information

Article

An Optimized Deep Learning Approach for Detecting Fraudulent Transactions

Said El Kafhali [1,*], Mohammed Tayebi [1] and Hamza Sulimani [2]

[1] Computer, Networks, Modeling, and Mobility Laboratory (IR2M), Faculty of Sciences and Techniques, Hassan First University of Settat, Settat 26000, Morocco; m.tayebi@uhp.ac.ma
[2] College of Computer Science & Engineering, Umm Al-Qura University, Makkah 24381, Saudi Arabia; hhhsulimani@uqu.edu.sa
* Correspondence: said.elkafhali@uhp.ac.ma

Abstract: The proliferation of new technologies and advancements in existing ones are altering our perspective of the world. So, continuous improvements are needed. A connected world filled with a vast amount of data was created as a result of the integration of these advanced technologies in the financial sector. The advantages of this connection came at the cost of more sophisticated and advanced attacks, such as fraudulent transactions. To address these illegal transactions, researchers and engineers have created and implemented various systems and models to detect fraudulent transactions; many of them produce better results than others. On the other hand, criminals change their strategies and technologies to imitate legitimate transactions. In this article, the objective is to propose an intelligent system for detecting fraudulent transactions using various deep learning architectures, including artificial neural networks (ANNs), recurrent neural networks (RNNs), and long short-term memory (LSTM). Furthermore, the Bayesian optimization algorithm is used for hyperparameter optimization. For the evaluation, a credit card fraudulent transaction dataset was used. Based on the many experiments conducted, the RNN architecture demonstrated better efficiency and yielded better results in a shorter computational time than the ANN LSTM architectures.

Keywords: fraud detection; Bayesian optimization; hyperparameter optimization; deep learning; credit card

Citation: El Kafhali, S.; Tayebi, M.; Sulimani, H. An Optimized Deep Learning Approach for Detecting Fraudulent Transactions. *Information* **2024**, *15*, 227. https://doi.org/10.3390/info15040227

Academic Editor: Gabriel Luque

Received: 14 March 2024
Revised: 1 April 2024
Accepted: 4 April 2024
Published: 18 April 2024

1. Introduction

The explosion of big data has enabled solutions for many complex problems. Different approaches using mathematical simulation and relevant ideas, such as artificial intelligence [1], the enhancement of computer capacity, and the use of super calculators, have made handling big data an easier task [2]. The enhancement of these technologies is necessary due to the continuous amount of data generated by humans every second. Traditional programming is not efficient for fixing data mining problems [3]. So, the idea of making computers function like the human brain, which learns from its tasks and makes complex decisions, has impacted our approach to the most challenging problems.

Likewise, these technologies have impacted global business [4]. Different financial institutions and banks have endeavored to improve the services they provide to their customers by developing more sophisticated payment systems and ensuring the secure transfer of large amounts of money [5]. Blockchain, artificial intelligence, big data, and other technologies have been successfully implemented into production in these network systems [6]. However, fraudulent payments involving credit cards have evolved to a more advanced level [7].

The COVID-19 pandemic is one of the most challenging problems for humanity. Countries have made significant progress in combating this disease. During this time, companies have successfully digitalized their businesses and streamlined their systems for users [8]. Credit cards have become the first method of payment on e-commerce

websites [9]. In contrast, criminals try to breach these payment systems to make illegal transactions; thus, security is essential in these systems. Many solutions for addressing this problem have been proposed, several of which use the abilities of machine learning algorithms for classifying fraudulent transactions based on historical datasets [10].

Every payment system for credit card transactions comprises four components: the merchant, cardholder, acquiring bank, and issuing bank. These four components communicate with each other using HTTP requests to make the transaction possible [11]. The transaction is handled and checked differently by the issuing bank. Their main objective is to confirm that the transaction is legitimate. This system can be described as follows. First, the cardholder presents their credit card to the merchant for the purchase of goods or services. Then, the card is swiped and entered into the point-of-sale software, and the processor sends out a request for authorization through the payment processing networks [12]. The issuing bank authorizes or rejects the transaction based on the funds available. After that, if the transaction is authorized, it is passed through the electronic networks to the processor, and the approval code is delivered to the point-of-sale device at the merchant's location [13]. The issuing bank then sends the money to the processing company to reimburse them for the purchase that was made. This whole process is completed in a matter of seconds. Figure 1 illustrates these steps in detail.

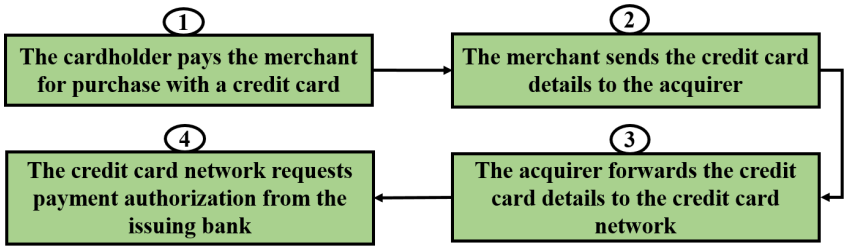

Figure 1. Card payment authorization process.

This paper introduces an innovative adaptive credit card fraud detection system, leveraging three robust deep learning architectures alongside the Bayesian optimization algorithm. Its goal is to allow the selection of the most effective architecture for fraudulent transaction classification. A key contribution of this work lies in adapting the Bayesian optimization approach to efficiently navigate the complex landscape of deep learning model architectures. Unlike traditional methods such as grid search or random search, Bayesian optimization addresses the computational challenges and exponential complexity inherent in deep learning models, offering a more effective search mechanism. The system's performance is evaluated using three distinct scenarios, where the Bayesian algorithm is fine-tuned across 50, 70, and 100 iterations, respectively, and the European credit card dataset is utilized for benchmarking. Furthermore, a comprehensive set of evaluation metrics is proposed to assess the performance of deep learning architectures with hyperparameter optimization using the Bayesian technique. This work significantly advances credit card fraud detection by presenting a holistic approach that integrates advanced techniques to achieve superior accuracy and efficiency in identifying fraudulent transactions.

This work makes the following contributions:

- A hyperparameter tuning technique based on Bayesian optimization is proposed for selecting the best-performing deep learning architecture.
- Bayesian optimization is implemented for designing the best-performing deep learning architectures, including RNN, LSTM, and ANN.
- Several experiments based on the European credit card dataset are performed, and the obtained results show that the RNN is the most efficient with Bayesian hyperparameter optimization compared to the LSTM and ANN techniques.

The rest of this paper is organized as follows. Section 2 provides a review of the literature. Section 3 describes the methods used in this study. Section 4 presents the

proposed methodology and materials. The results and discussion are presented in Section 5. Finally, in Section 6 we present our conclusions and a brief description of our future work.

2. Related Works

The detection of illegal transactions has received much attention in the past decade. This has resulted from the number of transactions made by cardholders and the advanced approaches used by criminals to hack credit card information using techniques such as site duplication. In this section, we review some important works on the detection of fraudulent transactions using various techniques, such as machine learning and deep learning, as well as different approaches for enhancing these models, such as feature engineering, hyperparameter optimization, feature selection, and so on.

In [14], the authors implemented a new solution for fraud transaction detection that was deployed in Apache Spark for real-time fraud identification. This solution used the following technologies: Kafka for programming tasks, Cassandra for dataset management, and Spark for real-time preprocessing and training. This solution was subjected to many experiments, to show its scalability and efficiency. Therefore, the paper introduces a novel fraud detection solution leveraging Apache Spark, Kafka, and Cassandra for real-time processing, demonstrating scalability and efficiency. Nonetheless, integrating multiple technologies may introduce complexity, while the system's adaptability to evolving fraud tactics requires further examination. Another solution, presented in [15], exploits several data mining techniques. Its main idea is to use a contrast vector for each transaction based on its cardholder's historical behavior sequence. Then they profile the distinguishing ratio between the current transaction and the cardholder's preferred behavior. After that, they used ContrastMiner, an algorithm implemented for discovering hidden patterns and differentiating fraudulent from non-fraudulent behavior. This was followed by combining predictions from different models and selecting the most effective pattern. Experiments conducted on real online banking data demonstrate that this solution yields significantly higher accuracy and lower alert volumes compared to the latest benchmark fraud detection system. The latter incorporates domain knowledge and traditional fraud detection methods. However, the utilization of ContrastMiner can be regarded as a potent approach for mitigating interpretability issues. Numerous experiments with extensive real online banking data have demonstrated its superiority in accuracy and reduction in alert volumes compared to conventional fraud detection methods, suggesting promising advancements in fraud detection efficiency.

Likewise, various deep learning architectures are successfully implemented for fraud transaction detection. They show efficient results in handling fraud transactions in many credit card datasets used for computation. For example, ref. [16] invented a new approach for identifying fraudulent transactions. For this solution to detect fraud, hierarchical cluster-based deep neural networks (HC-DNNs) utilize anomaly characteristics pre-trained through an autoencoder as initial weights for deep neural networks. Through cross-validation, this method detected fraud more efficiently than conventional methods. In addition, the suggested solution can help discover the relationship between fraud types. Otherwise, mobile transactions are becoming more popular due to the increase in the number of smartphones that are attacked by fraudsters. Thus, securing those systems for payment is crucial. That is why our research introduces a novel approach for fraud detection using hierarchical cluster-based deep neural networks (HC-DNNs), leveraging anomaly characteristics pre-trained via autoencoder as initial weights. Despite its superior performance demonstrated through cross-validation, limitations arise with the growing prevalence of mobile transactions prone to fraud attacks, necessitating enhanced security measures to safeguard payment systems. The supervised machine learning algorithm XGBboost classifier [17] is used to propose a solution [18]. The obtained results demonstrate the strength of this solution.

Similarly, another solution occurs in [19]. It detects credit card fraud transactions via employing 13 machine learning classifiers and a real credit card dataset. Their idea was

to generate aggregated features using the genetic algorithm technique. Those generated features were compared with the original feature, as the best representative feature for identifying fraudulent transactions. As a result, based on many experiments, the aggregated features are more representative and efficient in identifying fraudulent transactions. The approach in this article presents a promising solution by combining multiple models. It generates aggregated features using the genetic algorithm technique. Despite yielding promising outcomes in various experiments, potential limitations may arise from dataset quality and scalability concerns, necessitating further investigation to assess real-world applicability accurately. In [20], the authors proposed an efficient credit card fraud transaction solution based on representational learning. This solution implemented an innovative network architecture via an efficient inductive pooling operator and a careful downstream classifier. Several experiments conducted on a real credit card dataset demonstrated the outperformance of the proposed solution against the state-of-the-art methods. Generally, this introduces a credit card fraud detection solution using representational learning, showcasing superior performance via innovative network architecture. While promising, scalability challenges and the solution's adaptability to evolving fraud tactics require further scrutiny for real-world efficacy.

In [21], the authors implemented an intelligent credit card fraud transaction detection system. They implemented an aggregation strategy to classify fraudulent transactions. Its mechanism was to aggregate transactions to capture the cardholder's period behavior and use them for model estimation. This solution is benchmarked on a real credit card dataset, and it shows higher performance in stopping abnormal transactions. Furthermore, the authors proposed a credit card fraud transaction detection method based on hyperparameter optimization [22]. They used a differential evolution algorithm for selecting the performing hyperparameters of the XGboost algorithm. They benchmarked their solution with state-of-the-art classifiers. As a result, the proposed solution is efficient in distinguishing between fraud and non-fraud transactions. However, the solution's scalability and adaptability to emerging fraud tactics necessitate further examination for practical deployment.

Unlike the cited solutions, our research introduces an adaptive system capable of effectively combating fraudulent transactions. By leveraging optimization techniques and exploiting the power of various deep learning architectures such as RNN, ANN, and LSTM, our approach aims to uncover hidden patterns within the dataset, facilitating the classification of fraudulent from non-fraudulent transactions. Notably, we employ Bayesian optimization as our main method for selecting the best architecture, addressing the complexity of deep learning models and resource-intensive computations. This method stands out for its effectiveness in adjusting hyperparameters and effectively exploring the search space.

3. Background

3.1. Deep Learning Technique

Deep learning has become important in academic research [23]. It is a subfield of artificial intelligence and a part of machine learning techniques [24]. It derives its idea from the human brain's mechanism, in which a large number of neurons are connected to discover hidden patterns in the output of a dataset [25]. Each deep learning architecture has three parts. First, the input layer is responsible for transmitting its inputs to the hidden layer. Second, the hidden layer is responsible for transmitting information from the input layer to the output layer through certain processes. Third, the output layer is responsible for producing outputs based on the input layer's information. There are as many neurons in the input layer as there are features that need to be taught to the network. The hidden layer of a neural network is determined by the solution to the problem. As a result, the number of hidden layers varies depending on the problem. The output layer classifies or labels the information from the input layer using calculations. The workflow of deep learning architectures is described in Figure 2.

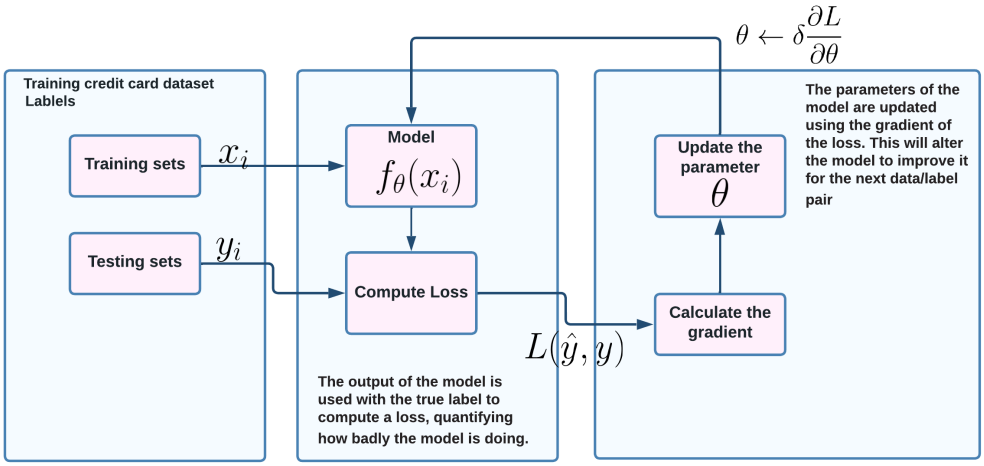

Figure 2. Workflow of deep learning architectures.

In each neuron in the hidden layer k, all inputs are summed with weight, then an activation function is applied as in Equation (1).

$$a_k = activation(\sum_{i=1}^{n} w_i x_i + bias_k) \tag{1}$$

where the parameters of the hidden layer k are a_k, which is the activation function output, w_i is the weight, and $bais_k$ is the vector bias. The Equation (2) explains the output stage operation.

$$\hat{y} = sigmoid(\sum_{i=1}^{n} w_h a_{k-1} + bias_k) \tag{2}$$

We use x_i to refer to the input features; otherwise, we use y to refer to the label, which has two values: 1 if the current transaction is abnormal, and 0 if it is normal. In addition, $f(x)$ represents the deep learning classifier. This model takes an input of x and outputs $\hat{y}$. So $\hat{y} = f(x)(*)$. This model has various parameters denoted by θ. As a result, the formula $(*)$ is transformed into $\hat{y} = f_\theta(x)$. The goal of the deep learning architecture was to select the best-performing parameters θ by optimizing a loss function using Equation (3) below:

$$\min_{\theta} \sum_{i=0}^{n} L(f_\theta(x_i), y_i) \tag{3}$$

To optimize Equation (3), gradient descent was implemented. This algorithm's mechanism is as follows. Suppose θ_k is the current state of the model that we want to improve. The next θ_{k+1} state is calculated using Formula (4) below:

$$\theta_{k+1} = \theta_k - \delta \frac{1}{n} \sum_{i=0}^{n} \frac{\partial L(f_{\theta_k}(x_i), y_i)}{\partial \theta_k} \tag{4}$$

where δ is the learning rate parameters, whose goal is to scale the gradient and thus control the step size, and $L(f_\theta(x_i), y_i)$ is the loss function. In this study, we are using adaptive momentum estimation as an optimization technique for the gradient descent algorithm. This technique combines gradient descent with momentum and Root Mean Square Propagation (RMSP) [26]. The goal of the momentum algorithm was to accelerate the gradient descent by taking into consideration the exponentially weighted average of the gradient. This makes the optimization converge faster. This technique is formulated mathematically by Equation (5) below:

$$\theta_{k+1} = \theta_k - \delta m_k \tag{5}$$

where

$$m_k = \gamma_1 m_{k-1} + (1-\gamma_1)\frac{\partial L}{\partial \theta_k} \tag{6}$$

Otherwise, the RMSP algorithm takes the exponential moving average instead of making the cumulative sum of the squared gradient by Equations (7) and (8) below.

$$\theta_{k+1} = \theta_k - \frac{a_k}{\sqrt{\nu_k + \varepsilon}}\frac{\partial L}{\partial \theta_k} \tag{7}$$

with

$$\nu_k = \gamma_2 \nu_{k-1} + (1-\gamma_2)\left[\frac{\partial L}{\partial \theta_k}\right]^2 \tag{8}$$

This is the mathematical formulation of the Adam optimizer based on Formulas (7) and (8) above:

$$m_k = \gamma_1 m_{k-1} + (1-\gamma_1)\frac{\partial L}{\partial \theta_k} \tag{9}$$

$$\nu_k = \gamma_2 \nu_{k-1} + (1-\gamma_2)\left[\frac{\partial L}{\partial \theta_k}\right]^2 \tag{10}$$

where γ_1, γ_2 are the decays of first and second momentum, respectively. Using Equations (11) and (12), we calculate bias-corrected momentum estimates for the first and second:

$$\hat{m}_k = \frac{m_k}{1-\gamma_1^k} \tag{11}$$

$$\hat{\nu}_k = \frac{\nu_k}{1-\gamma_2^k} \tag{12}$$

where $\hat{m}_k$ and $\hat{\nu}_k$ are the calculated bias estimates. Finally, we update the weights by Equation (13).

$$\theta_{k+1} = \theta_k - \delta\frac{\hat{m}_k}{\sqrt{\hat{\nu}_k} + \varepsilon} \tag{13}$$

while ε is a smoothing term that avoids division by zero (usually in the range of 10^{-8}). A summary of these steps can be found in Algorithm 1.

Algorithm 1 Adam optimization algorithm

Input: Learning rate δ, Objective function f, initial parameters θ, $m_0 = 0, \nu_0 = 0$, $\gamma_1, \gamma_2 \in [0,1], k = 0$
while θ not converged **do**
  $k = k + 1$
  $m_k = \gamma_1 m_{k-1} + (1-\gamma_1)\frac{\partial L}{\partial \theta_k}$
  $\nu_k = \gamma_2 \nu_{k-1} + (1-\gamma_2)\left[\frac{\partial L}{\partial \theta_k}\right]^2$
  $\hat{m}_k = \frac{m_k}{1-\gamma^k}$
  $\hat{\nu}_k = \frac{\nu_k}{1-\gamma_1^k}$
  $\theta_{k+1} = \theta_k - \delta\frac{\hat{m}_k}{\sqrt{\hat{\nu}_k}+\varepsilon}$
end while
Output: θ_k

3.2. Recurrent Neural Network (RNN)

The recurrent neural network architecture (RNN) is a deep learning model designed for handling sequential datasets [27]. In this dataset, each sample is represented as a sequence of values. Using this architecture, we can denote the feature matrix as $X = (x^1, x^2, \cdots, x^n)$ and the label variable as y. The RNN takes one element x^i of X at a time t and processes it

with a block of neural networks, which take into account the prediction y_{i-1} of the previous block. Each block has two inputs and produces two outputs [28]. Equations (14) and (15) evaluate the activation function and the outputs.

$$h^i = activation_h(\theta_{hh}h^{i-1} + \theta_{hx}x^{i-1} + b_h) \tag{14}$$

$$y^i = activation_h(\theta_{yh}h^i + b_y) \tag{15}$$

where $\theta_{hh}, \theta_{hx}, \theta_{yh}$ represent the matrix of weights for computing linear transformation. b_h, θ_y are two bias vectors. Figure 3 illustrates the RNN architecture.

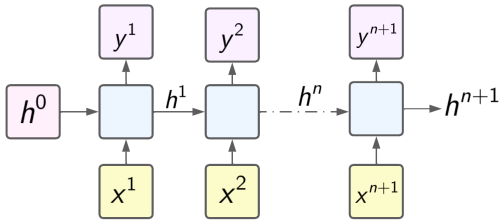

Figure 3. Recurrent neural network architectures.

3.3. Long Short-Term Memory (LSTM)

A long short-term memory network is another deep learning architecture that is like the RNN architecture [29]. The main difference is that in LSTM, a set of units is used to control the information. Figure 4 shows the LSTM architecture. The information flowing through successive positions in this diagram clearly shows the cell state C^{i-1} and the hidden state h^{i-1}. As the previous states are propagated to the next position, the cell state serves as the information from the previous states, and the hidden state determines how the information should be propagated. The hidden state h^i also serves as the output of this position [30].

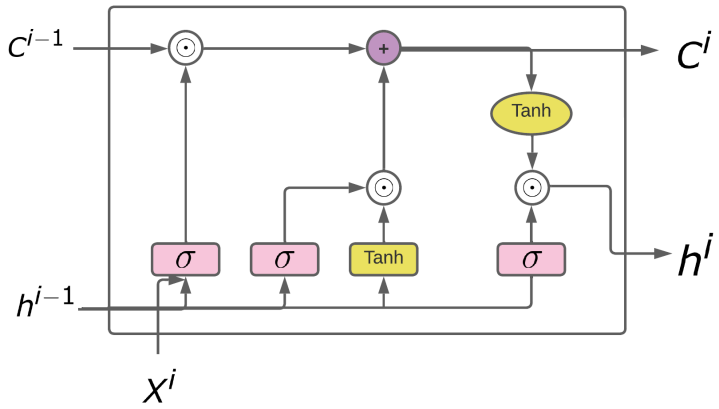

Figure 4. Long short-term memory architecture.

In the first steps of the LSTM system, the forget gate decides what information from the previous cell state to discard based on the two inputs h^{i-1} and x^i. The return vector V^i contains values between 0 and 1 for every element in the cell state C^{i-1}. This vector specifies how each C^{i-1} element's information is discarded. Mathematically, this process is formulated as in Equation (16).

$$V^i = sigmoid(\theta_V x^i + \theta'_V + b_V) \tag{16}$$

where θ_V and θ'_V are two matrices of weights, and b_V is the bias term.

In the next step, we determine what information from the new input x^i should be stored in the new cell state using Equation (17).

$$I_i = sigmoid(\theta_I x^i + \theta'_I + b_I) \tag{17}$$

The x^i vector is passed through a block of neurons to generate $\hat{C}^i$, which contains the candidate value for generating a new cell state using Equations (18) and (19) below:

$$\hat{C}^i = tanh(\theta_C x^i + \theta'_C h^{i-1} + b_C) \tag{18}$$

$$C^i = V^i \odot C^{i-1} + I^i \odot \hat{C}^i \tag{19}$$

where $\odot$ is the Hadamard product. As a final step, the hidden state h_i is generated using Equation (20).

$$h^i = O_t \odot tanh(C^i) \tag{20}$$

where, using Equation (21), O_i is the output gate, which is generated similarly to the forget gate.

$$O_i = sigmoid(\theta_O x^i + \theta'_O + b_O) \tag{21}$$

3.4. Bayesian Optimization

Despite using deep learning approaches instead of traditional machine learning algorithms, the architecture for accurate deep learning is still a major problem. This requires much effort to understand each hyperparameter in the search space and how it can affect the black-box function. Bayesian optimization is a solution for deep learning hyperparameter tuning [31]. The Bayesian approach is an efficient way of finding the extrema of functions that are computationally expensive. This method can be used when solving a black-box function. Using a deep learning architecture black-box function f, the optimization objective is to find the maximum value at the sampling point of Equation (22).

$$x^* = \arg\min_{x \in H} f(x) \tag{22}$$

where H is the search space for hyperparameters. This space contains continuous, categorical, and discrete variables. The Bayesian optimization algorithm relies on Bayes' theorem [32]. It can be described as follows. Considering evidence data E, the posterior probability $P(M \mid E)$ related to the classifier M is proportional to the likelihood $P(E \mid M)$ of over-serving H given model M multiplied by the prior probability of $P(M)$ calculated from Equation (23).

$$P(M \mid E) = P(E \mid M)P(M) \tag{23}$$

Bayesian optimization consists of combining the prior distribution of the function $f(x)$ with sample information to come up with the posterior distribution based on the Gaussian process. Then, the posterior distribution is used to determine where the function $f(x)$ is maximized. The criterion is represented by a utility function ϱ, which is also called the acquisition function. Additionally, ϱ is used for determining the next sample point to maximize the expected utility, taking into account sampling from areas of high uncertainty and high values. This helps reduce the number of samples collected. Furthermore, performance is enhanced even when the function has multiple local maxima.

As demonstrated, Algorithm 2 represents the main tasks of the Bayesian optimization workflow. It is composed of two blocks: Steps 3 and 4 of updating the posterior distribution and optimizing the acquisition function. The posterior distribution is continuously updated as more observations are collected. Based on the updated posterior, the point where the acquisition function is maximized is identified and added to the training dataset. The entire procedure is repeated until the maximum number of iterations is completed or the difference between the current value and the best value thus far is less than a predetermined threshold. It should be mentioned that, compared to other optimization techniques, including gradient descent algorithms, Bayesian optimization does not require

the explicit definition of function f. As a result, it has a larger range of applications. Here are the steps of Bayesian optimization:

Algorithm 2 Bayesian Optimization

Input : Objective function f, hyperparameter space H, initial data D_0
for $i = 1, \ldots, max\_iter$ **do**
    Fit GP to the data D_i
    Maximize the Acquisition function α_i over H to find the new iterate $x_{i+1} = \text{argmin}_{x \in \mathcal{H}} \alpha_i(x)$
    Evaluate $score_{i+1} = f(x_{i+1})$
    Update the data $D_{i+1} = D_i \cup (x_i, score_i)$
end for
Output:$(x_{\max}, score_{\max})$

1. Initialization: Choose a set of initial hyperparameters to sample the objective function.
2. Model construction: Fit the Gaussian Process to the observed data to catch the underlying structure of the function.
3. Acquisition function: Use the expected improvement to balance exploration and exploitation and determine the next best point to sample.
4. Sampling: Evaluate the objective function at the next best point determined by the acquisition function.
5. Updating: Update the Gaussian Process model with the newly obtained sample.
6. Repeat steps 3–5 until the maximum number of iterations is reached
7. Make the best hyperparameter values found by the optimization process as the optimum.

3.4.1. Gaussian Process

A Gaussian Process (GP) is a new supervised approach for solving probabilistic classification or regression problems [33]. Based on Bayesian learning and stochastic Gaussian Processes, this approach is characterized by two important pieces of information, the mean function (24)

$$m(x) = E(f(x)) \tag{24}$$

and the covariance function (25)

$$c(x, y) = E(f(x) - m(x)f(y) - m(y)) \tag{25}$$

Additionally, the standard notation of the GP with its mean function $m(x)$ and its covariance $c(x, y)$ is given by Formula (26) below:

$$f \sim GP(m(x), c(x, y)) \tag{26}$$

It is common to take the mean function as zero, but you can choose any function you want. We can always center our observed outputs to have a zero mean, and the covariance function must be positive definite distributions with finite dimensions. Furthermore, the exponential square function is a popular choice:

$$c(x_i, x_j) = exp(-\frac{1}{2}\left\|x_i - x_j\right\|^2) \tag{27}$$

where x_i and x_j are two samples in the dataset. Respectively, if $\left\|x_i - x_j\right\| < \varepsilon$, then $c(x_i, x_j) \sim 1$; otherwise, $c(x_i, x_j) \sim 0$. The following are the steps involved in determining the posterior distribution of $f(x)$:

- We denote by $D = \{(x_i, f(x_i), \cdots, (x_n, f(x_n)\}$ the training sets. The function values f are drawn according to a multivariate normal distribution, where

$$C' = \begin{bmatrix} c(x_1, x_1) & c(x_1, x_2) & \cdots & c(x_1, x_n) \\ c(x_2, x_1) & c(x_2, x_2) & \cdots & c(x_2, x_n) \\ \vdots & \vdots & \cdots & \vdots \\ c(x_n, x_1) & c(x_n, x_2) & \cdots & c(x_n, x_n) \end{bmatrix} \tag{28}$$

- In mathematics, C' is a function used to measure the degree to which two points are approximate samples. Furthermore, the diagonal element, without considering the effect of noise, consists of 1.
- Based on the function f, we calculate the function value $f_{i+1} = f(x_{i+1})$ at x_{i+1}. According to the assumption of GP, f_{i+1} is a $i+1$ dimensional normal distribution function, where

$$\begin{bmatrix} f_{1:i} \\ f_{i+1} \end{bmatrix} = \left(0, \begin{bmatrix} C & C \\ C^T & C(x_{i+1}, x_{i+1}) \end{bmatrix}\right) \tag{29}$$

where

$$f_{1:i} = [f_1, f_2, \cdots, f_i] \tag{30}$$

and

$$f_{1+i} \sim N(\mu_{i+1}, \sigma^2_{i+1}) \tag{31}$$

with

$$\mu_{i+1}(x_{i+1}) = C^T C^{-1} f_{i+1} \tag{32}$$

$$\sigma^2_{i+1}(x_{i+1}) = -C^T C^{-1} C + C(x_{i+1}, x_{i+1}) \tag{33}$$

3.4.2. Acquisition Function

Bayesian optimization employs the acquisition function ϱ to derive the maximum of the function f after collecting the posterior distribution of the objective function [34]. Typically, we assume that the large value of the objective function f matches the high value of the acquisition function. Consequently, increasing the acquisition function is the same as increasing function f.

$$x^* = arg \max_{x \in H} \varrho(x \mid D) \tag{34}$$

There are various types of acquisition functions. Examples include the probability of improvement (PI), upper confidence bound (UCB), and expected improvement (EI). In hyperparameter optimization, the used acquisition function is the expected improvement (EI). Anticipating the level of improvement a point can accomplish when investigating the area around its current optimum value is calculated by the EI function. The present ideal value point can be the local optimal solution, and the algorithm will locate the optimal value point in other locations in the domain if the improvement in the function value is smaller than the expected value after the procedure is run.

The level of development, the difference between the function's value at the sample point, and the current optimal value are known as I. The improvement function is 0 if the function value at the sample point is smaller than the current optimum value.

$$I(x) = max\{0, f_{i+1}(x) - f(x^*)\} \tag{35}$$

The EI function optimization technique dictates that we attempt to maximize EI in relation to the present optimum value $f(x^*)$.

$$\begin{aligned} x &= arg \max E[I(x)] \\ &= arg \max E[max\{0, f_{i+1}(x) - f(x^*)\}] \end{aligned} \tag{36}$$

where I is a random variable normally distributed with mean $\mu(x) - f(x^*)$ and standard deviation $\sigma(x)^2$. The probability density function of I is

$$f(I) = \frac{1}{\sqrt{2\pi}\sigma(x)} \exp\left(\frac{-(\mu(x) - f(x^*) - I)^2}{2\sigma^2(x)}\right) dI \tag{37}$$

EI is defined as

$$\begin{aligned} EI &= \int_{\infty}^{\infty} If(I)dI \\ &= \int_{0}^{\infty} \frac{1}{\sqrt{2\pi}\sigma(x)} \exp\left(\frac{-(\mu(x) - f(x^*) - I)^2}{2\sigma^2(x)}\right) dI \\ &= \sigma(x)[\Omega\phi(\Omega) + \psi(\Omega)] \end{aligned} \tag{38}$$

where

$$\Omega = \frac{\mu(x) - f(x^*)}{\sigma(x)} \tag{39}$$

Moreover, ϕ and ψ refer to the cumulative distribution and the probability density of the standard normal distribution, respectively. Those functions are defined as follows:

$$\phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp(-\frac{v^2}{2}) dv \tag{40}$$

and

$$\psi(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right) \tag{41}$$

4. Methodology and Materials

4.1. Dataset

This subsection describes in detail the credit card dataset used for computation (Table 1). The used dataset represents credit card transactions made by European cardholders that were made in September 2013 within two days [35]. This dataset was collected by two researchers as part of a partnership project between Libre Brussels University (LBU), and Worldline Company, which specializes in money transfer services. Their objective was to use big data approaches to catch fraudulent transactions. This dataset contains 30 features including the time, which denotes the time of the transaction, the number of transactions, and 28 other attributes named V1 to V28. The result of the transformation of the original attributes is set by using the Principal Component Analysis (PCA) technique [36].

Table 1. Features description.

Variable	Definition	Type
Class	Target feature in this dataset. Takes two values: 0: Legitimate; 1: Fraud	Categorical
Amount	The amount of the transaction sample	Numeric
Time	The difference in time between the first and the current transactions, in seconds	Numeric
V1 to V28	Features transformed using PCA technique to protect cardholders' privacy and confidentiality	Numeric

4.2. Data Preprocessing

The data preprocessing step is crucial in every machine learning pipeline. Its objective is to make the machine learning algorithm able to discover hidden patterns in the dataset.

In this study, various techniques are proposed for processing the computation dataset. These techniques are presented in the rest of this subsection. These data consist of credit card fraud transactions made by European cardholders. All features in these data are numerical. In addition, there is no missing value. All features are scaled except for time and amount. We propose MinMaxscaler (Equation (42)) for scaling those features.

$$x_{i,j} = \frac{x_{i,j} - Min(X_j)}{Max(X_j) - Min(X_j)} \tag{42}$$

where X_j is the feature j, and $x_{i,j}$ is the value of the feature X_j of sample i.

Figure 5 shows the distribution of the target variables for the used datasets. From this figure, we notice that our dataset is imbalanced, proved by the fact that the class of fraud transaction represents 0.17% of all transactions. Thus, a resampling technique is crucial for more accurate results.

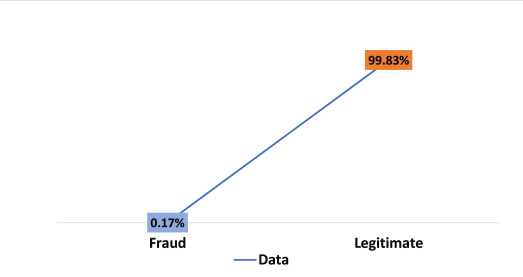

Figure 5. Target variable distribution per fraud and non-fraud transactions.

4.3. Imbalanced Learning

For machine learning algorithms, learning from imbalanced datasets is a challenging task [37]. It predicts all samples as the majority class, which leads to poor generalization and performance because they are not able to discover the hidden patterns for the minority class. The literature provides a variety of approaches to solving the problem, including resampling the majority class or generating a new fake sample of the minority class. In this article, we use a method called random undersampling (RUS) for balancing the dataset [38]. In this method, the steps are described in Algorithm 3, as follows:

Algorithm 3 RUS technique

1: Input: $D = D^0 \cup D^1$
2: $D' = D^1$
3: **for** $j = 1, 2, \cdots, card(D^1)$ **do**
4: Choose a sample at random. $x \in D^0$
5: $D' = D^{'} \cup \{x\}$
6: Remove x from D^0
7: **end for**
8: Output: D'

where D^1 denotes the fraud transaction subset and D^0 refers to the legitimate transaction subset in our dataset, and D' is the new undersampled dataset.

4.4. Hyperparameter Optimization Processes

Deep learning is a new approach to machine learning research that aims to bring it closer to artificial intelligence by mimicking the human brain's mechanisms. Likewise, the performance of each deep learning architecture is very dependent on many decisions. These include choosing the right neural network design, training procedures, and techniques for hyperparameter optimization [39]. In this article, we target enhancing the

detection of illegal fraud transactions using hyperparameters for selecting the best architectures of the used deep learning classifiers. This results from the fact that hyperparameter optimization has a significant impact on the performance of a classifier [40]. Various methods exist for this reason. These include grid search, random search, and metaheuristic algorithms [41]. These techniques have been employed successfully in many studies. This study aims at finding the best-performing method using Bayesian optimization. First, we denote by H the set of all hyperparameter values of a classifier. The set H can be interpreted as a real-valued n-dimensional vector, where n is the number of hyperparameters. Each hyperparameter can take a continuous, discrete, or categorical value. For our example, the hyperparameters subject to tuning in this work are dropout rate, the number of neurons (units) in each layer, epoch, batch size, learning rate, and activation function. Those hyperparameters are defined in the search space H. Given that $h \in H$, a classifier can be trained on the training dataset, and its precision score is estimated based on the 3-fold cross-validation algorithm. D denotes the training dataset and $f(h, D)$ is our estimated precision score. This function is the objective function to optimize for hyperparameter selection. The problem of searching for the best hyperparameters is formulated as in Equation (43).

$$h^* = \arg\max_h (f(h, D) : h \in H) \tag{43}$$

Figure 6 describes the hyperparameter optimization process. First, the training set is extracted from the dataset. This part of the dataset contains 70% of the samples. In the next step, the training set was split into three folders containing the same number of transactions, and every folder was divided into parts for training and validation, respectively. The model is trained on the training part and evaluated using the validation parts based on the precision score. The returning measurements for the 3-fold cross-validation techniques are the average score. This average score is sent to the optimization algorithms. The Bayesian algorithm process uses this score to select the new hyperparameters. These hyperparameters are used for designing the new classifier architectures. This process continues until the number of iterations is completed. Finally, the outcome of this hyperparameter optimization process is the best hyperparameters obtained in the computation. The workflow of the hyperparameter optimization can be described mathematically as follows (Algorithm 4):

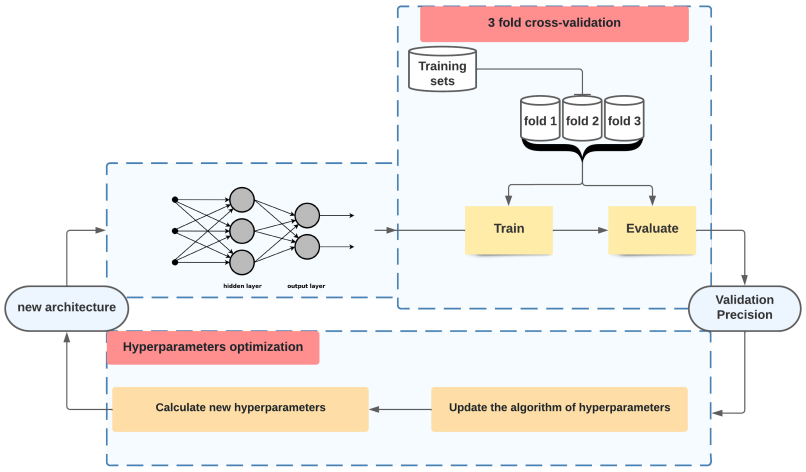

Figure 6. Hyperparameter optimization process workflow.

Algorithm 4 Pseudo-code of hyperparameter optimization with 3-fold cross-validation

Input: Objective function f, hyperparameter space H, testing set D^{test}, training set $D^{train} = \{(x_1, y_1), \cdots, (x_n, y_n)\}$, number of iterations T.
Select randomly a set of hyperparameters h
for $j = 1, \ldots, T$ **do**
    Partition D^{train} into D_1, D_2, D_3
    for $i = 1, \ldots, 3$ **do**
        $\hat{y}_i = f(D \setminus D_i, h)$
    end for
    $error(h) = \frac{1}{3} \sum_{i=1}^{3} Precision_{D_i}(\hat{y}_i, y_i)$
    update h
end for
Output: $h^* = argmin_h(error(h))$
$y_{h^*} = f(h^*, D^{test})$

Table 2 shows the deep learning hyperparameters used for optimization and their range. A brief description of each hyperparameter is given below.

Table 2. Search space for hyperparameter optimization.

Hyperparameter	Optimization Rate	Type
Activation function	$[ReLU, Sigmoid, Tanh]$	Categorical
Learning rate	$[0, 1.0]$	Continue
Dropout rate of layer 1	$[0.0, 0.6]$	Continue
Dropout rate of layer 2	$[0.0, 0.6]$	Continue
Batch size	$[1, 100]$	discrete
Epochs	$[1, 100]$	discrete
Number of neurons in layer 1	$[10, 30]$	discrete
Number of neurons in layer 2	$[10, 30]$	discrete
Number of LSTM Units in layer 1	$[10, 30]$	discrete
Number of LSTM Units in layer 2	$[10, 30]$	discrete
Number of RNN Units in layer 1	$[10, 30]$	discrete
Number of RNN Units in layer 2	$[10, 30]$	discrete

- Dropout is a regularization technique used to prevent overfitting in deep learning architectures and lead to accurate performance. Its mechanism is to remove some neurons from the layer. To fix this number, the user sets a dropout rate. This approach makes the training process noisy as a side effect.
- The learning rate hyperparameter's objective was to determine how much the model can adapt its parameters based on the estimated error. While a low learning rate leads to lengthy training processes, a large value leads to learning a suboptimal set of weights, resulting in an unstable learning process.
- The batch size is a gradient descent hyperparameter. It determines the number of training blocks to work through before updating the model parameters, where gradient descent is an iterative learning. The latter uses a training dataset to update the parameters of the model.
- The epoch is a hyperparameter of gradient descent that determines the number of complete passes through the training dataset.
- The activation function is implemented in an artificial neural network to learn and find intricate hidden patterns in the output dataset. So, the activation function can be seen as a controller for what information should be passed to the next neuron. It takes input data from the previous neurons and converts them into some form that can be the input of the next neuron. In this study, the activation hyperparameter takes three functions, namely ReLu (Equation (44)), Sigmoid (Equation (45)), and Tanh (Equation (46)). Those functions are described below.

$$Relu(x_i) = \max(0, x_i) \tag{44}$$

$$Sigmoid(x_i) = \frac{1}{1 + e^{x_i}} \tag{45}$$

$$Tanh(x_i) = \frac{e^{x_i} - e^{-x_i}}{e^{x_i} + e^{-x_i}} \tag{46}$$

5. Experimental Design

5.1. Model Architecture Design

This work proposes an intelligent solution based on a deep learning approach for detecting non-authorized transactions included in the European credit card dataset used for evaluation. This solution consists of four blocks. In the first block, a dataset is chosen. The next block is data preprocessing. After that, in the third block, the random undersampling method was implemented to address class imbalance issues. The fourth block implements hyperparameter optimization using 3-fold cross-validation. It calculates the performance of the hyperparameters seated in each iteration. The optimization was performed using Bayesian optimization. The main objective was to select the best-performing hyperparameters. Figure 7 describes the architecture and workflow of our solution.

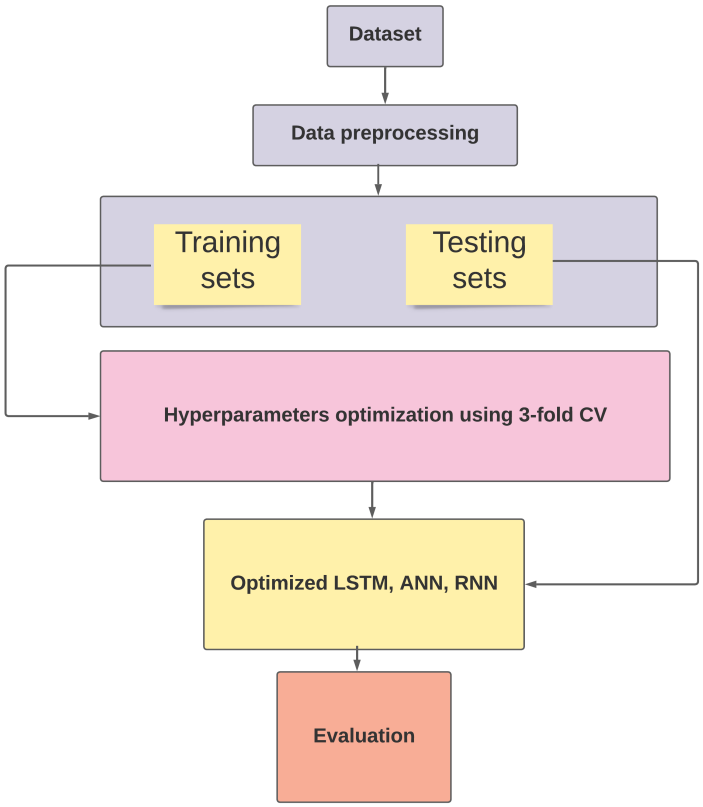

Figure 7. Proposed solution.

Algorithm 5 describes a methodology for deep learning-based fraud detection, tailored to the challenges of class imbalance and hyperparameter optimization. The following is a description of the algorithm:

Algorithm 5 Deep Learning-Based Fraud Detection

Input: Dataset D (European credit card dataset), Hyperparameter search space H, Number of iterations N, Performance metrics.
Input: Deep learning architecture (ANN, RNN, LSTM)
Initialize an empty list to store performance metrics: $performance\_list = []$
for $i = 1$ to N **do**
    Split dataset D into training and testing sets: $D_{train}, D_{test} = train\_test\_split(D)$
    Preprocess the training data: $D_{train\_preprocessed} = pre\_process(D_{train})$
    Implement random undersampling to address class imbalance: $D_{train\_balanced} = random\_undersampling(D_{train\_preprocessed})$
    for each hyperparameter combination h in H **do**
        Perform k-fold cross-validation (e.g., $k = 3$) on $D_{train\_balanced}$ using h: $performance\_metric = cross\_validate(D_{train\_balanced}, h)$
        Calculate the average performance metric P across folds for h
        Store h and its corresponding P in $performance\_list$
    end for
    Select the hyperparameter combination $h_{optimal}$ with the highest average P from $performance\_list$
    Train a model on the entire balanced training dataset $D_{train\_balanced}$ using $h_{optimal}$: $trained\_model = train\_model(D_{train\_balanced}, h_{optimal})$
    Evaluate the trained model on the testing set D_{test}: $test\_performance = evaluate\_model(trained\_model, D_{test})$
end for
Output: $test\_performance$

5.2. Statistical Measure

The selection of the evaluation metrics is a very important task for the best-performing model. In this subsection, we present the measurements used for evaluating our model. After the optimized deep learning architectures are tested on the testing set, the outputs are the following statistical values: TN (True Negative), TP (True positive), FN (False Negative), and FP (False Positive). These values are used to calculate the accuracy score (ACC), precision (PER), area under the curve (AUC), G-mean (GM), and sensitivity (SEN). Those evaluation metrics are formulated as follows:

1. The accuracy score refers to the ratio of correctly classified transaction packets (normal or abnormal) to total credit card transaction samples. It can be formulated mathematically as:
$$Accuracy(ACC) = \frac{TN + TP}{FP + TP + FN + TN} \tag{47}$$
2. Precision score: The ratio of correctly classified non-authorized transactions to the total number of identified fraud transactions. It can be calculated as:
$$Precision(PER) = \frac{TP}{FP + TP} \tag{48}$$
3. AUC: Receiver Operating Characteristics (ROC) Curve refers to the relationship between the false positive rate (FPR) on the x-axis and the true positive rate (TPR) on the y-axis. The Area Under the ROC Curve (AUC) is formulated as:
$$AUC = \int_0^1 \frac{TP}{TP + FN} d \frac{FP}{FP + TN} \tag{49}$$
4. G-Mean: This metric is used to measure the balance between fraud and the accuracy of the identification of legitimate transactions. Poor performance is indicated by a low G-Mean. This measure is important to avoid the model overfitting the normal transactions and underfitting the abnormal transactions.

$$G - Mean(GM) = \sqrt{\frac{TN \times TP}{(FP + TP)(TP + FP)}} \tag{50}$$

5. Sensitivity (recall) or the "true positive rate" refers to the number of fraudulent transactions that are correctly predicted. Its formula is as follows:

$$Sensitivity(\text{SEN}) = \frac{TP}{FN + TP} \tag{51}$$

6. Results and Discussion

The imbalanced European credit card dataset was transformed into a balanced dataset. This happened after removing several normal transactions from the majority class to test the three deep learning architectures with hyperparameter optimization using the Bayesian optimization approach. The random undersampling method with a sampling strategy is equal to 0.5. Moreover, the Bayesian optimization technique was used to design deep learning classifiers to improve the detection of fraudulent transactions. Several experiments were conducted with different numbers of iterations: 50, 70, and 100. Tables 3–5 present the performance metrics and execution times of three deep learning models ANN, LSTM, and RNN. For 50 iterations, the ANN model achieved an accuracy (ACC) of 0.8939, with perfect precision (PER) and a geometric mean (GM) of 0.8024. However, its sensitivity (SEN) was relatively low at 0.6439 compared to LSTM and RNN. The RNN model demonstrated the highest ACC and AUC (0.9593 and 0.9767, respectively) among all models. As the number of iterations increased to 70 and 100, all models showed improvements in ACC, PER, GM, and AUC. Particularly, the LSTM model consistently exhibited high performance across all metrics, with ACC exceeding 0.95 in both 70 and 100 iterations. In terms of execution time, the LSTM model consistently took longest, followed by RNN and ANN. However, the execution time increased with the number of iterations for all models, indicating a trade-off between computational resources and model performance. Overall, the results suggest that LSTM outperformed ANN and RNN in terms of both performance metrics and computational efficiency for credit card fraud detection. However, further analysis considering other factors such as interpretability and scalability is necessary to determine the most suitable model for real-world deployment in fraud detection systems.

Table 3. Results obtained using the Bayesian algorithm for 50 iterations.

Model	ACC	PER	GM	SEN	AUC
ANN	0.8939	1	0.8024	0.6439	0.8356
LSTM	0.7562	1	0.4264	0.1818	0.9291
RNN	0.9593	0.9596	0.9418	0.9015	0.9767

Table 3 shows the obtained results for the three classifiers using the Bayesian algorithm for hyperparameter optimization. The number of iterations is 50. From these results, the best accuracy score, 95.93%, is achieved by the RNN architecture, while the second-best score, 89.93%, is obtained by the ANN model. Moreover, the lowest score, 75.62%, is obtained using the LSTM model. Therefore, the RNN architecture outperforms the other deep learning models in classifying fraud from non-fraud transactions in the credit card dataset. Similarly, the best precision score is obtained with ANN and LSTM, at 1. As a consequence, the two models classify all fraudulent transactions correctly. In contrast, the RNN achieved a score of 95.96% for correctly classified fraud transactions. Taking the G-mean score into account, crucial when two classes have similar importance, this measure demonstrates the model's ability to distinguish between fraudulent and non-fraudulent transactions. The best G-mean score, 94.18%, is obtained with the RNN model, while the lowest score, 42.64%, is obtained with the LSTM model. In terms of sensitivity score, with the RNN, we obtained 90.15%, while (18.18%) was reached with the LSTM

model as the lowest score. Finally, 97.67% is the best AUC score obtained from the RNN architecture. As a result, RNN can be good at detecting fraud transactions based on Bayesian optimization for 50 iterations.

Table 4. Results obtained using the Bayesian algorithm for 70 iterations.

Model	ACC	PER	GM	SEN	AUC
ANN	0.9525	1	0.9170	0.8409	0.9207
LSTM	0.9548	0.9745	0.9288	0.8712	0.9372
RNN	0.9593	1	0.9293	0.8636	0.9752

The result of 70 iterations is shown in Table 4. It represents the resulting outcome of Bayesian optimization for hyperparameter tuning. These results describe the scores produced by ANN, LSTM, and RNN architectures. The accuracy score for the three classifiers is 95%, indicating that 95% of the transactions are correctly classified. Similarly, we obtained a score of 1 for the precision of LSTM and ANN. The lowest score is achieved by RNN, which correctly classifies 97% of fraud transactions. Furthermore, 92% as the best G-Mean score is achieved by LSTM and RNN. Similarly, we obtained the best sensitivity (87.12%) using the LSTM model and the best AUC using the RNN architecture (97.52%).

Table 5. Results obtained using the Bayesian algorithm for 100 iterations.

Model	ACC	PER	GM	SEN	AUC
ANN	0.9548	0.9827	0.9263	0.8636	0.9323
LSTM	0.9571	1	0.9252	0.8560	0.9641
RNN	0.9593	1	0.9293	0.8636	0.9752

Table 5 presents the results obtained for the three models with hyperparameter optimization based on Bayesian optimization for 100 iterations. From these results, we obtained the same accuracy score for the three classifiers, in which 95% of transactions are correctly identified. Furthermore, the best precision score is achieved for both the LSTM and ANN classifiers. On the contrary, ANN obtained the lowest score, at 98% of fraud transactions correctly classified. In terms of G-Mean, we obtained the same score for the three classifiers (92%). Likewise, the same sensitivity is achieved by the three classifiers (86%). Furthermore, the best AUC score was from RNN (97%).

Table 6 shows the computational time for the hyperparameter process. From this table, we notice that the best computational time is that of the ANN architecture, followed by the RNN architecture as the second-best time for hyperparameter searching. So, ANN is faster than RNN, which is faster than LSTM. However, the findings presented in the results table shed light on the impact of hyperparameter optimization iterations on the performance of the three distinct models, ANN, LSTM, and RNN, within the context of our research. Notably, the results showcase varying patterns of performance improvement across different models and iteration counts. For instance, the ANN model demonstrates steady performance gains with each increase in the number of iterations, suggesting a positive correlation between iteration count and model performance. Conversely, the LSTM model exhibits substantial performance improvements as the number of iterations increases, albeit with a slight decline in performance observed at 100 iterations. This possibly indicates the onset of diminishing returns or overfitting. Meanwhile, the RNN model displays notable performance enhancements up to 70 iterations, beyond which further iterations do not yield significant gains, suggesting a potential saturation point in performance improvement. These nuanced insights underscore the importance of carefully optimizing hyperparameters to maximize model performance, while also highlighting the need for monitoring performance trends to avoid potential pitfalls such as overfitting. Overall, these results contribute valuable insights to the field of machine learning and underscore the importance of iterative optimization processes in model development and deployment.

Table 6. Execution time results for hyperparameter optimization for different iterations.

Model	50 Iterations	70 Iterations	100 Iterations
ANN	604.66	737.03	1045.28
LSTM	1395.57	2761.79	2603.52
RNN	1039.28	1934.66	1934.66

Figures 8–10 support the results discussed in Tables 3–5. The hyperparameter process using RNN is the best choice for detecting fraud in transactions. This approach is more efficient than LSTM and ANN.

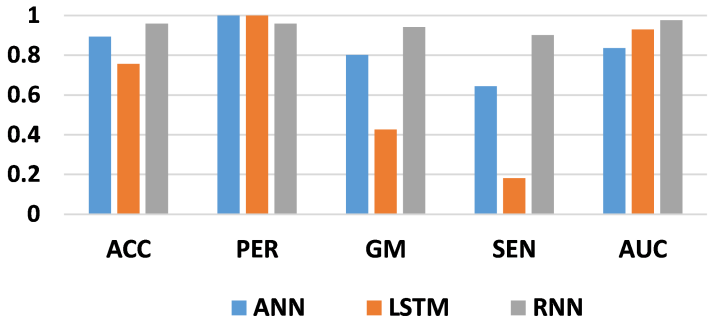

Figure 8. Results of 50 iterations of optimizing the hyperparameters for three deep learning architectures.

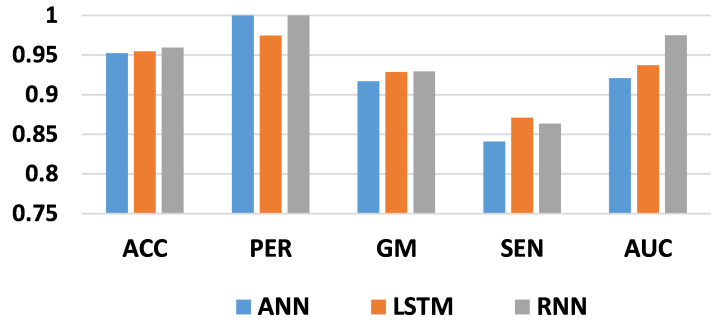

Figure 9. Results of 70 iterations of optimizing the hyperparameters for three deep learning architectures.

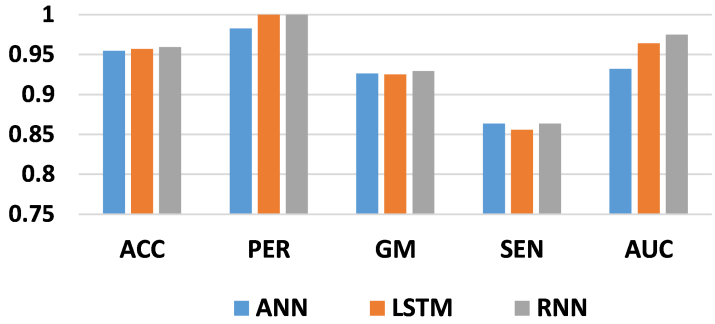

Figure 10. Results of 100 iterations of optimizing the hyperparameters for three deep learning architectures.

As Figure 11 shows, the hyperparameter optimization process of our system configuration requires a substantial amount of time due to the iterative nature of the loop. Typically, the time required for hyperparameter optimization can vary depending on factors such as the complexity of the dataset and the number of hyperparameters being tuned. In our experiments, this optimization process may take several hours to complete, as it involves iteratively evaluating the performance of different hyperparameter configurations. As for the frequency of optimization, it is crucial to note that this process is not a one-time task. Instead, it may need to be executed whenever there is a significant change in the input

dataset or when the performance metrics, such as accuracy, begin to degrade. By periodically re-optimizing the hyperparameters, we ensure that our model remains well-tuned and adaptive to changes in the data distribution or underlying patterns.

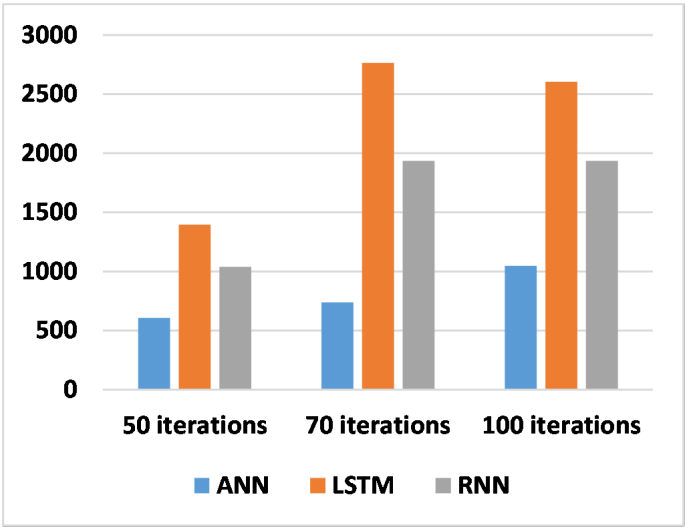

Figure 11. Results of the comparison of hyperparameter optimization execution times.

The AUC curve is an efficient and significant estimation of overall performance. It is a general measure of the accuracy of fraudulent transactions. Moreover, a higher AUC curve indicates better prediction performance. Figures 12–14 present the AUC curves of the Bayesian algorithm based on hyperparameter optimization for 50, 70, and 100 iterations, respectively, and the three deep learning architectures. The figures confirm the results presented in Tables 3–5. The AUC curve of the RNN architecture (orange line) is located closest to the figure's top-left corner, suggesting that this architecture's hyperparameter optimization is superior to ANN and LSTM techniques for credit card fraud identification. Another comparison is conducted based on the precision and recall curves. This graph plots the recall score on the x-axis and the precision on the y-axis. The precision–recall curve provides a full picture of the classification performance and is stable even in imbalanced datasets.

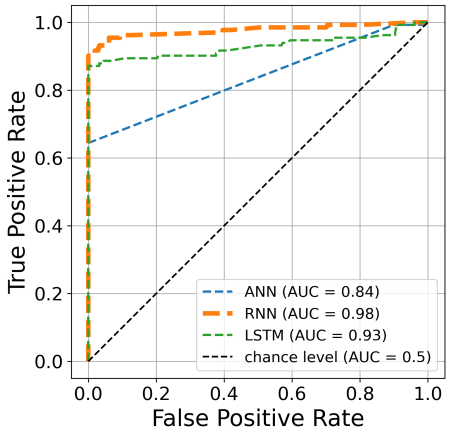

Figure 12. The ROC curve constructed after 50 iterations of optimizing hyperparameters using Bayesian algorithm.

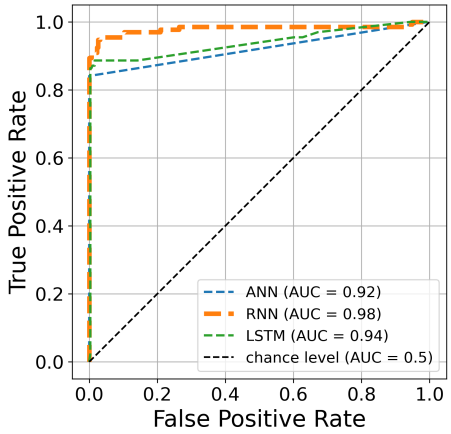

Figure 13. The ROC curve constructed after 70 iterations of optimizing hyperparameters using Bayesian algorithm.

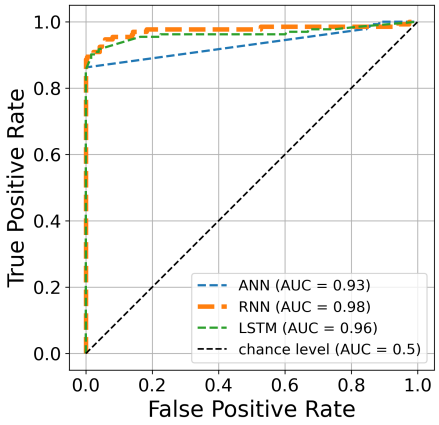

Figure 14. The ROC curve constructed after 100 iterations of optimizing hyperparameters using Bayesian algorithm.

Figures 15–17 clearly show visualizations of the precision–recall plot of the three deep learning architectures based on Bayesian optimization for 50, 70, and 100 iterations, respectively. The precision–recall curve of the RNN approach (orange line) is located closest to the upper right corners of the figures, proving that the RNN architectures for credit card fraud detection achieved better performance for this dataset.

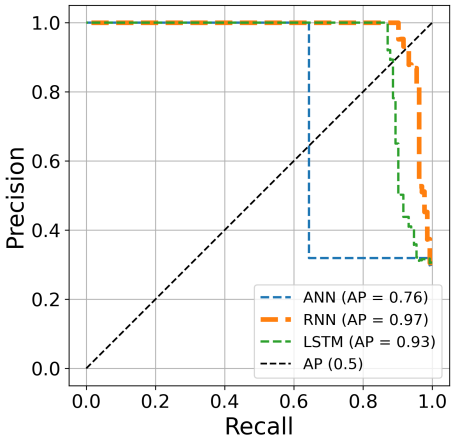

Figure 15. The precision–recall curve constructed after 50 iterations of optimizing hyperparameters using Bayesian algorithm.

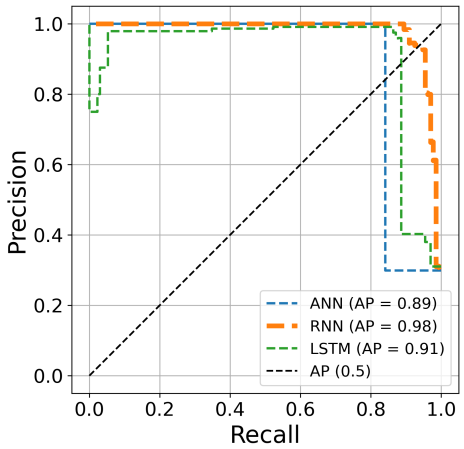

Figure 16. The precision–recall curve constructed after 70 iterations of optimizing hyperparameters using Bayesian algorithm.

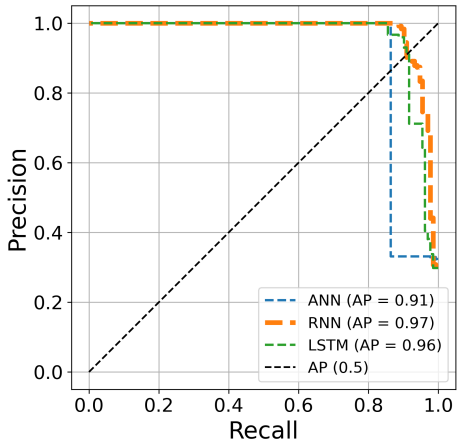

Figure 17. The precision–recall curve constructed after 100 iterations of optimizing hyperparameters using Bayesian algorithm.

7. Conclusions

This paper focuses on credit card fraud identification employing three distinct deep learning architectures: ANN, LSTM, and RNN. Utilizing the European credit card dataset, Bayesian optimization is employed for hyperparameter tuning. This study targets critical parameters such as epoch, batch size, activation function, learning rate, number of neurons per layer, number of LSTM units in each layer, and number of RNN units in each layer. Through Bayesian optimization, these hyperparameters are systematically tuned across 50, 70, and 100 iterations, leveraging a randomly undersampled dataset for computation. The experimental results exhibit RNN's consistent dominance across multiple key performance metrics. Particularly noteworthy is RNN's ability to achieve the highest accuracy (ACC) and area under the curve (AUC) scores, consistently maintaining values of 0.9593 across all iterations. Moreover, RNN demonstrates superior precision (PER) and sensitivity (SEN) compared to ANN and LSTM, with perfect precision scores (1.0) achieved in certain iterations and sensitivity scores ranging from 0.9015 to 0.8636. The geometric mean (GM) scores further reinforce RNN's superiority, consistently surpassing ANN and LSTM. However, it is worth mentioning that LSTM exhibits competitive performance, especially in terms of accuracy and AUC. In addition to performance, computational efficiency is considered, with ANN emerging as the fastest architecture in terms of execution time, followed by RNN and LSTM. These findings collectively suggest RNN's potential as an effective tool for credit card fraud detection tasks. Nonetheless, LSTM and ANN remain viable alternatives depending on specific computational resources and accuracy thresh-

olds. For future work, distributed hyperparameter optimization will be explored using the Apache Spark platform. This approach will involve comparing Spark with regular implementation, utilizing varying numbers of workers for distributed hyperparameter tuning across RNN, ANN, and LSTM architectures. Furthermore, the study will expand to include evaluation on multiple datasets and address imbalanced data challenges through synthetic oversampling methods, with the goal of implementing these approaches on the Databricks platform.

Author Contributions: Conceptualization, S.E.K.; Methodology, S.E.K. and M.T.; Software, M.T.; Validation, S.E.K.; Formal analysis, S.E.K., M.T. and H.S.; Investigation, S.E.K. and H.S.; Resources, M.T.; Data curation, M.T.; Writing—original draft, M.T.; Writing—review and editing, S.E.K., M.T. and H.S.; Visualization, M.T.; Supervision, S.E.K.; Project administration, S.E.K.; Funding acquisition, M.T. All authors have read and agreed to the pubhlished version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The paper uses a European dataset to test the efficiency of the algorithms. This dataset is publicly available at https://www.kaggle.com/mlg-ulb/creditcardfraud (accessed on 26 December 2023) and is free of charge.

Acknowledgments: The authors thank the anonymous reviewers for their valuable comments, which have helped us to considerably improve the content, quality, and presentation of this article. The researchers would also like to thank Abderrahim Chalfaouat for reviewing the article's language.

Conflicts of Interest: The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Abakarim, Y.; Lahby, M.; Attioui, A. An efficient real time model for credit card fraud detection based on deep learning. In Proceedings of the 12th International Conference on Intelligent Systems: Theories and Applications, Rabat, Morocco, 24–25 October 2018; pp. 1–7.
2. Arora, V.; Leekha, R.S.; Lee, K.; Kataria, A. Facilitating user authorization from imbalanced data logs of credit cards using artificial intelligence. *Mob. Inf. Syst.* **2020**, *2020*, 8885269. [CrossRef]
3. Błaszczyński, J.; de Almeida Filho, A.T.; Matuszyk, A.; Szeląg, M.; Słowiński, R. Auto loan fraud detection using dominance-based rough set approach versus machine learning methods. *Expert Syst. Appl.* **2021**, *163*, 113740. [CrossRef]
4. Branco, B.; Abreu, P.; Gomes, A.S.; Almeida, M.S.; Ascensão, J.T.; Bizarro, P. Interleaved sequence RNNs for fraud detection. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Virtual, 6–10 July 2020; pp. 3101–3109.
5. Ahmed, W.; Rasool, A.; Javed, A.R.; Kumar, N.; Gadekallu, T.R.; Jalil, Z.; Kryvinska, N. Security in next generation mobile payment systems: A comprehensive survey. *IEEE Access* **2021**, *9*, 115932–115950. [CrossRef]
6. Dornadula, V.N.; Geetha, S. Credit card fraud detection using machine learning algorithms. *Procedia Comput. Sci.* **2019**, *165*, 631–641. [CrossRef]
7. Tayebi, M.; El Kafhali, S. A weighted average ensemble learning based on the cuckoo search algorithm for fraud transactions detection. In Proceedings of the 2023 14th International Conference on Intelligent Systems: Theories and Applications (SITA), Casablanca, Morocco, 22–23 November 2023; pp. 1–6.
8. Fang, Y.; Zhang, Y.; Huang, C. Credit Card Fraud Detection Based on Machine Learning. *Comput. Mater. Contin.* **2019**, *61*, 185–195. [CrossRef]
9. Forough, J.; Momtazi, S. Ensemble of deep sequential models for credit card fraud detection. *Appl. Soft Comput.* **2021**, *99*, 106883. [CrossRef]
10. Hu, X.; Chen, H.; Zhang, R. Short paper: Credit card fraud detection using LightGBM with asymmetric error control. In Proceedings of the 2019 Second International Conference on Artificial Intelligence for Industries (AI4I), Laguna Hills, CA, USA, 25–27 September 2019; pp. 91–94.
11. Kousika, N.; Vishali, G.; Sunandhana, S.; Vijay, M.A. Machine learning based fraud analysis and detection system. *J. Phys. Conf. Ser.* **2021**, *1916*, 012115. [CrossRef]

12. Tan, G.W.H.; Ooi, K.B.; Chong, S.C.; Hew, T.S. NFC mobile credit card: The next frontier of mobile payment? *Telemat. Inform.* **2014**, *31*, 292–307. [CrossRef]
13. Alarfaj, F.K.; Malik, I.; Khan, H.U.; Almusallam, N.; Ramzan, M.; Ahmed, M. Credit Card Fraud Detection Using State-of-the-Art Machine Learning and Deep Learning Algorithms. *IEEE Access* **2022**, *10*, 39700–39715. [CrossRef]
14. Carcillo, F.; Dal Pozzolo, A.; Le Borgne, Y.A.; Caelen, O.; Mazzer, Y.; Bontempi, G. Scarff: A scalable framework for streaming credit card fraud detection with spark. *Inf. Fusion* **2018**, *41*, 182–194. [CrossRef]
15. Wei, W.; Li, J.; Cao, L.; Ou, Y.; Chen, J. Effective detection of sophisticated online banking fraud on extremely imbalanced data. *World Wide Web* **2013**, *16*, 449–475. [CrossRef]
16. Kim, J.; Kim, H.J.; Kim, H. Fraud detection for job placement using hierarchical clusters-based deep neural networks. *Appl. Intell.* **2019**, *49*, 2842–2861. [CrossRef]
17. El Kafhali, S.; Tayebi, M. XGBoost based solutions for detecting fraudulent credit card transactions. In Proceedings of the 2022 International Conference on Advanced Creative Networks and Intelligent Systems (ICACNIS), Bandung, Indonesia, 23 November 2022; pp. 1–6.
18. Hajek, P.; Abedin, M.Z.; Sivarajah, U. Fraud detection in mobile payment systems using an XGBoost-based framework. *Inf. Syst. Front.* **2023**, *25*, 1985–2003. [CrossRef] [PubMed]
19. Seera, M.; Lim, C.P.; Kumar, A.; Dhamotharan, L.; Tan, K.H. An intelligent payment card fraud detection system. *Ann. Oper. Res.* **2024**, *334*, 445–467. [CrossRef]
20. Van Belle, R.; Baesens, B.; De Weerdt, J. CATCHM: A novel network-based credit card fraud detection method using node representation learning. *Decis. Support Syst.* **2023**, *164*, 113866. [CrossRef]
21. Jha, S.; Guillen, M.; Westland, J.C. Employing transaction aggregation strategy to detect credit card fraud. *Expert Syst. Appl.* **2012**, *39*, 12650–12657. [CrossRef]
22. Tayebi, M.; El Kafhali, S. Credit Card Fraud Detection Based on Hyperparameters Optimization Using the Differential Evolution. *Int. J. Inf. Secur. Priv. (IJISP)* **2022**, *16*, 1–21. [CrossRef]
23. Mathew, A.; Amudha, P.; Sivakumari, S. Deep learning techniques: An overview. In Proceedings of the International Conference on Advanced Machine Learning Technologies and Applications, Jaipur, India, 13–15 February 2020; Springer: Singapore, 2021; pp. 599–608.
24. Salloum, S.A.; Alshurideh, M.; Elnagar, A.; Shaalan, K. Machine learning and deep learning techniques for cybersecurity: A review. In Proceedings of the International Conference on Artificial Intelligence and Computer Vision, Cairo, Egypt, 8–9 April 2020; Springer: Cham, Switzerland, 2020; pp. 50–57.
25. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef] [PubMed]
26. Li, Z.; Pan, Z.; Li, Y.; Yang, X.; Geng, C.; Li, X. Advanced root mean square propagation with the warm-up algorithm for fiber coupling. *Opt. Express* **2023**, *31*, 23974–23989. [CrossRef]
27. Sherstinsky, A. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Phys. D Nonlinear Phenom.* **2020**, *404*, 132306. [CrossRef]
28. Wang, C.; Niepert, M. State-regularized recurrent neural networks. In Proceedings of the International Conference on Machine Learning. PMLR, Long Beach, CA, USA, 9–15 June 2019; pp. 6596–6606.
29. Yu, Y.; Si, X.; Hu, C.; Zhang, J. A review of recurrent neural networks: LSTM cells and network architectures. *Neural Comput.* **2019**, *31*, 1235–1270. [CrossRef] [PubMed]
30. Smagulova, K.; James, A.P. A survey on LSTM memristive neural network architectures and applications. *Eur. Phys. J. Spec. Top.* **2019**, *228*, 2313–2324. [CrossRef]
31. Garnett, R. *Bayesian Optimization*; Cambridge University Press: Cambridge, UK, 2023.
32. Swinburne, R. Bayes' Theorem. *Rev. Philos. Fr. L'etranger* **2004**, *194*, 250–251.
33. Dunlop, M.M.; Girolami, M.A.; Stuart, A.M.; Teckentrup, A.L. How deep are deep Gaussian processes? *J. Mach. Learn. Res.* **2018**, *19*, 1–46.
34. Wu, J.; Chen, X.Y.; Zhang, H.; Xiong, L.D.; Lei, H.; Deng, S.H. Hyperparameter optimization for machine learning models based on Bayesian optimization. *J. Electron. Sci. Technol.* **2019**, *17*, 26–40.
35. Credit Card Fraud Dataset. 2023. Available online: https://www.kaggle.com/mlg-ulb/creditcardfraud/data (accessed on 26 December 2023).
36. Abdi, H.; Williams, L.J. Principal component analysis. *Wiley Interdiscip. Rev. Comput. Stat.* **2010**, *2*, 433–459. [CrossRef]
37. El Kafhali, S.; Tayebi, M. Generative adversarial neural networks based oversampling technique for imbalanced credit card dataset. In Proceedings of the 2022 6th SLAAI International Conference on Artificial Intelligence (SLAAI-ICAI), Colombo, Sri Lanka, 1–2 December 2022; pp. 1–5.
38. Hordri, N.F.; Yuhaniz, S.S.; Azmi, N.F.M.; Shamsuddin, S.M. Handling class imbalance in credit card fraud using resampling methods. *Int. J. Adv. Comput. Sci. Appl.* **2018**, *9*, 390–396. [CrossRef]
39. Feurer, M.; Hutter, F. Hyperparameter optimization. In *Automated Machine Learning: Methods, Systems, Challenges*; Springer: Cham, Switzerland, 2019; pp. 3–33.

40. Tayebi, M.; El Kafhali, S. Hyperparameter optimization using genetic algorithms to detect frauds transactions. In Proceedings of the International Conference on Artificial Intelligence and Computer Vision, Settat, Morocco, 28–30 June 2021; Springer: Cham, Switzerland, 2021; pp. 288–297.
41. Tayebi, M.; El Kafhali, S. Performance analysis of metaheuristics based hyperparameters optimization for fraud transactions detection. *Evol. Intell.* **2024**, *17*, 921–939. [CrossRef]