LiveDI: An Anti-theft Model Based on Driving Behavior

Hashim Abu-gellban, Long Nguyen, Mahdi Moghadasi, Zhenhe Pan, Fang Jin
Department of Computer Science, Texas Tech University
{hashim.gellban, long.nguyen, mahdi.moghadasi, zhenpan, fang.jin}@ttu.edu

ABSTRACT

Anti-theft problem has been challenging since it mainly depends on the existence of external devices to defend from thefts. Recently, driver behavior analysis using supervised learning has been investigated with the goal to detect burglary by identifying drivers. In this paper, we propose a data-driven technique, LiveDI, which uses driving behavior removing the use of external devices in order to identify drivers. The built model utilizes Gated Recurrent Unit (GRU) and Fully Convolutional Networks (FCN) to learn long-short term patterns of the driving behaviors from drivers. Additionally, we improve the training time by utilizing the Segmented Feature Generation (SFG) algorithm to reduce the state space where the driving behaviors are split with a time window for analysis. Extensive experiments are conducted which show the impact of parameters on our technique and verify that our proposed approach outperforms the state-of-the-art baseline methods.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Security and privacy** → **Authentication**.

KEYWORDS

Driving Behavior, Cybersecurity, Driver Classification, Driver Identification, Machine Learning, Neural Network, Time Series.

ACM Reference Format:
Hashim Abu-gellban, Long Nguyen, Mahdi Moghadasi, Zhenhe Pan, Fang Jin. 2020. LiveDI: An Anti-theft Model Based on Driving Behavior. In *2020 ACM Workshop on Information Hiding and Multimedia Security (IH&MMSec'20), June 22–24, 2020, Denver, CO, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3369412.3395069

1 INTRODUCTION

Nowadays, manufacturers are using a single board as an intelligent controller in the car to be responsible for many functions. For example in the recent model of BMW, what is called as an iDrive interface, can help drivers to use voice commands for navigation. While more cars are equipped with computer modules on-board, which provide an interface between the driver and the mechanical modules of the vehicle, it also exposes an opportunity to reverse engineer the technology to possibly steal the vehicle. According to a report published in 2017, there are 773,139 thefts of motor vehicles in the U.S.[1]. An increasing number of vehicle thefts through onboard computer hacking has been reported. A recent report showed that by exploiting the vulnerabilities of the car onboard system, thieves had begun to steal cars remotely[2].

To address this issue, car manufacturers include a security module to vehicles. This security module is often integrated with Global Positioning System (GPS) and Global System for Mobile Communication (GSM). This tracking embedded component is installed usually in a secret part of the vehicle to transmit the position of the vehicle in the term of coordinates to the GSM system [1–4]. Although using the tracking module in the vehicle makes it easier to find the stolen car, it may not necessarily prevent a burglary scenario in advance. In addition, many car drivers may consider the GPS tracking process to be an invasion of privacy. Another technology is based on biometric approaches. They have shown initial promising results to authorize the car driver [5]. However, due to economic reasons for the car manufacturers and also the accuracy of such models, these techniques are a burden in the industry to be implemented [6].

Virojboonkiate et al. [7] used a dataset for 13 shuttle bus drivers containing 100 sensors. They converted the whole data (9000 million examples) into 4000 histograms and trained them using neural network (NN). This technique required a large matrix size to present the state space (i.e., a histogram with 100 bars) resulting in accuracy (below 76%).

Another approach to detecting car burglary is applying tree-based classification algorithms and supervised neural network, to analyze the driver behavioral habits. Kwak et al. and Martinelli et al. [8, 9] employed Random Forest and J48 for 10 driver dataset (94,401 rows) using the KIA model with 15 and 51 sensors, respectively. However, these approaches classified each row individually without applying data reduction techniques, which required long training time and had slow testing time. [8] also added only μ and σ of a short window size (i.e., 60) for 15 sensors to the input data with high-dimensional spaces (i.e., 45 features in total for each row). Kwak et al. applied the Random Forest algorithm for classification, and calculated the similarity between a new behavior profile and the driving profile of the selected driver by the classification model. The similarity is used for driver verification to determine legitimate/ illegitimate drivers of a car. Also, this verification approach helps police discovering the identity of the illegitimate driver, and deterring burglars. However, the Random Forest, LSTM-FCN, and other eager learning methods [10–14] need a long time to build a model using raw data without decreasing the state size. The number of licensed drivers in the U.S. was approximately 227 million in

IH&MMSec '20, June 22–24, 2020, Denver, CO, USA

ACM ISBN 978-1-4503-7050-9/20/06…$15.00
https://doi.org/10.1145/3369412.3395069

[1] https://ucr.fbi.gov/crime-in-the-u.s/2017/crime-in-the-u.s.-2017/topic-pages/motor-vehicle-theft
[2] https://www.bbc.com/news/technology-29786320 (2014), [Last Access 4/8/2020]

2018[3]. To analyze a large number of drivers, we may breakdown the problem into smaller problems by building a classification model per group of drivers. The new sample of an illegitimate driver is evaluated using all models to select a driver per group. After that, we compare the profiles of these drivers with the new profile to find the identity of the illegitimate driver (the highest similarity). For instance, if we use this methodology in the U.S. with 10 drivers per group and every model takes 6 minutes (e.g., MLSTM-FCN in Table 1), we need roughly 22 million models and approximately 259 years for a single machine to build the models. To address a long training time issue, we need an approach to build the models in a shorter time. Our approach builds a model in 17 seconds and it takes around 12 years (decrease 95%) to build all models. If we use the parallel programming on HPC, it may reduce the building time from years to weeks, which makes our approach more practical for the anti-theft problem and for maintaining the models when the drivers change their driving behaviors.

To tackle these problems of state size and training/ testing time to make the approach suitable for national wide implementation (e.g. the U.S.), we proposed LiveDI (Live Driver Identification), a deep learning based model which predicts driver from driving behaviors requiring short learning time but keeping high performance. The model utilizes the SFG algorithm for feature generation by reducing the input space to speed up the learning and evaluation process. Moreover, it combines the Gated Recurrent Unit (GRU) and FCN architecture to learn short and long term temporal patterns. The GRU architecture helps decreasing training time while maintains temporal learning capability as in the Long Short-Term Memory (LSTM) architecture. The FCN architecture learns higher latent features and enhances the predictive performance. Additionally, a Batch Normalization (BN) unit is also added to the network to enhance its stability. Extensive experiments are done to validate the model. More specifically, our main contributions in this paper are:

- To the best of our knowledge, we are one of the first in combining Gated Recurrent Unit (GRU) and Fully Convolutional Networks (FCN) in multivariate time series setting, to take advantages of the fast training in GRU while maintaining the capability to learn temporal patterns as in Long Short-Term Memor (LSTM). The FCN component is used as an additional latent feature extractor for the classification model. We also added the Batch Normalization (BN) unit to the model, to enhance overall model performance.
- We employed the Segmented Feature Generation (SFG) algorithm for a given multivariate time series in our proposed framework to extract important features while reducing the data size. Combining segments of 11 effective statistical features per sensor enhances the performance and declines the running time for training the model.
- Extensive experiments are conducted to show the advantages of our proposed model. The results show that driving behavior based driver identification with LiveDI is promising and outperforms competing approaches for medium and long multivariate time series.

[3]https://www.statista.com/statistics/198029/total-number-of-us-licensed-drivers-by-state/

2 PROBLEM FORMULATION

Here, we present our multivariate time series driver classification problem as follows:

The input: The input of the model is the measures of M sensors representing different driving behaviors performed by D drivers. We can denote this as pairs of driving behaviors and the performed driver number: $(B, Y) = (B^1, y_1), (B^2, y_2), ..., (B^D, y_D)$, where B^i is the driving behaviors of the driver y_i; $y_i \in [1, D]$ representing the class label. B^i will be a multivariate time series with M dimension, $B^i = [S^1, S^2, ..., S^M]$ representing all driving behaviors recorded by sensors.

Problem definition: Given the input B^i which is driving behaviors compose of M featured sensors $[S^1, S^2, ..., S^M]$, we have to find a function f that predicts which driver the behaviors belong to. Let $\{X_1, X_2, ..., X_F\}$ be the set of F features extracted from B, the goal of this problem is to find the driver $y \in [1, D]$ fitted with the behaviors most: $y = f(X_1, X_2, ..., X_F)$

3 RELATED WORK

We briefly review some prior approaches in the anti-theft solutions for vehicles.

Traditional Classification Algorithms using Decision Tree-Based and Lazy Learning Methods. Kwak et al. [8] analyzed the driver behaviors through a multi-step process using data collected from vehicle sensors and applying the Random Forest algorithm. In another classification based work Ezzini et al. collected driving patterns such as the brake pedal and GPS information in a duration of 1 and 5 minutes, and/ or 3 sensors physically attached to participants (e.g., temperature and ECG). Their approaches mainly utilize traditional classification algorithms like KNN (i.e., lazy learning, no training, too slow), Random Forest, Extra-Trees, Decision Tree, and Gradient Boosting [10]. [11] used smartphones as an alternative to vehicle sensors to collect data. Features such as Accelerometer, Gyroscope, Magnetometer, and GPS collected through phone and ran over a gradient boosting tree (GBT) classifier. Alternatively, a Gaussian mixture model (GMM) was used to identify the driver in [15]. The model was evaluated using a real driving dataset without producing good results.

Deep Learning for Time Series Classification. Deep learning based methods recently received attention by researchers for the driving behavioral classification. The first application of deep learning over GPS data started with work of Dong et al. by using Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) to extract features from GPS data and used them in a classification, which increased the input size 7 times. El Mekki et al. proposed to use LSTM based method for the driving classification problem [13]. His work has shown that a LSTM based methods - FCN-LSTM - gave the best results among other methods that were evaluated (StackedRNN, NoPoolCNN, CNN). In other research, Chen et al. [14] developed an autoencoder NN called DNCAE to identify drivers without reducing the input size (B^i) of NN.

These methods yielded valid results as noted by each study, however, shorter training time with high accuracy is still challenging to address. We proposed to employ recent advances in machine learning techniques to overcome such challenges.

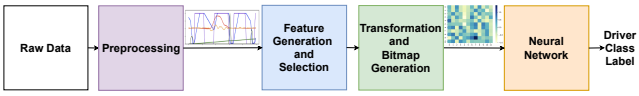

Figure 1: Data processing pipeline in LiveDI.

4 REAL-TIME DRIVER IDENTIFICATION FRAMEWORK

We present our proposed LiveDI in this Section. The framework includes three steps: *preprocessing*, *feature generation* and *real-time driver classification*.

4.1 Preprocessing

A time series is a sequence of values indexed which are ordered according to the time. We split the dataset into training and testing datasets with shuffling and sorting by the class label and the time attributes as the time series classification algorithms assume the data to be in a chronological sequence. We normalize the predictors to make them be treated equally in the classification model.

$$x_m^{'} = \frac{x_m - x_{min_m}}{x_{max_m} - x_{min_m}} \tag{1}$$

where x_{min_m} and x_{max_m} are the minimum and maximum of the m^{th} sensor, $\forall m \in \{1, 2, ..., M\}$. Normalizing the input values between [0,1] is to scale each attribute individually.

4.2 Segmented Feature Generation (SFG)

Training a multivariate time series (MTS) model is expensive. We need to find an approach to train the neural network faster. We employed the SFG algorithm to extract important features from MTS. It also reduces the state space and results in decreasing the training time. Figure 1 depicts the main steps in the data processing pipeline. After preprocessing the raw data, the output passes through the SFG algorithm which composes of two phases: feature generation and selection, and bitmap generation. Afterward, it goes through the neural network for classification.

Algorithm 1: The segmented feature generation algorithm.

function segmented_feature_generation (MTS)

Input : multivariate time series $MTS = X \in \mathbb{R}^{D \times M \times T}$.

Output: multivariate time series
$SFG\_MTS = X^{'} \in \mathbb{R}^{D \times F \times T'}$, where $T' = W * M$

1. transpose MTS per driver (y): $MTST = MTS^T$.
2. split $MTST$ into W segments:
   $SMTST = split\_per\_class\_label(MTST, W)$.
3. generate F features from $SMTST$ for each segment in all drivers (D): $FSMTST = generate\_features(SMTST, F)$.
4. compute the bitmap ($BMTST$) by scaling the data per each variable (m): $x'_m = \frac{x_m - \mu_m}{\sigma_m}$.
5. reshape $BMTST$ into multivariate time series:
   $SFG\_MTS = reshape(BMTST, D, F, W * M)$

return SFG_MTS

We summarized overall processing sequences in Algorithm 1. Let D represent the number of drivers and F denotes the number of generated features. The length of MTS in the input (X) is T and in the output ($X^{'}$) is $T^{'}$. SFG transforms the input of MTS into its transpose (MTS^T) which is sent to the sliding window segmentation, to split the data into W sliding windows. SFG uses a non-overlapped sliding window to extract a summary for each window. The generated features are a set of statistics about the data (i.e., μ, median, absolute energy, σ, σ^2, minimum value, maximum value, $skew$, kurtosis, mean squared error, and mean crossings). Next, the algorithm generates the bitmap by scaling the transformed features [16]. Normalizing features is very important to enhance the efficiency of the classification model as neural network suffers from large values [17]. It reduces the required time for training the neural network, and increases the performance by preventing bias to large input values [18].

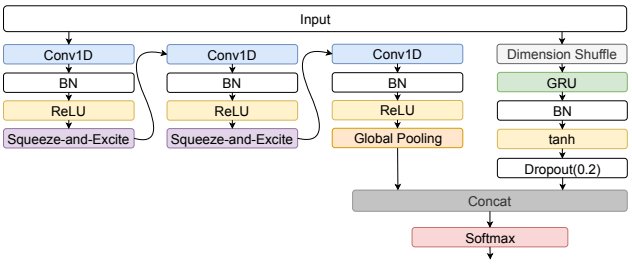

Figure 2: The architectures of LiveDI.

4.3 Real-time Driver Classification

Two key neural network architectures employed to build the classification model are GRU and FCN. We first describe them then discuss their combination.

4.3.1 Gated Recurrent Units Network (GRU). It addresses the vanishing gradient problem of LSTM in a long time series [19]. The update gate prevents the problem as it is designed to learn the amount of past information required to flow for future information. GRU is trained faster than LSTM [20]. It is also similar to LSTM on the ability to learn the pattern of the long-short term. GRU is used to find high-level features at several time steps which increase the performance [21]. It is designed to train a neural network model faster and more efficiently. Below are the update equations:

$$\begin{aligned} z_t &= \sigma(W_z x_t + U_z h_{t-1}) \\ r_t &= \sigma(W_r x_t + U_r h_{t-1}) \\ \tilde{h}_t &= tanh(W x_t + U(r_t \odot h_{t-1})) \\ h_t &= (1 - z_t) h_{t-1} + z_t \tilde{h}_t \end{aligned} \tag{2}$$

where z_t and r_t are the update and the rest gates at time t. $\tilde{h}_t$ denotes to the current memory content, and h_t represent the final memory at the current time step. W and U denote to the weight matrices.

4.3.2 Fully Convolutional Networks (FCN) . Fully Convolutional Networks (FCN) are efficiently used in discovering the semantic segmentation of pictures [22]. The convolution operation contains of three 1-D convolution layers. $\otimes$ denotes the convolution operator.

The basic convolution block is formulated as follows:

$$\begin{aligned} y &= W \otimes x + b \\ s &= BN(y) \\ h &= ReLU(s) \end{aligned} \tag{3}$$

BN is used to reduce the learning time and to support the generalization. FCN uses a low computation cost rectified linear units (ReLU) with BN for lower computation cost after each 1-D convolution.

We use squeeze-and-excite (i.e, computational block) after the layer BN and ReLU, to update channel feature responses by discovering interdependencies between the channels [23, 24]. The computational block is built for any transformation $F_{tr} : X \rightarrow U, X \in \mathbb{R}^{H' \times W' \times C'}, U \in \mathbb{R}^{H \times W \times C}$. The set of filter kernel is represented by $V = [v_1, v_2, ..., v_C]$ where c is the c^{th} filter for the corresponding channel of X. The output of F_{tr} is represented by $U = [u_1, u_2, ..., u_C]$.

$$u_c = v_c \otimes X = \sum_{s=1}^{C'} V_c^s \otimes X^s \tag{4}$$

V_c^s represents the 2D spatial kernel. Squeeze is employed to embed global information by applying the global average pool to compute channel-wise statistics. a statistic $z \in \mathbb{R}^C$ is calculated by reducing the dimensions of U by $H \times W$ spatial dimensions. c^{th} element of z is computed by the following formula:

$$z_c = F_{sq}(u_c) = \frac{1}{(H \times W)} \sum_{i=1}^{H} \sum_{j=1}^{W} u_c(i, j) \tag{5}$$

To aggregate the statistics data, the excite operation is calculated after the squeeze operation by implemented a gating mechanism using sigmoid activation function. The excite is computed by:

$$s = F_{ex}(z, W) = \sigma(g(z, W)) = \sigma(W_2 \delta(W_1 z)) \tag{6}$$

W_1 and W_2 are the dimensionality decreasing/ increasing dimension layers. ReLU is depicted by δ. The output of squeeze-and-excite is calculated by scaling the U with the activation, as follows:

$$\tilde{x}_c = F_{scale}(u_c, s_c) = s_c . u_c \tag{7}$$

$\tilde{X} = [\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_C]$ is the channel-wise multiplication by rescaling u_c with $s_c \in \mathbb{R}^T$, where T denotes to the temporal dimension to calculate the channel-wise statistics. The first two 1-D convolution layers with two BN and ReLU are followed by the squeeze-and-excite computational unit. FCN has the global pooling operation after the three convolution layers, to address the overfitting issue and to decrease the number of weights.

4.3.3 Combining GRU and FCN:. Before combining GRU and FCN architectures, we appended a BN layer and the *tanh* activation function as shown in Figure 2. The experiment on our dataset shows that this extension for multivariate time series enhances the model performance. This combination is defined by:

$$\begin{aligned} h^c &= h^g \oplus h^d \\ y &= softmax(h^c) \end{aligned} \tag{8}$$

where $\oplus$ is the concatenation operator, h^c is the output of concatenating the global pooling (h^g) with the dropout (h^d). The class label is predicted by the $softmax$ activation. The reason behind adding a new layer (BN) is to improve the GRU layer by scaling the parameters [25]. BN helps in stabilizing and improving the performance of the model. It also prevents the neural network from saturating nonlinearities issue.

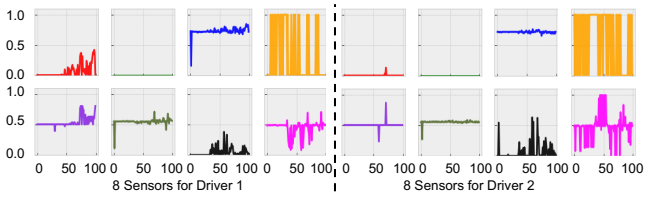

Figure 3: Normalized examples of two drivers' driving patterns during the first 100 time steps. The color encoding: red (accelerator pedal), green (filtered accelerator pedal), blue (acceleration speed longitudinal), orange (brake switch), blue-violet (road gradient), dark olive green (acceleration speed lateral), black (steering wheel speed), and magenta (steering wheel angle).

5 COMPETING APPROACHES

For the first 3 baseline methods, the multivariant time series are converted from 3-D to 2-D shape (the same shape as the raw data [8]), since the *MTS* shape is not suitable for these methods. The following techniques are used as baselines for comparison:

Random Forest: It is also called the random decision forests. The Random Forest algorithm was developed by Breiman [26]. During the training process, the algorithm selects features randomly to build several decision trees from the training dataset with replacement using the bagging technique.

XGBoost: Chen created a classification algorithm called eXtreme Gradient Boosting XGBoost [27]. XGBoost is a scalable tree boosting technique based on the gradient boosting machine [28]. XGBoost can be run in parallel to reduce the training time and have been used in cybersecurity domain [29].

TPOT: The Tree-Based Pipeline Optimization Tool (TPOT) is an Automatic Machine Learning tool (AutoML) [30]. TPOT uses stochastic search algorithms (e.g., genetic programming) and the pipelines' tree representation to automatically build machine learning pipelines.

MLSTM-FCN: Multivariate LSTM-FCN with squeeze-and-excite block (i.e., called MLSTM-FCN) is used for predicting multivariate time series [24].

MALSTM-FCN: These are variants of MLSTM-FCN in which the LSTM is added with attention technique [24].

6 EXPERIMENTS AND RESULTS

6.1 Dataset Description

The dataset contains two paths [8]. It is extracted from real driving (51 sensors) using the KIA Motors Corporation's model in South Korea. Ten drivers participated in this experiment for approximately 23 hours. We used the first path (i.e., 45908 examples), and only 8 attributes were extracted from mechanical sensors instead of 15 sensors used in the previous research [8]. Figure 3 shows the 8 sensors for two drivers during the first 100 time steps of the training dataset. We can see that the two drivers have different speeding habits or the driver one steers smoother while the other driver

Table 1: Overall performance summary of our proposed approach compared with baseline algorithms using metrics. The bold face represents better performance.

Algorithm	Accuracy	Precision	Recall	F1-score	Kappa	Training (secs)	Epochs
Random Forest	0.64	0.64	0.64	0.63	0.59	65	-
XGBoost	0.51	0.52	0.51	0.51	0.45	142	-
TPOT	0.66	0.66	0.66	0.65	0.61	127	-
MLSTM-FCN	0.60	0.55	0.60	0.56	0.55	361	150
MALSTM-FCN	0.60	0.55	0.60	0.56	0.55	361	150
LiveDI	**0.90**	**0.85**	**0.90**	**0.86**	**0.88**	**17**	**15**

shows a larger fluctuation. We split the dataset into training (50%) and testing (50%) datasets.

6.2 Overall Performance Summary

Table 1 shows the comparative results of our proposed algorithm against the baseline methods. In this experiment, we train and test all algorithms using the whole dataset (i.e., 1500 examples for each driver). The classical traditional algorithm did not perform well (accuracy between 51% to 66%). The root cause of this issue is that these algorithms are not scalable for big data. The good trait of the Random Forest algorithm comparable with other traditional algorithms is the execution time (65 seconds) to train the model. It is clear that XGBoost took a long time (2 minutes and 22 seconds) to finish training, resulting in a bad performance (51%). We set the maximum running time for TPOT AutoML tool to be 2 minutes; the early stop may affect the performance of the tool as it searches for the best algorithm and its parameters by applying an optimization algorithm (e.g., the genetic algorithm). The best traditional classification algorithm produced 66% accuracy using the TPOT tool. We believe the traditional algorithms are not suitable for real-time classification because of the poor performance and/ or the long training time to generate a classification model.

MLSTM-FCN and MALSTM-FCN gave unpleasant results (i.e., accuracy 60%) for 150 epochs. This is because they require a larger number of iterations to perform better, which consumes a longer time to train the models especially with big data (i.e., 12000 values per driver = 1500 * 8, where 1500 is the length of MTS and 8 is the number of sensors). As a result, it took approximately 6 minutes to run each algorithm. Our proposed LiveDI addresses all of these issues. We achieve our target by incorporating SFG in LiveDI to find a fast training model and obtain high performance (i.e., 90% accuracy, 85% precision, 90% recall, 86% F1-score, 88% kappa). We trained all the training dataset using LiveDI for only 15 epochs in 17 seconds. Faster training and learning time make LiveDI applicable for real-time driver classification.

6.3 The Effect of SFG

Figure 4 shows that SFG is very effective in our proposal deep neural network architecture. In this experiment, we ran LiveDI with SFG and without SFG using 13 epochs and 2 sliding windows. Different lengths of multivariate time series were used (the first 15, 60, 100, 1000, and 1500 of the whole multivariate time series). LiveDI with SFG outperformed the performance of its variant without SFG. This is because the other variant did not reduce the state space. F1-score of LiveDI increased dramatically in increasing the size of the dataset.

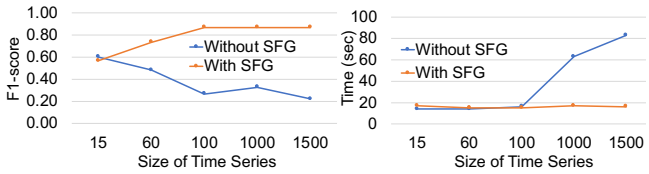

Figure 4: Comparing LiveDI performance incorporating SFG and without SFG algorithm.

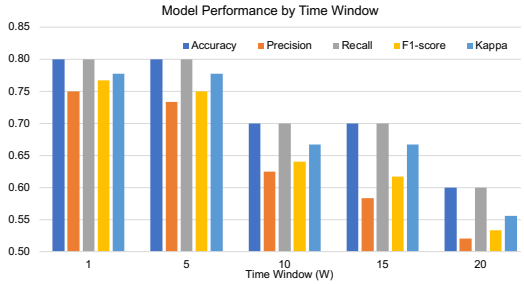

Figure 5: LiveDI performance by varying time window W.

The F1-score was between 56% and 86%. However, the other variant only gave F1-score between 22% and 60%. Additionally, using SFG accelerated the training process, where LiveDI finished training in 17 seconds; however, LiveDI without SFG took 14 to 84 seconds. Therefore, the experiment's results show the usefulness of SFG by speeding up the training process keeping high performance.

6.4 Time Window Segmentation Effect

We ran the LiveDI in the whole training dataset for 15 epochs using different sliding window sizes (i.e., 1, 5, 10, 15, and 20) as shown in Figure 5. As an example using SFG, for one driver behavior, the size of MTS using 1 sliding window is equal to 88 (11 features * 1 window size * 8 variables) while the size of MTS is 440 using 5 sliding windows. For 20 sliding windows, the MTS size is 1760 which is increased approximately 17% of the size of original MTS without using SFG. Generally, increasing window sizes reduces the performance of the LiveDI, as it increases the input size (MTS) of the neural network having a fixed number of epochs. LiveDI needs more epochs to train the long MTS for a large number of sliding windows (e.g., 20). As we fixed the number of epochs to 15, the performance dramatically dropped to 60% (i.e., accuracy and recall).

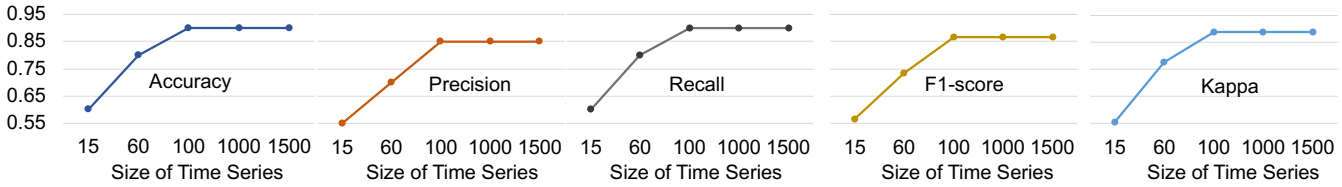

Figure 6: Performance metrics for LiveDI by increasing the size of MTS.

6.5 The Length of The Multivariate Time Series Effect

The purpose of this Section is to show how the size of the training dataset does not affect the performance of our proposed methods using more than 100 time steps. We ran LiveDI using the first 15, 60, 100, 1000, 1500 time steps for *15 epochs* as shown in Figure 6. We trained LiveDI in less than 17 seconds in all of these experiments as their MTS had been decreased to a fix state size since we set the same window size to 2. For an example of the whole training dataset, LiveDI reduces MTS of each driver from 12,000 ($T \times M = 1500 * 8$) to 176 ($F * W * M = 11 * 2 * 8$). Executing neural networks in LiveDI with a 98.53% decrease in the state space reduced the required number of epochs and the time to learn the temporal patterns. Using the reduction technique in LiveDI with proper parameters (e.g., window size = 2) even with a small number of iterations, effectively decreases the training time and increases the performance (e.g., accuracy 90%) for medium and long sequences. LiveDI did not perform well with the short multivariate time series (i.e., length of the multivariate time series < 100) because the statistical values (i.e. features generated by LiveDI) did not provide distinguishable information for each driver's behaviors, and/ or the number of epochs was not enough to train the model.

7 CONCLUSION

In this paper, we proposed LiveDI, a new approach to identify drivers from their driving behaviors. LiveDI employs the SFG algorithm for generating features while reducing the input size. Additionally, LiveDI utilizes GRU (an enhancement of LSTM) combining with FCNs to learn better long-short term temporal patterns. Through an experiment on 10 drivers performing in real car driving environment, our approach showed 90% accuracy and 86% F1-score which outperformed the state-of-art baseline methods.

For future work, we plan to address the cold start problem when a little or no driving behavior of an authorized driver is found in the system. Then, we would like to extend the number of drivers and deploy for other real on-road driving experiments.

REFERENCES

[1] K. Kadiri and O. A. Adegoke, "Design of a GPS/GSM based anti-theft car tracker system." *Current Journal of Applied Science and Technology*, pp. 1–8, 2019.

[2] A. Agarwal and A. Agarwal, "Futuristic automobile accident and theft notifier with location tracker." in *ICICCD*. Springer, 2017, pp. 343–349.

[3] M. Shaikh and K. E. Momin, "GPS and GSM based anti-theft vehicle system." 2018.

[4] P. C. Shreyas, R. Roopalakshmi, K. B. Kari, R. Pavan, P. Kirthy, and P. Spoorthi, "IoT-based framework for automobile theft detection and driver identification." in *ICCNT*. Springer, 2019, pp. 615–622.

[5] M. Villa, M. Gofman, and S. Mitra, "Survey of biometric techniques for automotive applications." in *Information Technology-New Generations*, 2018, pp. 475–481.

[6] N. Agashe and S. Nimbhorkar, "A survey paper on continuous authentication by multimodal biometric." *International Journal of Advanced Research in Computer Engineering & Technology*, vol. 4, no. 11, pp. 4247–4253, 2015.

[7] N. Virojboonkiate, P. Vateekul, and K. Rojviboonchai, "Driver identification using histogram and neural network from acceleration data." in *2017 IEEE 17th International Conference on Communication Technology (ICCT)*, 2017, pp. 1560–1564.

[8] B. I. Kwak, J. Woo, and H. K. Kim, "Know your master: Driver profiling-based anti-theft method." in *PST*. IEEE, 2016, pp. 211–218.

[9] F. Martinelli, F. Mercaldo, V. Nardone, A. Orlando, and A. Santone, "Who's driving my car? a machine learning based approach to driver identification." in *ICISSP*, 2018, pp. 367–372.

[10] S. Ezzini, I. Berrada, and M. Ghogho, "Who is behind the wheel? driver identification and fingerprinting." *Journal of Big Data*, vol. 4, no. 1, 2019.

[11] F. Tahmasbi, Y. Wang, Y. Chen, and M. Gruteser, "Poster: Your phone tells us the truth: Driver identification using smartphone on one turn." in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. ACM, 2018, pp. 762–764.

[12] W. Dong, J. Li, R. Yao, C. Li, T. Yuan, and L. Wang, "Characterizing driving styles with deep learning." *arXiv preprint arXiv:1607.03611*, 2016.

[13] A. El Mekki, A. Bouhoute, and I. Berrada, "Improving driver identification for the next generation of in-vehicle software systems." in *VT*. IEEE, 2019.

[14] J. Chen, Z. Wu, and J. Zhang, "Driver identification based on hidden feature extraction by using adaptive nonnegativity-constrained autoencoder." in *Applied Soft Computing 74*, 2019, pp. 1–9.

[15] C. Miyajima, Y. Nishiwaki, K. Ozawa, T. Wakita, K. Itou, K. Takeda, and F. Itakura, "Driver modeling based on driving behavior and its evaluation in driver identification." in *IEEE 95, no. 2*, 2007.

[16] N. Kumar, V. N. Lolla, E. Keogh, S. Lonardi, C. A. Ratanamahatana, and L. Wei, "Time-series bitmaps: a practical visualization tool for working with large time series databases." in *SIAM data mining*. SIAM, 2005, pp. 531–535.

[17] R. A. Zitar and A. Al-Jabali, "Towards neural network model for insulin/glucose in diabetics-II." in *Informatica 29, no. 2*. Informatica, 2005.

[18] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning." in *15th HotNets*. ACM, 2016, pp. 50–56.

[19] K. Cho, B. V. Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation." in *arXiv preprint arXiv*, 2014.

[20] L. H. Nguyen, Z. Pan, O. Openiyi, H. Abu-gellban, M. Moghadasi, and F. Jin, "Self-boosted time-series forecasting with multi-task and multi-view learning." in *arXiv preprint arXiv:1909.08181*, 2019.

[21] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling." in *arXiv preprint arXiv:1412.3555*, 2014.

[22] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation." in *CVPR*. IEEE, 2015, pp. 3431–3440.

[23] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks." in *CVPR*, 2015, pp. 7132–7141.

[24] F. Karim, S. Majumdar, H. Darabi, and S. Harford, "Multivariate LSTM-FCNs for time series classification." in *Neural Networks 116*. Elsevier, 2019.

[25] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift." in *arXiv preprint arXiv:1502.03167*, 2015.

[26] L. Breiman, "Random forests." *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[27] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system." in *22nd ACM SIGKDD*. ACM, 2016, pp. 785–794.

[28] J. Zhou, E. Li, M. Wang, X. Chen, X. Shi, and L. Jiang, "Feasibility of stochastic gradient boosting approach for evaluating seismic liquefaction potential based on SPT and CPT case histories." in *Journal of Performance of Constructed Facilities 33, no. 3*, 2019.

[29] Z. Chen, F. Jiang, Y. Cheng, X. Gu, W. Liu, and J. Peng, "XGBoost classifier for DDoS attack detection and analysis in SDN-based cloud." in *BigComp*. IEEE, 2018, pp. 251–256.

[30] S. O. R. Sohn, Andrew and J. H. Moore, "Toward the automated analysis of complex diseases in genome-wide association studies using genetic programming." ACM. Proceedings of the Genetic and Evolutionary Computation Conference, 2017, pp. 489–496.