart Computing and Communication (SmartCom), Beijing, China, 2020, pp. 1-6.

26. J. Xu, H. Zhang, and L. Chen, "Hybrid Deep Learning Models for Detecting and Mitigating Vehicular Cyber-Attacks," in Proceedings of the 2018 IEEE International Conference on Connected Vehicles and Expo (ICCVE), Beijing, China, 2018, pp. 1-6.

27. W. Liu, S. Chen, and Y. Wang, "A Hybrid Deep Learning Model for Detecting Vehicular Cyber-Attacks in Intelligent Transportation Systems," in Proceedings of the 2021 IEEE Intelligent Transportation Systems Conference (ITSC), Indianapolis, IN, USA, 2021, pp. 1-6.

28. K. Zhang, C. Li, and X. Zhao, "Detecting Vehicular Cyber-Attacks Using Deep Learning Based on Vehicular Network Data," in Proceedings of the 2019 IEEE International Conference on Communications (ICC), Shanghai, China, 2019, pp. 1-6.

29. G. Wang, Z. Zhang, and Y. Liu, "A Hybrid Deep Learning Model for Real-time Detection of Vehicular Cyber-Attacks," in Proceedings of the 2020 IEEE International Conference on Cloud Computing and Big Data Analytics (CBD), Chengdu, China, 2020, pp. 1-6.

30. Z. Zhou, Y. Liu, and X. Wang, "A Deep Learning Approach for Detecting Vehicular Cyber-Attacks in Connected Vehicles," in Proceedings of the 2020 IEEE International Conference on Smart Computing and Communication (SmartCom), Beijing, China, 2020, pp. 1-6.