IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 21 March 2024, accepted 20 April 2024, date of publication 25 April 2024, date of current version 8 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3393909

RESEARCH ARTICLE

Deep Learning Model for Driver Behavior Detection in Cyber-Physical System-Based Intelligent Transport Systems

BRIJ B. GUPTA[1,2,3], **(Senior Member, IEEE),**
AKSHAT GAURAV[4], **(Graduate Student Member, IEEE),**
KWOK TAI CHUI[5], **(Member, IEEE), AND VARSHA ARYA**[6,7]

[1]Department of Computer Science and Information Engineering, Asia University, Taichung 413, Taiwan
[2]Symbiosis Centre for Information Technology (SCIT), Symbiosis International University, Pune 411004, India
[3]Center for Interdisciplinary Research, University of Petroleum and Energy Studies (UPES), Dehradun 248007, India
[4]Ronin Institute, Montclair, NJ 07043, USA
[5]Department of Electronic Engineering and Computer Science, Hong Kong Metropolitan University (HKMU), Hong Kong
[6]Department of Business Administration, Asia University, Taichung 413, Taiwan
[7]Department of Electrical and Computer Engineering, Lebanese American University, Beirut 1102, Lebanon

Corresponding author: Brij B. Gupta (bbgupta.nitkkr@gmail.com)

The research described in this article is fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (UGC/FDS16/E06/22).

ABSTRACT As Intelligent Transport Systems (ITS) continue to evolve, the quest for improving road safety and transportation efficiency has gained renewed emphasis.One of the pivotal aspects in this endeavor is the detection and analysis of driver behavior. Recognizing signs of fatigue, distraction, or inattentiveness is critical in enhancing road safety and optimizing traffic flow. In this paper, we present a pioneering approach to driver behavior detection within the realm of ITS using deep learning models in the Cyber-Physical Systems (CPS) framework. Our research focuses on the discernment of critical behaviors such as eye closure, open-eye state, yawning, and non-yawning instances. With an unwavering commitment to road safety and transportation efficiency, we've harnessed the power of deep learning to design, develop, and train an exceptionally accurate model. Through rigorous evaluation, we achieved an impressive 94% accuracy. Our findings unveil the potential of CPS-based solutions for real-time driver behavior monitoring, providing a foundation for safer roadways and more streamlined traffic management. The proposed deep learning model offers robust and accurate predictions, enabling timely responses to various driving conditions. This research significantly advances the field of driver behavior analysis within the context of intelligent transportation systems, with broad implications for road safety and traffic management.

INDEX TERMS Driver behavior detection, deep learning, cyber-physical systems (CPS), intelligent transport systems (ITS), road safety, driver monitoring, behavioral analysis, artificial intelligence (AI).

I. INTRODUCTION

Intelligent Transport Systems (ITS) have acted as an efficient solution for improving the operational performance of traffic systems in smart cities, reducing traffic congestion, and increasing safety for travellers [1], [2], [3], [4], [5]. ITS has benefited from cutting-edge technologies such as artificial intelligence, the Internet of Things (IoT), and deep learning. These new-age technologies allow for collecting and analyzing large amounts of data, which can be used to develop intelligent algorithms and models for various transportation applications. Due to the inclusion of these technologies, ITS can provide various services to end users, including road safety and accident prediction, traffic prediction and control, and vehicle control and monitoring [6], [7], [8], [9].

The main components of ITS include advanced sensors, cameras, computers, electronics, and telecommunication technologies [10], [11], [12]. These components work together to collect and analyze data related to traffic conditions, vehicle movements, and driver behaviors. ITS

The associate editor coordinating the review of this manuscript and approving it for publication was Mouquan Shen.

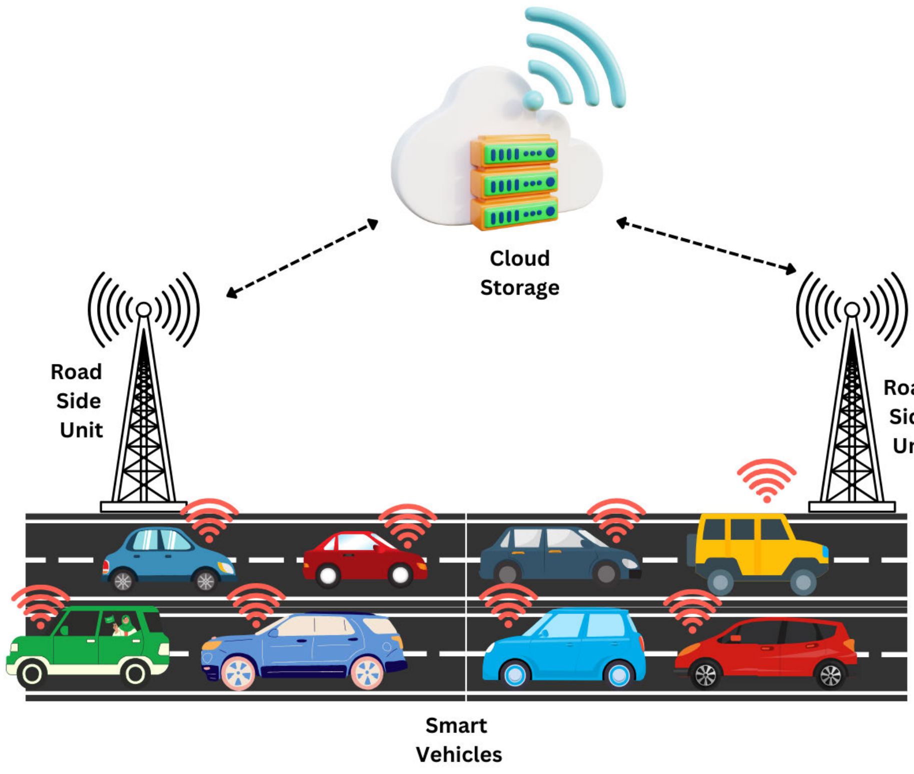


FIGURE 1. Smart transport system.

including Road Side Units (RSUs), vehicles, and servers, as represented in Figure 1. RSUs are deployed along roads and equipped with sensors to detect road events and communicate with vehicles equipped with Onboard Units (OBUs) [13], [14], [15]. These OBUs are powerful nodes with various sensors, communication modules, and computing capabilities, enabling the wireless exchange of traffic information between vehicles and roadside infrastructure, thereby enhancing road safety and transportation network efficiency [16], [17], [18]. The integration of RSUs and OBUs in ITS has attracted attention due to their potential to enhance road safety, improve traffic management, and alleviate road accidents [19], [20], [21].Furthermore, the deployment of RSUs and OBUs facilitates the monitoring and management of traffic flow, contributing to the development of advanced traffic management systems [22], [23], [24]. The collaboration between RSUs and OBUs also enables the implementation of intelligent upgrades in transportation infrastructure, optimizing communication environments and deployment methods [25], [26]. This leads to the development of new frameworks to analyze the behavior pattern of the driver.

More than 80% of road accidents happen due to abnormal driver behavior [27]. Many road safety reports regard human behavior as the most important factor in the likelihood of accidents. The detection and classification of aggressive or abnormal driver behavior is an essential requirement in the real world to avoid deadly road accidents and to protect road users [28]. In this context, ITS can effectively enhance road safety by accurately detecting and analyzing driver behavior. Due to this, researchers are constantly working on developing advanced machine learning models and deep learning techniques for driver behavior detection in ITS [29], [30]. However, there are many limitations to detecting driver behaviors using deep learning models, as the deep learning models need a vast amount of data, and their training time is also high. In addition to that, deep learning models also require significant computational resources for training and inference. Hence, a model that can effectively detect driver behavior in real-time while considering these limitations is needed. In this context, we proposed a deep learning model that efficiently detects driver behavior in real time. The contribution of our work is as follows: We conducted an in-depth review of existing machine learning methods and their application in driver behavior detection within intelligent transportation systems. Based on our review, we found that deep learning models have shown promising results in accurately detecting and analyzing driver behaviors. To improve the efficiency of driver behavior detection, we proposed a novel deep learning model that is based on Convolutional Neural Network (CNN). This model considers the limitations of deep learning models by optimizing its architecture for real-time detection and incorporating techniques such as data augmentation and transfer learning to improve the model's performance with limited training data. To get better understanding of the readers Table 1 presents tha abbreviations used in the paper.

TABLE 1. Abbreviation table.

Abbrevation	Meaning
ITS	Intelligent Transport System
CPS	Cyber-Physical Systems
AI	, Artificial Intelligence
IoT	Internet of Things
RSUs	Road Side Units
OBUs	Onboard Units
CNN	Convolutional Neural Network
IRL	Inverse reinforcement learning
RNN	Recurrent neural networks
FC	fully connected neural network
LSTM	Long short-term memory
AVI	Automatic vehicle identification
CPU	Central Processing Unit

The rest of the paper is organized as follows: Section II provides a comprehensive overview of related work in driver behavior detection. Section III details the proposed approach for driver behavior detection using deep learning. Section IV presents the results and discussion, and finally, section V concludes the paper.

II. RELATED WORK

Research suggests that deep learning models [31], [32] and machine learning [33], [34], such as inverse reinforcement learning (IRL) and recurrent neural networks (RNN), can be used to predict driver behavior. Wu et al. [35] proposed a joint IRL-DL framework to predict drivers' future behavior in ride-hailing platforms, achieving consistent and remarkable improvements over models without drivers' preference vectors. Wei et al. [36] proposed a hybrid neural network prediction model based on RNN and a fully connected neural network (FC) to accurately predict lane-changing behavior in real traffic scenarios. The proposed model achieved a prediction accuracy of 93.5% and improved the prospective time of prediction by about 2.1s on average. Liu et al. [37] proposed a new algorithm of driver behavior model based on the whale optimization algorithm-restricted Boltzmann machine (WOA-RBM) method, which achieved an accuracy of 90% in MATLAB simulation. Overall, these papers suggest that deep learning models can be effective in predicting driver behavior.

Jin et al. [38] proposes a DL framework driven by automatic vehicle identification (AVI) data to model drivers' behaviors and incorporate travel time prediction in the next location prediction problem. Arbabi et al. [39] embed the Intelligent Driver Model into deep neural networks to create a transparent and interpretable model for driver behavior prediction. Abdelrahman et al. [40] presents a robust data-driven framework for calculating drivers' risk profiles using supervised machine learning. Jiang et al. [41] uses a deep CNN and a long short-term memory (LSTM)–based model to detect driver inattention and predict upcoming abnormal operations on the road. These papers demonstrate the potential of deep learning models in predicting driver behavior and improving road safety.

III. PROPOSED MODEL

A. DATA PREPROCESSING

The first step of our proposed approach is data preprocessing. Algorithm 1 presents the steps of data preprocessing.

Algorithm 1 Data Transformation and DataLoader Setup

```
   Data: train_data, test_data, BATCH_SIZE
   Result: train_dataloader, test_dataloader
1  Function main:
       /* Data Transformation                              */
2      data_transforms ← Compose([]);
3      for transform in transforms do
4          if transform is Resize then
5              data_transforms.add(Resize(size=(64,64)));
6          if transform is RandomHorizontalFlip then
7              data_transforms.add(RandomHorizontalFlip
                 (p=0.5));
8          if transform is ToTensor then
9              data_transforms.add(ToTensor());
       /* Convert datasets into data
          loaders                                          */
10     train_dataloader ←
           DataLoader(dataset=train_data,
           batch_size=BATCH_SIZE,
           num_workers=os.cpu_count(), /* Count
           the number of CPUs                              */
11     shuffle=True) /* Training DataLoader
           */
12     test_dataloader ←
           DataLoader(dataset=test_data,
           batch_size=BATCH_SIZE,
           num_workers=os.cpu_count(), /* Count
           the number of CPUs                              */
13     shuffle=False) /* Testing DataLoader
           */
14     return train_dataloader, test_dataloader
```

- Data Transformation: The first essential step in our algorithm is data transformation. In this stage, we meticulously process the dataset to ensure its compatibility with our deep learning model. This entails a series of critical operations to enhance the dataset's quality and suitability. These transformations are defined and stored within the data_transforms set.
- For Loop Iteration: Subsequently, we employ a for loop to systematically iterate through a list of transformations. Each transformation in the list is carefully assessed and applied in sequence, ensuring that the data is subjected to each defined operation.

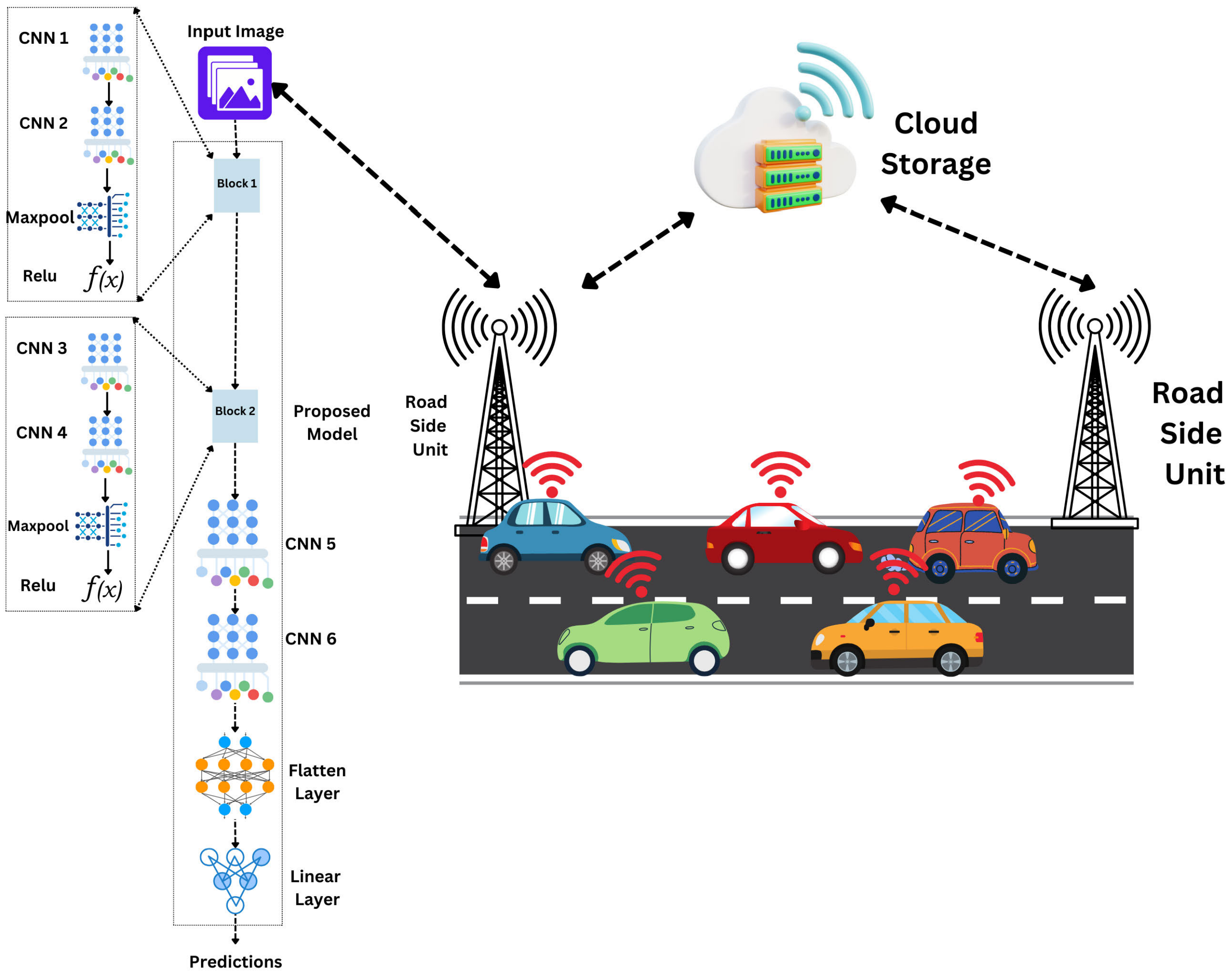


FIGURE 2. **Model architecture.**

- Resize Transformation: The first transformation involves resizing the images in the dataset to a standardized 64 $\times$ 64 resolution. This resizing operation is pivotal, as it guarantees that all images conform to the same dimensions, which is essential for consistent input to the deep learning model.
- Random Horizontal Flip: The next transformation assesses whether it is a "RandomHorizontalFlip" operation. If affirmative, it introduces a random horizontal flip to the images, with a 50% probability. This augmentation technique enhances dataset diversity and, in turn, contributes to the model's robustness.
- ToTensor Transformation: Lastly, if it is a "ToTensor" operation. If so, it converts the images into tensors. Deep learning models require data in a tensor format, making this step indispensable for preparing the data for neural network processing.
- Data Loader Setup - Training: With data transformation completed, the algorithm proceeds to establish data loaders, vital for efficiently supplying data to the deep learning model during training and testing. In the context of training, detailed in Lines 27-37, the training data loader is configured with various parameters.
- Training Data Loader Configuration: The configuration of the training data loader includes specifying the dataset (training data), defining the batch size, leveraging available central processing unit (CPU) cores for parallel data loading, and enabling shuffling. Shuffling the data during training, as described in Line 34, is a crucial practice to introduce randomness and prevent the model from memorizing the order of the data, thus mitigating the risk of overfitting.
- Data Loader Setup - Testing: Similar to the training data loader, a data loader is set up for testing.
- Testing Data Loader Configuration: The configuration of the testing data loader involves specifying the dataset (testing data), defining the batch size, utilizing CPU cores for data loading, and, crucially, disabling

shuffling. The absence of shuffling in the testing data loader ensures that the test data remains consistent during model evaluation.

B. MODEL ARCHITECTURE

The proposed model forms the cornerstone of our deep learning-based driver behavior prediction system, as represented in Figure 2 and algorithm 2. Below is a detailed overview of the architecture, depicted layer by layer, along with their respective dimensions and parameter counts:

Algorithm 2 Deep Learning Model Architecture

```
Data: input_shape, hidden_shape, output_shape
Result: TinyVGG Model Architecture
1  Function main:
       /* Initialize the model                          */
2      TinyVGG ← class TinyVGG(nn.Module) ;
3      TinyVGG.__init__(self, input_shape,
       hidden_shape, output_shape) ;
       /* Initialize Convolutional
          Blocks                                        */
4      conv_blocks ← [] ;
5      for layer in [1, 2, 3] do
6          conv_block ← nn.Sequential(...) ;
7          conv_blocks.add(conv_block) ;
       /* Initialize Classifier                        */
8      classifier ← nn.Sequential(...) ;
       /* Forward Pass                                  */
9      def forward(self, x) ;
10     for block in conv_blocks do
11         x ← block(x) ;
12     x ← classifier(x) ;
13     return x ;
```

- **Input Layer**: The input layer of the model receives images of size 64 × 64 pixels.
- **Sequential Block 1**:
  - Convolutional Layer 1: Applies a 2D convolution operation with 10 filters, a kernel size of 3 × 3, and ReLU activation.
  - Convolutional Layer 2: Another 2D convolution with 10 filters, a kernel size of 3 × 3, and ReLU activation.
  - MaxPooling Layer: Reduces the spatial dimensions by performing max-pooling with a pool size of 2 × 2.
- **Sequential Block 2**:
  - Convolutional Layer 3: Performs a 2D convolution with 10 filters, a kernel size of 3 × 3, and ReLU activation.
  - Convolutional Layer 4: Another 2D convolution with 10 filters, a kernel size of 3 × 3, and ReLU activation.
  - MaxPooling Layer: Further reduces spatial dimensions with a 2 × 2 pool size.
- **Sequential Block 3**:
  - Convolutional Layer 5: Carries out a 2D convolution with 10 filters, and ReLU activation.
  - Convolutional Layer 6: Another 2D convolution with 10 filters and ReLU activation.
- **Sequential Block 4**:
  - Flatten Layer: Converts the dimensionality of the input data
  - Linear Layer: A fully connected layer that maps the flattened features to output representing the classes.

C. MODEL TRANING STAGE

The algorithm 3 represents the model training process. During model training, we first store the data in different batches. After that, we set the optimizer and loss functions. Then, finally, we start training our model. The details of the algorithm are as follows:

Algorithm 3 Model Training Procedure

```
Data: model, dataLoader, lossFunction, optimizer,
      computingDevice
Result: Average Training Loss and Accuracy
1  Function Main:
2      model.train() ;
3      trainingLoss, trainingAccuracy ← 0, 0 ;
4      for batch in dataLoader do
5          inputs, targets ← PrepareData(batch,
            computingDevice) ;
6          predictions ← model(inputs) ;
7          loss ← lossFunction(predictions, targets) ;
8          trainingLoss + = loss.item() ;
9          optimizer.zero_grad() ;
10         loss.backward() ;
11         optimizer.step() ;
12         correctPredictions ←
            CalculateAccuracy(predictions, targets) ;
13         trainingAccuracy + = correctPredictions ;
14     trainingLoss ← trainingLoss /
        NumberOfBatches(dataLoader) ;
15     trainingAccuracy ← trainingAccuracy /
        NumberOfBatches(dataLoader) ;
16     return trainingLoss, trainingAccuracy ;
```

- Setting the Model for Training: The "train_step" begins by setting the deep learning model to training mode.

This step is essential as it notifies the model to adjust its behavior for training, including operations like dropout and batch normalization. The model is then initialized, and we start with zero values for the training loss and accuracy metrics (Line 4). These metrics will be updated as the model processes the data.

- Iterating Through Data Batches: The core of the "train_step" involves a loop that iterates through the data batches in the provided data loader. In each iteration (Line 7), we fetch a batch of data, including input features (X) and corresponding labels (y), and ensure these data points are on the target device, which is often a GPU for faster computation (Line 10).
- Forward Pass: Predictions: Next, the algorithm proceeds with the forward pass, where the input features (X) are passed through the deep learning model (Line 13). This generates predictions (y_pred) for the given data batch. These predictions are essential for calculating the loss.
- Calculating the Loss: The loss measures how far off our model's predictions are from the actual labels. This difference is quantified by a loss function (loss_fn). The computed loss is added to the ongoing training loss, allowing us to monitor how well the model is learning from the data.
- Optimizer and Backpropagation: After calculating the loss, the algorithm prepares for the backpropagation step by zeroing out the gradients of the model's parameters to avoid accumulating gradients from previous iterations. Then, it performs backpropagation to compute gradients with respect to the loss. These gradients guide the optimization process. The final step involves optimizing the model's parameters using the optimizer, effectively updating the model's weights to minimize the loss.
- Evaluating Accuracy: Additionally, the algorithm evaluates the model's accuracy for the current batch. It calculates the predicted class by taking the maximum probability from the softmax output and comparing it to the true labels. This process provides insights into how well the model is classifying the data.
- Adjusting Metrics: Finally, the algorithm adjusts the metrics to represent the average loss and accuracy per batch. Dividing the accumulated loss and accuracy values by the number of batches (data points processed) ensures that these metrics are interpretable and useful for tracking the model's performance over time.

D. MODEL TESTING STAGE

Algorithm 4 represents the model testing process. During model testing, we calculate the model output and then update the optimizer. Finally, we calculate the accuracy for each batch. The steps of model testing as follows:

- Evaluating Model Performance through Testing Phase: In the realm of deep learning, ensuring that a trained model performs well on unseen data is paramount. This is where the "test_step" algorithm comes into play, helping us to assess the model's capabilities and generalization on data it has not encountered during training.

Algorithm 4 Model Testing Procedure

```
   Data: model, testDataLoader, evaluationMetric,
         computingDevice
   Result: Average Test Loss and Accuracy
1  Function Main:
2      model.SetToEvaluationMode() ;
3      testLoss, testAccuracy ← 0, 0 ;
4      EnterInferenceMode ;
5      for batch in testDataLoader do
6          inputs, targets ← PrepareBatch(batch,
             computingDevice) ;
7          predictions ← model(inputs) ;
8          loss ← evaluationMetric(predictions,
             targets) ;
9          testLoss + = ExtractLossValue(loss) ;
10         correctPredictions ←
             CalculateCorrectPredictions(predictions,
             targets) ;
11         testAccuracy + = correctPredictions /
             TotalPredictions(predictions) ;
12     testLoss ← testLoss /
         TotalBatches(testDataLoader) ;
13     testAccuracy ← testAccuracy /
         TotalBatches(testDataLoader) ;
14     return testLoss, testAccuracy ;
```

- Preparing the Model for Evaluation: The "test_step" begins by setting the deep learning model to evaluation mode. This mode adjusts the model's behavior to ensure that it doesn't perform training-related operations like weight updates. Following this, the algorithm initializes the test loss and accuracy metrics to zero, which will be used to track the model's performance.
- Switching to Inference Mode: During the testing phase, the model must operate in inference mode, ensuring consistent behavior. This mode is enabled using the "with torch.inference_mode()" construct. Within this context, the algorithm can assess the model's performance on the test data.
- Iterating Through Test Data Batches: The heart of the "test_step" is a loop that iterates through the test data batches provided by the data loader. For each batch, the algorithm retrieves the input features (X) and their corresponding labels (y) and ensures they are on the designated device, typically a GPU.
  - Forward Pass: Generating Predictions: The model performs a forward pass with the input features, generating predictions in the form of logits

(raw scores) for each class. These logits reflect the model's confidence in each possible classification.
  - Calculating Loss: To evaluate the quality of predictions, the algorithm calculates the loss, which quantifies how different the predicted logits are from the actual labels. The chosen loss function (loss_fn) is employed for this purpose. The computed loss is added to the test loss, which accumulates the loss across all batches.
- Evaluating Model Accuracy: The model's accuracy is also assessed during the testing phase. The algorithm identifies the predicted class by selecting the class with the highest probability. It then compares this predicted class to the true labels to check how well the model classifies the data. This accuracy measurement is added to the test accuracy metric.
- Adjusting Metrics for Averaging: To provide interpretable and useful metrics, the algorithm divides the accumulated test loss and test accuracy values by the number of batches processed, resulting in the average loss and accuracy per batch.

E. OPTIMIZER

The Adam optimizer is a robust and adaptive optimization algorithm that efficiently adjusts deep learning model parameters during the training process. It balances the momentum and scale of gradients, facilitating quicker convergence and improved training dynamics. The working of Adam optimizer is presented in algorithm 5.

Algorithm 5 Adam Optimizer

```
Data: Initial model parameters $\theta_0$, learning rate $\alpha$,
      $\beta_1$, $\beta_2$, and $\varepsilon$
Result: Updated model parameters $\theta_t$
1  Function main:
       /* Initialize time step and
          first and second moment
          estimates                    */
2      $t \leftarrow 0$ ;
3      $m_0 \leftarrow 0$ ;
4      $v_0 \leftarrow 0$ ;
5      while stopping criteria not met do
6          $t \leftarrow t + 1$ ;
7          $g_t \leftarrow \nabla f(\theta_{t-1})$ ;
8          $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ ;
9          $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot (g_t \odot g_t)$ ;
10         $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$ ;
11         $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$ ;
12         $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}}$ ;
13     return $\theta_t$ ;
```

- Initialization and Setting Parameters: The Adam optimizer begins by initializing several important variables. It sets the time step (t) to 0 and initializes the first and second-moment estimates (m and v) to zero. Additionally, the algorithm takes in essential hyperparameters: the learning rate (α) and two exponential decay rates (β_1 and β_2).
- Main Optimization Loop: The heart of the Adam optimizer is the main optimization loop. It iterates through a series of steps until certain stopping criteria are met. In each iteration, the following operations occur:
  - Incrementing Time Step: The time step (k) is incremented to keep track of the progress through the optimization process.
  - Calculating Gradient: The gradient of the loss function with respect to the model parameters is computed (g_t). This gradient information guides the optimizer in updating the model.
  - Updating First Moment Estimate: The algorithm updates the first-moment estimate (m_t) using an exponentially moving average. This estimate accumulates information about the gradient's direction, incorporating past gradient information.
  - Updating the Second Moment Estimate: The second-moment estimate (v_t) is updated. This estimate keeps track of the square of the gradients, providing information about the scale of the gradient.
  - Bias Correction: The algorithm performs bias correction on both the first and second-moment estimates to address bias introduced in the initial time steps, yielding $\hat{m}^t$ and $\hat{v}^t$.
  - Updating Model Parameters: The critical step in optimization is updating the model parameters (θ_t). This update depends on the learning rateα, the bias-corrected first-moment estimate ($\hat{m}^t$), and the square root of the bias-corrected second-moment estimate ($\hat{v}^t$). The parameter update helps the model move towards the optimal configuration.
- Stopping Criteria: The algorithm stops when specific criteria, such as reaching a certain number of iterations or achieving a desired level of accuracy, are met.

IV. RESULT AND DISCUSSION

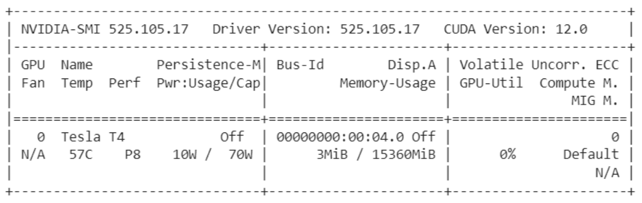
A. SIMULATION ENVIRONMENT

Our experimentation was conducted within a controlled simulation environment, represented in Figure 3, utilizing the Google Colab platform for execution. The system specifications of the environment are as follows:

- PyTorch Version: 2.0.1+cu118
- CUDA Version: 12.0
- GPU: NVIDIA Tesla T4

B. DATA PREPROCESSING

In our study, we employed a series of preprocessing steps to handle the Kaggle dataset containing four distinct classes: "eyes open," "eyes closed," "yawn," and "not yawn."



```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.105.17   Driver Version: 525.105.17   CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   57C    P8    10W /  70W |      3MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

FIGURE 3. Simulation environment.

- **Dataset Distribution**: Our dataset comprises four classes, each with various images. Specifically, there are 579 images in “yawn,” 581 images in “eyes closed,” 580 images in “no_yawn,” and 581 images in “eyes open” for training. Additionally, for testing, there are 144 images in “yawn,” 145 images in “eyes closed,” 145 images in “no_yawn,” and 145 images in “eyes open.”
- **Data Loaders and Batch Size**: To efficiently handle the dataset, we utilized Python’s data loader functionality, which enables us to load and preprocess data in batches. We configured a batch size of 32 for both training and testing data. This batch size helps in optimizing memory usage and accelerates training by processing a subset of data at a time.
- **Data Transformation**: To enhance the randomness of the data and promote generalization, we applied data transformations with a probability of 0.5. These transformations include random horizontal flips, rotations, and adjustments in brightness and contrast. The use of random transformations enriches the dataset by presenting the model with a diverse range of inputs during training, making it more robust to variations in real-world scenarios. Figure 4 and Figure 5 presents the original and transformed images.
- **Image Resizing**: To make the model more lightweight and efficient, we resized the images to a common dimension of 64 × 64 pixels. This reduction in image size not only conserves computational resources but also helps accelerate the training process without significantly compromising the model’s ability to learn meaningful features.

By implementing these data preprocessing steps, we ensure that our model receives appropriately formatted and diverse inputs during training and testing. This, in turn, contributes to the model’s ability to generalize well to real-world scenarios and accurately predict driver behavior, ultimately bolstering the efficacy of our proposed deep learning-based approach within the 6G Vehicular Network context.

C. RESULT PRESENTATION

In our experimentation, the developed deep learning model exhibited a commendable level of performance, both in terms of accuracy and training dynamics. The model took approximately 18 seconds per iteration during the training phase, carried out over a span of 50 epochs, resulting in a total running time of approximately 419.690601 seconds. This runtime indicates efficient model training, taking into consideration the computational complexity involved in processing the dataset.

Image class:yawn|Image_shape:(480, 640, 3)-->[height,width,color]|

FIGURE 4. Sample traning image.

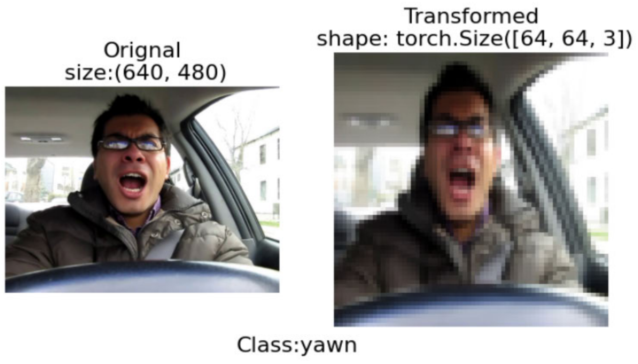


FIGURE 5. Image transformation.

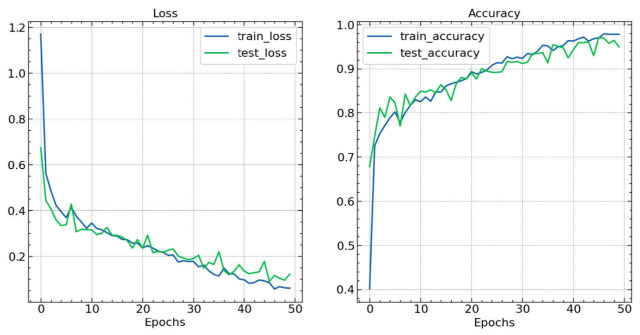


FIGURE 6. Accuracy and loss curve.

The accuracy and loss curves (Figure 6) plotted for both the training and test phases further substantiate the model’s efficacy. The curves demonstrate that the model trains accurately, without apparent signs of overfitting or underfitting. As represented in Figure 6, our model generalizes well to unseen data.

Furthermore, the classification report (Figure 7) and confusion matrix (Figure 8) provides a comprehensive overview of the model’s performance. The precision, recall, and F1-score metrics for each class reflect high levels of accuracy in classification. The model’s ability to accurately distinguish between “Closed” and “Open” classes is

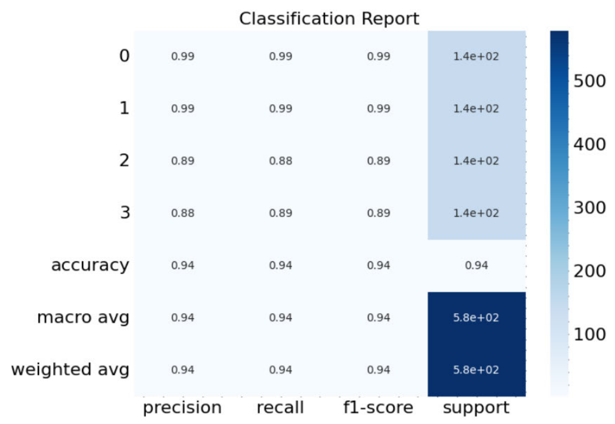


FIGURE 7. Classification report.

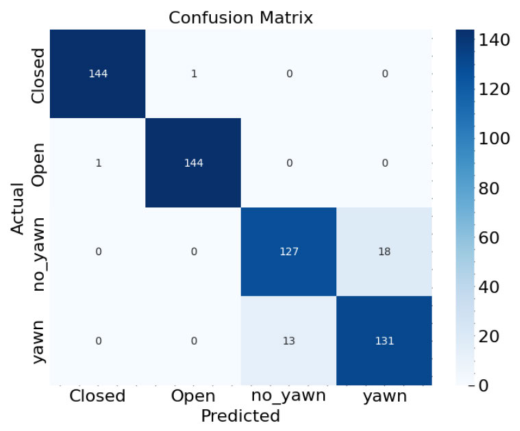


FIGURE 8. Confusion matrix.

relatively high. In addition to that, our model achieved an impressive F1-score of 0.99. The "Yawn" and "No Yawn" classes demonstrate respectable performance with an F1-score of 0.89.

Our model achieved an overall accuracy of 94%. The macro and weighted average F1-scores of 0.94 underscore the model's consistency in delivering accurate class predictions. These performance metrics, in combination with the other assessment results, point towards the model's reliability and its potential to be employed as a viable solution in real-world applications.

D. RESULT COMPARISION

In this section, we conducted a comprehensive comparative analysis to benchmark the performance of our proposed CNN-based approach against a spectrum of existing methods in the field (Table 2). The analysis encompassed a variety of techniques and their associated performance metrics, primarily focusing on accuracy or F1 scores.

The work of Liu et al. [37] adopted a whale optimization algorithm-restricted Boltzmann machine, achieving an accuracy of 90%. In contrast, conventional deep learning models like ResNET50 and Inception V3, which both utilize Convolutional Neural Networks (CNNs), reported higher accuracies of 92.76% and 92.26%, respectively. Similarly, studies conducted by Seong et al. [42] and Sahoo et al. [43] employed CNN frameworks, yielding accuracies of 91.12% and 92%, respectively.

Exploring recurrent neural network architectures, Saleh et al. [44] and Xie et al. [45] implemented LSTM-based methods, recording an F1 score of 91% and an accuracy of 87.26%, respectively. Additionally, Xie and Zhu [46] utilized a Random Forest approach, albeit with a notably lower F1 score of 70.47%

In stark contrast, our proposed work, leveraging an advanced CNN methodology, has demonstrated a superior performance, achieving an accuracy of 94%. This result not only highlights the robustness and efficacy of our approach but also positions it as a leading methodology in this domain, outstripping the performance metrics of the previously established methods.

TABLE 2. Comparative analysis.

Work	Technique Used	Accuracy/F1 score
[37]	whale optimization algorithm-restricted Boltzmann machine	90%
ResNET50	CNN	92.76%%
Inception V3	CNN	92.26%
[42]	CNN	91.12%
[43]	LSTM, Bi-LSTM, GRU	92%
[44]	LSTM-LSTM	91% (F1 score)
[45]	Random Forest	87.26% (F1 score)
[46]	Random Forest	70.47% (F1 score)
Proposed Work	CNN	**94%**

V. CONCLUSION

Due to the development in the field of ITS, there is a need for updated protocols and standards that fulfil the current requirements. In this context, we proposed a deep-learning framework that efficiently identifies drivers' behavior in ITS. Our research has led to promising results in driver behavior detection within the framework of Intelligent ITS and CPS. Through the development and training of our deep learning model, we achieved an impressive accuracy rate of 94% in recognizing critical driver behaviors, including eye closure, open-eye states, yawning, and non-yawning instances. These results are not only a testament to the capabilities of deep learning within CPS-based ITS but also point toward a brighter future for road safety and transportation efficiency. By harnessing the power of artificial intelligence, our system can provide real-time monitoring of driver behaviors, thereby enhancing the safety of our roadways. The practical implications of these results are substantial. The high accuracy and robustness of our model pave the way for its real-world implementation. Such a system has the potential to respond effectively to dynamic and diverse driving conditions, significantly reducing the risks associated with distracted or fatigued drivers. This, in turn, contributes to a safer and more optimized transportation network. For future work, we plan to add more real-time datasets and test our model in real-time. Also, in the future, we will include more parameters and data such as vehicle telemetry data, environmental sensors data, etc.

REFERENCES

[1] R. Nasim and A. Kassler, "Distributed architectures for intelligent transport systems: A survey," in *Proc. 2nd Symp. Netw. Cloud Comput. Appl.*, Dec. 2012, pp. 130–136.

[2] R. K. D. R. K. Dhanaraj, S. K. R. K. Dhanaraj, B.-G.-K. S. Kadry, and Y. N. B.-G. Kang, "Probit cryptographic blockchain for secure data transmission in intelligent transportation systems," *J. Internet Technol.*, vol. 23, no. 6, pp. 1303–1313, Nov. 2022.

[3] Z. Zhou et al., "GAN-Siamese network for cross-domain vehicle re-identification in intelligent transport systems," *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 5, pp. 2779–2790, Sep./Oct. 2023, doi: 10.1109/TNSE.2022.3199919.

[4] A. M. Srivastava, P. A. Rotte, A. Jain, and S. Prakash, "Handling data scarcity through data augmentation in training of deep neural networks for 3D data processing," *Int. J. Semantic Web Inf. Syst.*, vol. 18, no. 1, pp. 1–16, Apr. 2022.

[5] W. Qian, H. Li, and H. Mu, "Circular LBP prior-based enhanced GAN for image style transfer," *Int. J. Semantic Web Inf. Syst.*, vol. 18, no. 2, pp. 1–15, Dec. 2022.

[6] S. Anbukkarasi and C. Dhivyaa, "Ai techniques for future smart transportation," in *Artificial Intelligence for Future Intelligent Transportation*. New York, NY, USA: Academic, 2024, pp. 243–268.

[7] W. Al-Shaar, N. Nehme, O. Bonin, J. Adjizian-Gérard, and M. Al-Shaar, "Passengers receptivity of a new public transport mode: Case of a BRT project in Lebanon," *Comput. Urban Sci.*, vol. 2, no. 1, p. 25, Aug. 2022.

[8] M. Deveci, D. Pamucar, I. Gokasar, M. Köppen, and B. B. Gupta, "Personal mobility in metaverse with autonomous vehicles using Q-rung orthopair fuzzy sets based OPA-RAFSI model," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 12, pp. 15642–15651, Dec. 2023, doi: 10.1109/TITS.2022.3186294.

[9] N. T. U. Nhi, T. M. Le, and T. The Van, "A model of semantic-based image retrieval using C-tree and neighbor graph," *Int. J. Semantic Web Inf. Syst.*, vol. 18, no. 1, pp. 1–23, Feb. 2022.

[10] A. Samson, P. Akinlolu, and O. Olugbenga, "Smart traffic signal control system for two inter-dependent intersections in Akure, Nigeria," *J. Eng. Stud. Res.*, vol. 28, no. 3, pp. 82–92, Oct. 2022, doi: 10.29081/jesr.v28i3.010.

[11] M. Haddad and C. Mansour, "Energy and emission modelling for climate change mitigation from road transportation in the middle east: A case study from Lebanon," in *Climate Change and Energy Dynamics in the Middle East* (Understanding Complex Systems), H. Qudrat-Ullah and A. Kayal, Eds. Cham, Switzerland: Springer, 2019, doi: 10.1007/978-3-030-11202-8_3.

[12] F. J. G. Peñalvo, A. Sharma, A. Chhabra, S. K. Singh, S. Kumar, V. Arya, and A. Gaurav, "Mobile cloud computing and sustainable development: Opportunities, challenges, and future directions," *Int. J. Cloud Appl. Comput.*, vol. 12, no. 1, pp. 1–20, Oct. 2022.

[13] J. Radak, B. Ducourthial, V. Cherfaoui, and S. Bonnet, "Detecting road events using distributed data fusion: Experimental evaluation for the icy roads case," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 1, pp. 184–194, Jan. 2016.

[14] G. Salloum and J. Tekli, "Automated and personalized meal plan generation and relevance scoring using a multi-factor adaptation of the transportation problem," *Soft Comput.*, vol. 26, no. 5, pp. 2561–2585, Mar. 2022.

[15] R. K. S. Rajput, D. Goyal, A. Pant, G. Sharma, V. Arya, and M. K. Rafsanjani, "Cloud data centre energy utilization estimation: Simulation and modelling with iDR," *Int. J. Cloud Appl. Comput.*, vol. 12, no. 1, pp. 1–16, Oct. 2022.

[16] E. Eso, Z. Ghassemlooy, S. Zvanovec, J. Sathian, and A. Gholami, "Fundamental analysis of vehicular light communications and the mitigation of sunlight noise," *IEEE Trans. Veh. Technol.*, vol. 70, no. 6, pp. 5932–5943, Jun. 2021.

[17] J. Zhang and K. B. Letaief, "Mobile edge intelligence and computing for the Internet of Vehicles," *Proc. IEEE*, vol. 108, no. 2, pp. 246–261, Feb. 2020.

[18] M. A. Khoudja, M. Fareh, and H. Bouarfa, "Deep embedding learning with auto-encoder for large-scale ontology matching," *Int. J. Semantic Web Inf. Syst.*, vol. 18, no. 1, pp. 1–18, Apr. 2022.

[19] Q. Ali, N. Ahmad, A. Malik, G. Ali, and W. Rehman, "Issues, challenges, and research opportunities in intelligent transport system for security and privacy," *Appl. Sci.*, vol. 8, no. 10, p. 1964, Oct. 2018.

[20] J. Mwiti, E. Abande, and M. Stephen, "Role of intelligence transport system in the fight against road accidents in Kenya," *Int. J. Comput. Appl.*, vol. 178, no. 11, pp. 17–22, May 2019.

[21] R. W. Liu, Y. Guo, Y. Lu, K. T. Chui, and B. B. Gupta, "Deep network-enabled haze visibility enhancement for visual IoT-driven intelligent transportation systems," *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 1581–1591, Feb. 2023.

[22] S. C. Rajkumar and L. J. Deborah, "An improved public transportation system for effective usage of vehicles in intelligent transportation system," *Int. J. Commun. Syst.*, vol. 34, no. 13, 2021, Art. no. e4910.

[23] B. Tian, B. T. Morris, M. Tang, Y. Liu, Y. Yao, C. Gou, D. Shen, and S. Tang, "Hierarchical and networked vehicle surveillance in ITS: A survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 1, pp. 25–48, Jan. 2017.

[24] Q. Zhang, Z. Guo, Y. Zhu, P. Vijayakumar, A. Castiglione, and B. B. Gupta, "A deep learning-based fast fake news detection model for cyber-physical social services," *Pattern Recognit. Lett.*, vol. 168, pp. 31–38, Apr. 2023.

[25] Z. Liu, H. Song, H. Tan, H. Hao, and F. Zhao, "Evaluation of the cost of intelligent upgrades of transportation infrastructure for intelligent connected vehicles," *J. Adv. Transp.*, vol. 2022, pp. 1–15, Jan. 2022.

[26] G. N. Nguyen, N. H. L. Viet, M. Elhoseny, K. Shankar, B. B. Gupta, and A. A. A. El-Latif, "Secure blockchain enabled cyber–physical systems in healthcare using deep belief network with ResNet model," *J. Parallel Distrib. Comput.*, vol. 153, pp. 150–160, Jul. 2021.

[27] S. Balasubramani and D. J. Arvindhar, "A predictive decision model for an efficient detection of abnormal driver behavior in intelligent transport system," *J. Manage. Inf. Decis. Sci.*, vol. 24, pp. 1–10, Jan. 2021.

[28] S. Bouhsissin, N. Sael, and F. Benabbou, "Driver behavior classification: A systematic literature review," *IEEE Access*, vol. 11, pp. 14128–14153, 2023.

[29] Z. Li, C. Lu, G. Cheng, J. Gong, J. Li, and L. Wei, "Driver behavior modelling at the urban intersection via canonical correlation analysis," in *Proc. 3rd Int. Conf. Unmanned Syst.*, Nov. 2020, pp. 564–569, doi: 10.1109/icus50048.2020.9274914.

[30] Q. Hou and J. Dong, "Robust adaptive event-triggered fault-tolerant consensus control of multiagent systems with a positive minimum interevent time," *IEEE Trans. Syst., Man, Cybern., Syst.*, 2023.

[31] J. V. Tembhurne, M. M. Almin, and T. Diwan, "Mc-DNN: Fake news detection using multi-channel deep neural networks," *Int. J. Semantic Web Inf. Syst.*, vol. 18, no. 1, pp. 1–20, Feb. 2022.

[32] S. Li, D. Qin, X. Wu, J. Li, B. Li, and W. Han, "False alert detection based on deep learning and machine learning," *Int. J. Semantic Web Inf. Syst.*, vol. 18, no. 1, pp. 1–21, Apr. 2022.

[33] A. Gaurav, K. Psannis, and D. Peraković, "Security of cloud-based medical Internet of Things (MIoTs): A survey," *Int. J. Softw. Sci. Comput. Intell.*, vol. 14, no. 1, pp. 1–16, Nov. 2021.

[34] M. Alauthman, A. Al-Qerem, S. Alangari, A. M. Ali, A. Nabo, A. Aldweesh, I. Jebreen, A. Almomani, and B. B. Gupta, "Machine learning for accurate software development cost estimation in economically and technically limited environments," *Int. J. Softw. Sci. Comput. Intell.*, vol. 15, no. 1, pp. 1–24, Oct. 2023.

[35] G. Wu, Y. Li, S. Luo, G. Song, Q. Wang, J. He, J. Ye, X. Qie, and H. Zhu, "A joint inverse reinforcement learning and deep learning model for drivers' behavioral prediction," in *Proc. 29th ACM Int. Conf. Inf. Knowl. Manage.*, Oct. 2020, pp. 2805–2812.

[36] C. Wei, F. Hui, and A. J. Khattak, "Driver lane-changing behavior prediction based on deep learning," *J. Adv. Transp.*, vol. 2021, pp. 1–15, Apr. 2021.

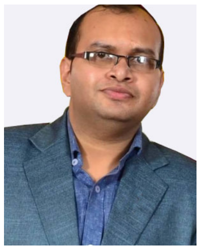
[37] J. Liu, Y. Jia, Y. Wang, and P. Dolezel, "Development of driver-behavior model based onWOA-RBM deep learning network," *J. Adv. Transp.*, vol. 2020, pp. 1–11, Sep. 2020.

[38] K. Jin, X. Li, W. Wang, X. Hua, and S. Qin, "When and where to go next: Deep learning framework for modeling drivers' behaviors using automatic vehicle identification data," *Transp. Res. Rec., J. Transp. Res. Board*, vol. 2676, no. 6, pp. 387–398, Jun. 2022.

[39] S. Arbabi, D. Tavernini, S. Fallah, and R. Bowden, "Learning an interpretable model for driver behavior prediction with inductive biases," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2022, pp. 3940–3947.

[40] A. E. Abdelrahman, H. S. Hassanein, and N. Abu-Ali, "Robust data-driven framework for driver behavior profiling using supervised machine learning," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 4, pp. 3336–3350, Apr. 2022.
[41] L. Jiang, W. Xie, D. Zhang, and T. Gu, "Smart diagnosis: Deep learning boosted driver inattention detection and abnormal driving prediction," *IEEE Internet Things J.*, vol. 9, no. 6, pp. 4076–4089, Mar. 2022.
[42] J. Seong, C. Lee, and D. S. Han, "Neural architecture search for real-time driver behavior recognition," in *Proc. Int. Conf. Artif. Intell. Inf. Commun. (ICAIIC)*, Feb. 2022, pp. 104–108.
[43] G. K. Sahoo, S. K. Das, and P. Singh, "Two layered gated recurrent stacked long short-term memory networks for driver's behavior analysis," *Sādhanā*, vol. 48, no. 2, pp. 1–18, Apr. 2023.
[44] K. Saleh, M. Hossny, and S. Nahavandi, "Driving behavior classification based on sensor data fusion using LSTM recurrent neural networks," in *Proc. IEEE 20th Int. Conf. Intell. Transp. Syst. (ITSC)*, Oct. 2017, pp. 1–6.
[45] J. Xie, A. R. Hilal, and D. Kulic, "Driving maneuver classification: A comparison of feature extraction methods," *IEEE Sensors J.*, vol. 18, no. 12, pp. 4777–4784, Jun. 2018.
[46] J. Xie and M. Zhu, "Maneuver-based driving behavior classification based on random forest," *IEEE Sensors Lett.*, vol. 3, no. 11, pp. 1–4, Nov. 2019.

BRIJ B. GUPTA (Senior Member, IEEE) received the Ph.D. degree in computer science and engineering from the Indian Institute of Technology Roorkee, India, in 2011. He is a Visiting/Adjunct Professor at several universities worldwide. He serves as the Director of the International Center for AI and Cyber Security Research and Innovations (CCRI) and holds the position of Distinguished Professor in the Department of Computer Science and Information Engineering (CSIE) at Asia University, Taiwan. With over 18 years of professional experience, he has authored over 500 papers in journals and conferences, including 35 books and 12 patents, with over 29,000 citations. His research interests include information security, cyber-physical systems, cloud computing, blockchain technologies, intrusion detection, AI, social media, and networking. He serves as a Member-in-Large on the Board of Governors of the IEEE Consumer Technology Society (2022–2024). He has received numerous national and international awards, including the Canadian Commonwealth Scholarship (2009), Faculty Research Fellowship Award (2017) from MeitY, Government of India, IEEE GCCE Outstanding and WIE Paper Awards, and the Best Faculty Award (2018 and 2019) from NIT KKR, respectively. He has been recognized as a Clarivate Web of Science Highly Cited Researcher in Computer Science (top 0.1% researchers globally) consecutively in 2022 and 2023, and has been included in Stanford University's ranking of the world's top 2% scientists consecutively in 2020, 2021, 2022, and 2023. He was selected as the 2021 Distinguished Lecturer in IEEE CTSoc. He currently serves as the Editor-in-Chief of IJSWIS, IJSSCI, STE, and IJCAC and leads a Book Series with CRC and IET press. He has also served as a TPC member for over 150 international conferences and as an Associate/Guest Editor for various journals and transactions.



AKSHAT GAURAV (Graduate Student Member, IEEE) received the master's degree in computer engineering (cyber security) from the National Institute of Technology Kurukshetra, Haryana, India. He is currently a Ph.D. Researcher with Asia University, Taiwan, and a Researcher with Ronin Institute, Montclair, NJ, USA. He is also working on the projects related to DDoS attack detection, intrusion detection, IoT security, cloud/fog computing, cryptography, etc.

KWOK TAI CHUI (Member, IEEE) received the B.Eng. degree in electronic and communication engineering (business intelligence minor) and the Ph.D. degree in electronic engineering from the City University of Hong Kong, Hong Kong, in 2013 and 2018, respectively. He had industry experience as a Senior Data Scientist with Internet of Things (IoT) Company. He is with the Department of Technology, School of Science and Technology, Hong Kong Metropolitan University, as an Assistant Professor. He has more than 90 research publications, including edited books, book chapters, journal articles, and conference papers. His research interests include computational intelligence, data science, energy monitoring and management, intelligent transportation, smart metering, health care, machine learning algorithms, and optimization. He has served in various editorial position in ESCI/SCIE-listed journals, including the Managing Editor for *International Journal on Semantic Web and Information Systems*, a Topic Editor for *Sensors*, an Associate Editor for *International Journal of Energy Optimization and Engineering*.

VARSHA ARYA received the M.S. degree from Rajasthan University, India, in 2015. She has been a Researcher for the last seven years. She has published more than 20 papers in top journals and conferences. Her research interests include business administration, technology management, cyber-physical systems, cloud computing, healthcare, and networking.