mathematics

MDPI

Article

Power Transformer Fault Diagnosis Using Neural Network Optimization Techniques

Vasiliki Rokani [1,*], Stavros D. Kaminaris [1], Petros Karaisas [1] and Dimitrios Kaminaris [2]

[1] Department of Electrical and Electronics Engineering, University of West Attica, GR-12244 Egaleo, Greece; skamin@uniwa.gr (S.D.K.); karaisas@uniwa.gr (P.K.)
[2] Institute of Physics, Ecole Polytechnique Federale de Lausanne (EPFL), 1015 Lausanne, Switzerland; dimitrios.kaminaris@epfl.ch
* Correspondence: vrokani@uniwa.gr

Abstract: Artificial Intelligence (AI) techniques are considered the most advanced approaches for diagnosing faults in power transformers. Dissolved Gas Analysis (DGA) is the conventional approach widely adopted for diagnosing incipient faults in power transformers. The IEC-599 standard Ratio Method is an accurate method that evaluates the DGA. All the classical approaches have limitations because they cannot diagnose all faults accurately. Precisely diagnosing defects in power transformers is a significant challenge due to their extensive quantity and dispersed placement within the power network. To deal with this concern and to improve the reliability and precision of fault diagnosis, different Artificial Intelligence techniques are presented. In this manuscript, an artificial neural network (ANN) is implemented to enhance the accuracy of the Rogers Ratio Method. On the other hand, it should be noted that the complexity of an ANN demands a large amount of storage and computing power. In order to address this issue, an optimization technique is implemented with the objective of maximizing the accuracy and minimizing the architectural complexity of an ANN. All the procedures are simulated using the MATLAB R2023a software. Firstly, the authors choose the most effective classification model by automatically training five classifiers in the Classification Learner app (CLA). After selecting the artificial neural network (ANN) as the sufficient classification model, we trained 30 ANNs with different parameters and determined the 5 models with the best accuracy. We then tested these five ANNs using the Experiment Manager app and ultimately selected the ANN with the best performance. The network structure is determined to consist of three layers, taking into consideration both diagnostic accuracy and computing efficiency. Ultimately, a (100-50-5) layered ANN was selected to optimize its hyperparameters. As a result, following the implementation of the optimization techniques, the suggested ANN exhibited a high level of accuracy, up to 90.7%. The conclusion of the proposed model indicates that the optimization of hyperparameters and the increase in the number of data samples enhance the accuracy while minimizing the complexity of the ANN. The optimized ANN is simulated and tested in MATLAB R2023a—Deep Network Designer, resulting in an accuracy of almost 90%. Moreover, compared to the Rogers Ratio Method, which exhibits an accuracy rate of just 63.3%, this approach successfully addresses the constraints associated with the conventional Rogers Ratio Method. So, the ANN has evolved a supremacy diagnostic method in the realm of power transformer fault diagnosis.

Keywords: power transformers; fault diagnosis; DGA; ANN; MATLAB; neural network optimization

MSC: 68T07

check for updates

Citation: Rokani, V.; Kaminaris, S.D.; Karaisas, P.; Kaminaris, D. Power Transformer Fault Diagnosis Using Neural Network Optimization Techniques. *Mathematics* **2023**, *11*, 4693. https://doi.org/10.3390/math11224693

Academic Editor: Nicu Bizon

Received: 3 October 2023
Revised: 15 November 2023
Accepted: 17 November 2023
Published: 19 November 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

A power transformer (PT) is a device that transmits power energy through circuits via electromagnetic induction. It is an integral component of the electrical power system (EPS) and has the effect of increasing or decreasing the voltage of an alternating current

power supply. Power transformers are essential in ensuring electrical energy is efficiently and reliably transmitted over long distances. The fundamental idea behind transformer theory is the utilization of a magnetic field generated by one coil to create an electromotive force (EMF) in a second coil [1]. The electrical apparatus comprises windings made of either copper or aluminum, a core composed of thin sheets of magnetic steel, and insulating materials such as high-density paper and mineral oil [2]. The transformer is a very intricate piece of equipment. Numerous forces are at play inside the tank, including phenomena such as ageing, chemical processes, electric and magnetic fields, thermal expansion and contraction, fluctuations in load, and the force of gravity. The transformer is subject to several external factors, such as through-faults, significant ambient temperature variations, voltage surges, and other forces like gravity and the Earth's magnetic field. Transformers are susceptible to a diverse range of problems [3].

The categorization of faults is as follows:

- Electrical: Partial Discharge, Corona, Arcing, Oil's Breakdown Voltage;
- Thermal: Cellulose Overheating, Oil Overheating;
- Mechanical: Winding, Core Deformations.

The transformer transmits high voltage levels that create a strong electrical field. This field causes strain on the insulation, and the rate at which it breaks down depends on the current insulation and the surrounding situations. The assessment of the insulation properties has significant implications for the longevity of a PT since it is strongly correlated with the durability of the insulation material. Several important characteristics may be assessed in relation to the insulating components of PTs. The frequently used methods are Dissolved Gas Analysis (DGA) in the insulating oil, assessment of insulation resistance, evaluation of partial discharges, and measurement of the power factor or tangent delta [2,3]. The DGA technique is a highly used approach in contemporary monitoring systems [4]. The literature has presented many methodologies for evaluating the DGA method, including the Key Gas, IEC ratio, Duval triangle, Doernenburg ratio, Rogers Ratio, and logarithmic nomograph methods. The fault classification techniques use reference tables and charts that have been developed based on the quantities or specific ratios of gases [5,6]. The issue authors have to cope with is that conventional DGA has limits because it mainly relies on empirical approaches and lacks mathematical formulations, hindering their ability to analyze all types of faults effectively. Consequently, in several instances, an inaccurate or unresolved diagnosis is seen. This occurs when more than one fault arises in a transformer or when the concentration of gases is near the threshold. In order to address this issue and improve the dependability of defect detection, the current research initiative employs an artificial neural network (ANN) [7,8].

After conducting a literature review on power transformer fault diagnosis (PTFD) employing DGA and artificial neural networks (ANNs), several research gaps and areas for further investigation were identified [7]. From our point of view, the most crucial research gap could be the limited study of model optimization techniques. Multiple studies concentrate on applying ANNs to PTFD but often skip the specifics of optimizing the ANN architecture and training process to improve the diagnostic precision. These issues can be addressed using optimization algorithms, regularization techniques, and hyperparameter tuning; all these will be further defined in Section 2.

To enhance the diagnostic precision of traditional DGA, an artificial neural network (ANN) was developed to improve the diagnostic model's resilience and accuracy. Moreover, it was selected to combine the conventional DGA technique, Roger's Ratio Method, with an ANN, as this method is widely used in the literature and gives accurate predictions for power transformer faults [7,8]. Furthermore, an optimization procedure is conducted to maximize the accuracy and decrease the architectural complexity of the proposed network. The optimization procedure is executed by adjusting the model's hyperparameters. The available literature on hyperparameter tuning is limited, with Machine Learning (ML) algorithm developers only supplying brief descriptions of their functions [9]. While scientific papers and online tutorials offer some additional information, there is a lack of

more extensive studies that investigate significant inquiries, such as the extent to which a model can be enhanced using tuning, the most effective tuning strategy, and the influence of tuning a specific hyperparameter on different datasets. More research on these topics would be beneficial in order to establish the best practices for hyperparameter tuning. This study aims to address the research concerns previously indicated within the realm of PTFD. Our goals are twofold:

1. Maximizing the Accuracy for PTFD.
2. Minimizing the Architectural Complexity of the ANN.

The present publication utilizes the MATLAB R2023a software to construct an optimized Multi-Layer Feedforward Backpropagation Neural Network. The experiment is conducted as follows: We train 30 NNs and then choose the top 5 models based on the validation accuracy. In order to determine which of these five ANNs was the most effective, we put them through a series of tests using the Experiment Management app.

Finally, we pick the optimal ANN to optimize its hyperparameters and, decisively, to provide the model with the greatest accuracy. Three different automated methods (a Bayesian optimizer, a random search, and a grid search), as well as three different optimizers (Adam, SGDM, and RMSprop), are compared and contrasted to assist in the hyperparameter selection for our ANN.

The performance and simulation of the optimized ANN are achieved by using MATLAB R2023a software. The accuracy of the conventional Rogers Method and the ANN method is calculated using the type (A) = (Total Samples Correctly Classified)/(Total Samples).

2. Materials and Methods

2.1. Dissolved Gas Analysis

The use of DGA involves the utilization of oil samples extracted from an operational PT to monitor the state of the transformer. DGA is an effective diagnostic tool used to identify early stage problems in oil-filled transformers far in advance of their progression into significant failures [10,11]. The insulation system used in an oil-filled transformer comprises mineral oil as the liquid insulating medium and paper as the solid insulating material [12]. The dependability of power transformers is contingent upon the efficiency and effectiveness of the insulating system to endure several stressors, including thermal, mechanical, and electrical ones [4,9].

The evolution of these stresses pertains to the chemical degradation of the dielectric insulation's oil or cellulose molecules. The primary byproducts of degradation consist of gaseous compounds that exhibit solubility in the oil matrix, hence enabling their detection using Gas Chromatography (GC). GC is a widely used method utilized to separate, identify, and quantify gas mixtures [7,10]. The examination of these gaseous substances aids in the identification of early-stage defect categories [13,14]. The DGA methodology encompasses the processes of gas detection, quantification, and characterization [11,14].

The Rogers Ratio Method

The Rogers Ratio Method involves the examination of four gases that are produced in the oil of a PT, serving as indications of faults [8]. The chemical compounds hydrogen (H_2), methane (CH_4), ethane (C_2H_6), ethylene (C_2H_4), and acetylene (C_2H_2) are of interest in this context [13,14].

Four gas ratios are derived from the concentrations calculated in parts per million (ppm). The gas ratios MH, EM, EE, and AE are expressed as the following:

- MH = CH_4/H_2
- EM = C_2H_6/CH_4
- EE = C_2H_4/C_2H_6
- AE = C_2H_2/C_2H_4

Each of these gas ratios is separated into ranges based on their respective values.

PTFD is accomplished via the use of a coding system that is developed from the boundaries of gas concentrations, expressed in ppm, as given in Table 1. Table 2 provides the gas ratio code for each fault type, indicating the presence of 12 distinct fault categories [8,13].

Table 1. Rogers Ratio code [8,13].

Ratio Code	Range	Code
MH (CH_4/H_2)	$x < 0.1$	5
	$0.1 \leq x \leq 1.0$	0
	$1.0 \leq x \leq 3.0$	1
	$x > 3.0$	2
EM (C_2H_6/CH_4)	$x < 1.0$	0
	$x \geq 1.0$	1
EE (C_2H_4/C_2H_6)	$x < 1.0$	0
	$1.0 \leq x \leq 3.0$	1
	$x > 3.0$	2
AE (C_2H_2/C_2H_4)	$x < 0.1$	0
	$0.1 \leq x \leq 3.0$	1
	$x > 3.0$	2

Table 2. Fault types according to the Rogers Ratio Method [8,13].

No.	CH_4/H_2	C_2H_6/CH_4	C_2H_4/C_2H_6	C_2H_2/C_2H_4	Fault Type
1	0	0	0	0	No Fault
2	1–2	0	0	0	<150 °C Thermal Fault
3	1–2	1	0	0	150–200 °C Thermal Fault
4	0	1	0	0	200–300 °C Thermal Fault
5	0	0	1	0	General Conductor Overheating
6	1	0	1	0	Winding Circulating Currents
7	1	0	2	0	Core and Tank Circulating Currents Overheated Joints
8	5	0	0	0	Partial Discharge
9	5	0	0	1–2	Partial Discharge with Tracking
10	0	0	0	1	Flashover Without Power Follow-Through
11	0	0	1–2	1–2	Arc with Power Follow-Through
12	0	0	2	2	Continuous Sparking to Floating Potential

The 12 different types of faults are classified into five categories (F1, F2, F3, F4, F5) in order to obtain only five outputs for the ANN. A reduced number of outputs makes the network more functional and flexible in the MATLAB R2023a software. The numbers of fault types according to the Rogers Ratio Method are included in parentheses.

The five different types of faults are:

- F1: No-Fault (1);
- F2: Low Energy Discharge (2), (12);
- F3: High Energy Discharge (9), (10), (11);
- F4: Low and Medium Thermal Faults (3), (4), (5), (7), (8);
- F5: High-Temperature Thermal Faults (6).

The data samples and transformer states were created utilizing the IEEE DGA datasets [15] and the dataset from our previous work [8].

2.2. Artificial Neural Networks

Transformer fault estimation is a complex non-linear mapping problem due to the fact that inputs and outputs are variables with no linear relationship [16,17]. Based on the extant literature, this particular network configuration is widely used [18,19]. The ANN is used to determine the transformer's faults due to its precise and sufficient performance in

classification issues. However, the ANN is prone to constraints such as weak convergence and local optima. An optimization method is proposed to overcome these issues, maximize the accuracy, and decrease the architectural complexity. According to this method, an intensive search is performed for the finest training algorithm and its hyperparameters, including the neurons, learning rate, activation functions, L2 regularization factor, and momentum, that may provide the desired results.

In our prior research documented in article [9], a comprehensive approach was introduced for assessing the efficiency of DGA using artificial neural networks (ANNs). The four gas ratios are the inputs for the ANN, and the five transformer's incipient faults are the output targets. Due to the NN's central role in this paper, a brief overview of the approach and the primary description and influencing parameters is crucial.

ANNs are computational models comprising connected nodes, or "neurons," that operate together to process and explore information. These interconnected neurons allow neural networks to tackle various tasks, from natural language processing to image and speech recognition, and even autonomous driving. The ANN scheme is presented in Figure 1 below:

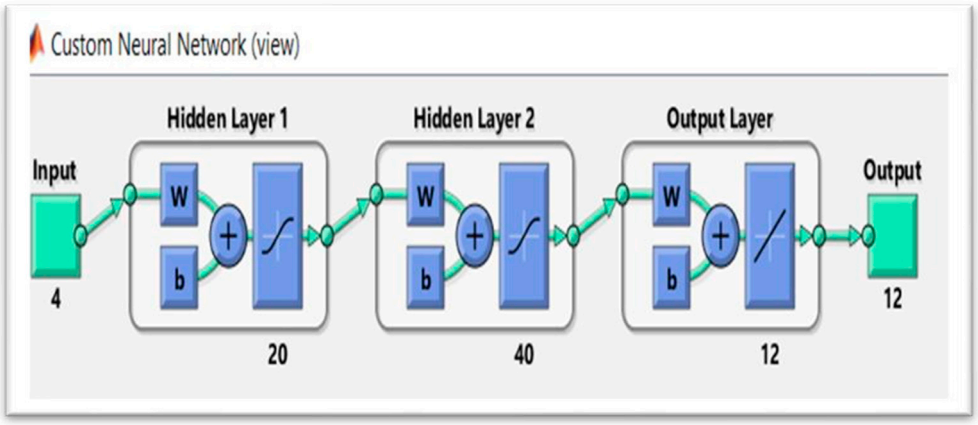


Figure 1. Scheme of an artificial neural network.

Backpropagation

The multi-layer network should be trained by operating a backpropagation algorithm, which is also used as the learning rule of deep learning [20]. The method of backpropagation allows the network to obtain knowledge from the training data by modifying the weights of the neurons according to the error or loss experienced during the training process. This method can be explained as follows [21,22]:

Forward-propagation: For each neuron in layer l, the weighted sum z is computed as:

$$z = W_l * a_{prev} + b_l \quad (1)$$

where W_l represents the weight matrix, a_{prev} stands for the activation function from the previous layer, and b_l is the bias vector for layer l. Then, the output a of each neuron is obtained by applying an activation function f, $a = f(z)$.

- Loss function: The choice of the loss function depends on the specific problem. If we denote the predicted output of the ANN as $\hat{y}$ and the target output as y, then the loss function L measures the discrepancy between $\hat{y}$ and y, such as the mean squared error for regression or cross-entropy for classification.

Backpropagation: For each neuron in the output layer, the partial derivative of the loss function with respect to its output is computed:

$$\delta_{out} = \frac{\partial L}{\partial \hat{y}} \quad (2)$$

Then, the gradient of the neuron's output with respect to its input is computed by multiplying δ_{out} by the derivative of the activation function $f'(z)$:

$$\delta_{out} = \delta_{out} * f(z) \tag{3}$$

Weight update: The weights and biases are updated using the gradient descent algorithm. For each weight connecting neuron j in layer l to neuron i in layer $(l+1)$, the weight is computed as:

$$\Delta w_{ij} = -r * \delta_{\iota} * a_j \tag{4}$$

where r is the learning rate, δ_{ι} is the gradient error of neuron i, and a_j is the activation function of neuron j.

Backpropagation through hidden layers: In order to compute the error gradients for the neurons in the hidden layers, the gradients from the subsequent layers are propagated backward. For each neuron in layer l, the error gradient is computed as:

$$\delta_l = f'(z_l) * \left[W^T * \delta_{l+1}\right] \tag{5}$$

where W^T is the transpose of the weight matrix connecting layer l to layer $(l+1)$.

2.3. Model's Parameters and Hyperparameters

Before designing an artificial neural network, we need to consider many issues related to modeling the network because these affect its performance. More specifically, we must define the model's parameters and hyperparameters [23–25].

The model's parameters are evaluated by the algorithm. Algorithm determines how to operate the input data to obtain the desired output, and they are obtained during the training process. These are typically the weights.

The hyperparameters of an ANN are variables that define the network's architecture and establish how the network is trained. They are specified before the training process. We can classify them as follows [26,27]:

1. Hyperparameters that define the neural network structure:
   - The quantity of hidden layers;
   - The quantity of nodes in each layer;
   - The kind of activation function.
2. Hyperparameters determine the way the network training and directly control the training process:
   - Learning Rule Optimizer type;
   - Weight initialization methods;
   - Type of loss (or cost) function;
   - Learning rate;
   - Batch size;
   - Number of epochs;
   - Training steps.
3. Hyperparameters that operate the regularization effect and directly control the overfitting:
   - Regularization parameters;
   - Lambda-λ in L1 and L2 regularization;
   - Dropout rate during the dropout regularization.

Figure 2 illustrates the hyperparameters' classification.

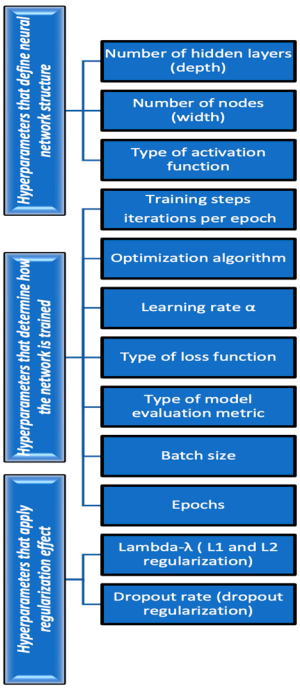


Figure 2. Hyperparameter classification.

2.4. The Hyperparameter Optimization

Hyperparameter optimization is the procedure of detecting the hyperparameter combination to reach the highest performance and effectiveness of the mode and is really a function optimization problem. The general formation for optimization issues is as follows [27,28]:

$$\min \mathrm{l}(\mathrm{w}) = \frac{1}{\mathrm{n}} \sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{li}(\mathrm{h}(\mathrm{xi}; \mathrm{w}), \mathrm{yi}) \quad (6)$$

where:

- $(\mathrm{xi}, \mathrm{yi})$ with i = 1, 2, . . ., *n*—the training data;
- w—the model's weights;
- h—prediction function;
- li—loss function.

The complexity of an ANN demands a large amount of storage and computing power; our goal is to build models that have a minimal level of complexity.

In the first phase of our model, the training procedure aims to determine the appropriate quantity of neurons and layers. The second phase of training aspires to develop the model's accuracy further.

2.4.1. Optimization of the Hyperparameters That Define the Neural Network Structure

A usual ANN includes three layers: the input, the hidden, and the output. The number of nodes in the input layer is similar to the number of input features. The number of neurons at the output layer equals the quantity of output variables [29,30].

The steps to follow to build a multi-layer network are as follows:

- The determination of input nodes;
- The identification of hidden layers and nodes;
- The determination of output nodes;
- The activation function.

The activation function is the final hyperparameter that determines the network structure. The goal of the activation function is to orient non-linearities into the network and is utilized in the layers of neural networks.

No activation function is needed for the input layer. The kind of activation function to be employed in the hidden layers and the output layer is determined by the problem we aim to solve.

Although there are many approaches to finding the optimal ANN architecture, none can guarantee the optimal solution for all real prediction problems. More than one hidden layer can comprise each ANN. Kolmogorov's theorem states that 'an ANN with one hidden layer could be selected if someone chooses a suitable quantity of neurons in the hidden layer' [30]. Moreover, the basic empirical principle of machine learning, Ockham's Razor, states that 'The best models are simple models that fit the data well'. The complexity of an ANN demands a large amount of storage and computing power, so we try to construct models with low complexity [23].

- The bias–variance dilemma [31,32].

Bias: The bias is the error between the prediction of the values by the network and the right value. Bias defines the model's capability to learn from the training data. A high-bias network causes a large error in the training and testing data. An algorithm should be low-bias to keep away from the problem of underfitting.

Variance: The difference between the training accuracy and testing accuracy. The variance defines how well a network can generalize to the test data.

The bias–variance tradeoff appears as it is complicated to minimize bias and variance simultaneously. When the complexity of an ANN increases, its bias decreases, and its variance increases. Contrarily, when the complexity of an ANN decreases, its bias increases, and its variance decreases. The optimum complexity of an ANN relies on the available dataset and the specific model [33].

2.4.2. Optimization of Hyperparameters That Determine the State of the Network Training and Directly Control the Training Process

The hyperparameters that define the way the network training and directly control the training process are [26,34,35]:

Learning rule—the optimizer type: Reveals the algorithm employed for the network training. There are various types of optimizers commonly used in ANNs. Some of them are gradient descent (backpropagation) [36], stochastic gradient descent (SGD) [36], Stochastic Gradient Descent with Momentum (SGDM), the Levenberg–Marquardt algorithm [23], Bayesian regularization [23], RMSprop (Root Mean Square Propagation) and Adam (Adaptive Moment Estimation) [37]. Choosing an optimizer often relies on the specific issue, architecture, and dataset; in our project, we use a trial-and-error method to find the finest one for the PTFD.

Weight initialization methods: Weight initialization refers to arranging the initial values of the weights in an ANN. Appropriate initialization is essential because it can affect an ANN's convergence speed and performance. Some standard weight initialization methods include random initialization, Xavier initialization, and He initialization. These methods aim to set the initial weights to balance the activation values and gradients during training.

Type of loss (or cost) function: The cost function estimates the difference between the forecasted output of an ANN and the real output. It quantifies the error of the model's predictions and is operated to direct the learning process. The selection of the loss function depends on the model type and the data's nature. Standard loss functions are the mean squared error (MSE), Huber Loss, and categorical cross-entropy.

Learning rate: This hyperparameter controls the step size at which the model updates its weights during training.

Batch size: It defines the amount of training data that is processed before the model's weights are updated. Employing a larger batch size during training can result in a shorter training time but may demand more memory. Shorter batch sizes can supply more stochasticity and better generalization, but they take longer to converge.

Number of epochs: An epoch represents a full pass through the complete training dataset during the training process of an ANN. The number of epochs determines how frequently the model passes and learns from the complete dataset. When the number of epochs rises, the model's performance is enhanced, but it also poses a risk of overfitting. The optimum number of epochs relies on the problem's complexity and the sample's size; moreover, it is usually defined using experimentation.

Training steps: Training steps refer to the iterations or updates made to the model's parameters during the training process. Each training step involves feeding a batch of training examples to the model, computing the loss, and updating the weights based on the gradients. The number of training steps depends on factors such as the batch size, training dataset size, and convergence criteria. The number of epochs and the size of each epoch determine it.

2.4.3. Optimization of Hyperparameters That Operate Regularization Effect and Directly Control the Overfitting

- Techniques to prevent overfitting

Overfitting in machine learning refers to a phenomenon where a model becomes excessively specialized in capturing the patterns and noise present in the training data to the extent that it fails to generalize well to new, unseen data. Overfitting often arises when a model is overly complex compared to the available data, causing the model to memorize the training examples rather than learn the underlying relationships. Consequently, the model may demonstrate high accuracy on the training data but perform poorly when presented with new data, rendering it unreliable and limited in practical applications. Overcoming overfitting is a significant challenge in machine learning, which requires careful consideration of various aspects such as model selection, feature engineering, regularization techniques, and appropriate validation strategies [32]. These measures aim to mitigate the effects of overfitting and foster robust generalization, ensuring that the model can effectively apply its learned knowledge to new, unseen data [33,35]. The most common methods for overcoming overfitting are:

- L_1 regularization
- L_2 regularization
- Early Stopping

L_1 regularization: Also referred to as Lasso regularization, this is a technique employed in neural networks to combat overfitting and encourage sparsity in the learned weights. It achieves this by incorporating a regularization term into the loss function, which penalizes the presence of large weight values. To be more specific, this approach involves adding the sum of the absolute values of the weights (denoted as R) to the loss function. Assuming the weights of the neural network are represented as W, the modified loss function can be expressed as follows:

$$\mathbf{R} = \boldsymbol{\lambda} * \sum |\mathbf{W}| \tag{7}$$

where $\boldsymbol{\lambda}$ is the regularization parameter that controls the strength of the process. Thus, we have the regularized loss function (RGF), which is obtained by adding the $\mathbf{L}_1$ regularization term to the loss function:

$$\mathbf{L_{reg}} = \mathbf{L} + \mathbf{R} \tag{8}$$

In this method, the weight update is slightly different from the standard backpropagation algorithm. The derivate of the regularized loss function concerning the weights is calculated and then employed to modify the weights. The gradient of the RGF is computed as:

$$\frac{\partial \mathbf{L_{reg}}}{\partial \mathbf{W}} = \frac{\partial \mathbf{L}}{\partial \mathbf{W}} + \boldsymbol{\lambda} * \mathbf{sign}(\mathbf{W}) \tag{9}$$

where $\frac{\partial \mathbf{L}}{\partial \mathbf{W}}$ is the gradient of the original loss function with respect to the weights and $\mathbf{sign}(\mathbf{W})$ represents the sign of each weight element (positive, negative, or zero). For each

weight connecting neuron **j** in layer **l** to neuron **i** in $(\mathbf{l}+\mathbf{1})$ layer, the weight update is computed as:

$$\mathbf{\Delta w_{ij}} = -\mathbf{r}*\left[\frac{\partial \mathbf{L}}{\partial \mathbf{w_{ij}}} + \boldsymbol{\lambda}*\mathbf{sign}\left(\mathbf{w_{ij}}\right)\right] \tag{10}$$

where **r** is the learning rate.

L2 regularization: Also known as Ridge regularization, this is another commonly used technique in neural networks for preventing overfitting. In this regularization method, the regularization term is of the form:

$$\mathbf{R} = \boldsymbol{\lambda}*\sum \mathbf{W}^2 \tag{11}$$

where $\boldsymbol{\lambda}$ once again controls the strength of the process. The regularized loss function has the same form as Equation (5). The gradient of the regularized loss function with respect to the weights is computed as:

$$\frac{\partial \mathbf{L_{reg}}}{\partial \mathbf{W}} = \frac{\partial \mathbf{L}}{\partial \mathbf{W}} + 2*\boldsymbol{\lambda}*\mathbf{W} \tag{12}$$

For each weight $\mathbf{w_{ij}}$ connecting neuron **j** in layer **l** to neuron **i** in layer $(\mathbf{l}+1)$, the weight update is computed as:

$$\mathbf{\Delta w_{ij}} = -\mathbf{r}*\left(\frac{\partial \mathbf{L}}{\partial \mathbf{w_{ij}}} + 2*\boldsymbol{\lambda}*\mathbf{w_{ij}}\right) \tag{13}$$

By incorporating the $\mathbf{L}_1$ and $\mathbf{L}_2$ regularization terms into the loss function and adjusting the weight update rule, we can attain sparsity in the learned weights. This can be advantageous in reducing the ANN's complexity and enhancing its knowledge to generalize well to unseen data.

Lambda parameter—λ in L_1 and L_2 regularization

The λ factor assumes a significant role in determining the degree of L_1 and L_2 regularization.

Different values of λ correspond to different levels of regularization:

- When λ equals 0, no regularization is applied;
- When λ equals 1, complete regularization comes into effect;
- The default setting for Keras is λ = 0.01.

The differences between L1 and L2 are presented in Table 3.

Table 3. The differences between L_1 and L_2.

L_1 Regularization	L_2 Regularization
Shrinks weights magnitudes toward 0	Shrinks weights magnitudes to be small but not precisely 0
Penalizes the sum of the absolute values of the weights	Penalizes the sum of the square values of the weights
The cost of outliers current in the data raises linearly	The cost of outliers current in the data raises exponentially
Preferable when the model is simple	Preferable when the model is complex

In our methodology, we present a training procedure with L_2 regularization, which includes two steps, in order to construct the final network.

First step: Define the proper number of neurons and layers.

Second step: Upgrade the model's accuracy.

2.5. Visualize and Estimate the Performance of a Classifier in the Classification Learner App (CLA)

The experiment is executed in MATLAB R2023a in the Classification Learner app [38]. After training an ANN, the CLA automatically creates the confusion matrix (CM) and the receiver operating characteristic (ROC) curve of the model.

2.5.1. ROC Curves

The ROC curve [39,40] is widely employed in the realm of machine learning for classification tasks. In the context of a multi-class issue, it is possible to develop multiple ROC curves, each comparing the classifier's performance for one class against all the other classes. This is achieved by identifying each class as the "positive" class and calculating the ROC curve for that binary classification task. It provides a graphical representation of the performance of a classifier as the discrimination threshold varies. The ROC curve is formed by graphing the true positive rate (TPR) compared to the false positive rate (FPR) at different levels of threshold. The TPR, often referred to as sensitivity or recall, is the ratio of positive cases that were properly marked as positive. The false positive rate (FPR), conversely, is the ratio of negative events that are inaccurately identified as positive.

2.5.2. Confusion Matrix

The CM [39,40] is an essential tool in the realm of machine learning for estimating the performance of classification models. It helps to understand how well a model performs in terms of its predictions for different classes. The confusion matrix is especially useful when dealing with problems where you have multiple classes that your model is trying to classify instances into. For a classification problem with N classes, the CM is an $N \times N$ matrix that summarizes the performance of a classification model shown in Figure 3.

		Predicted			
		Class 1	Class 2	...	Class N
Actual	Class 1	TP	FN		FN
	Class 2	FP	TP		FN
	...	...	...		...
	Class N	FP	FP		TP

Figure 3. Confusion matrix.

Mathematically, these values can be expressed as follows:

- True positives (TP): The number of occurrences where the actual class is positive (actually positive) and the model predicted it as positive;
- True negatives (TN): The number of occurrences where the actual class is negative (actually negative), and the model predicted it as negative. This is often more relevant in binary classification;
- False positives (FP): The number of occurrences where the actual class is negative (actually negative), but the model predicted it as positive;
- False negatives (FN): The number of occurrences where the actual class is positive (actually positive), but the model predicted it as negative.

Using these values, various metrics can be calculated to assess the performance of the ANN [41,42], such as:

- Accuracy: (TP + TN)/(TP + TN + FP + FN);
- Precision: TP/(TP + FP);
- Recall = TP/(TP + FN);
- F1 Score = 2 ∗ (Precision ∗ Recall)/(Precision + Recall);
- Specificity = TN/(TN + FP).

Recall concentrates on the model's capability to determine positive instances correctly.

F1 Score balances precision and recall, which is valid when evaluating both false positives and false negatives.

Specificity calculates the model's ability to accurately specify negative instances, which is essential in procedures where false positives are inappropriate.

- These metrics provide different perspectives on the performance of the ANN. The confusion matrix and the derived show where the model is making correct predictions

and where it is making mistakes, which is essential for improving the model or choosing the right model for a given task.

2.6. Experiment

In this manuscript, the MATLAB R2023a software is applied to develop an optimized Multi-Layer Feedforward Backpropagation Neural Network. In a few words, the experiment is carried out as follows: After training 30 NNs, we select the five models with the highest validation accuracy. We then conduct tests on these five ANNs using the Experiment Manager app and ultimately select the neural network with the best performance.

Finally, the best ANN is selected to optimize the hyperparameters and, conclusively, to present the model with the best accuracy. In this task, we examine three automated techniques, Bayesian optimizer, random search, and grid search, and three optimizers, Adam, SGDM, and RMSprop, to assist the hyperparameter selection for our ANN. The workflow of our method is as follows:

Flowchart:

1. Data extraction;
2. Load data in the Classification Learner app;
3. Select classifier options;
4. Train classifiers;
5. Choose the best classifier type (ANN);
6. Train 30 ANNs with different parameters;
7. Select the five most accurate networks;
8. Train and test 5 ANNs and select the most accurate model;
9. ANN hyperparameter optimization;
10. Visualize and assess the ANN's performance;
11. Select the ANN with the best accuracy for PTFD.

2.6.1. First Experiment—Selecting the Best Classifier Type

In this experiment, the most effective classification model is chosen by automatically training six classifiers and comparing their validation results (Table 1).

From the Statistics and Machine Learning Toolbox, we select the Classification Learner app (CLA) [39]. This app conducts supervised machine learning in order to classify data by training various types of models. For training a model, a known set of data (the four gas ratios) called observation is provided as the input. We provide as output the known responses to the input data called labels or classes (the five transformer's incipient faults). The Classification Learner training consists of two stages [39,40]:

Model validation: Train a model using a validation scheme (that is applied to monitor the state of the training stage and to fine-tune the performance of the ANN). By default, the application employs cross-validation to prevent overfitting.

Full model: A model based on complete data without validation. The application synchronously trains the full model and the validated model.

An important issue for the CLA is that it does not use test data for training the model. So, it displays the validated model's results. The diagnostic criteria, for example, the model accuracy and visualizations, such as the confusion matrix, represent the validated model's outcomes. The CLA employs Bayesian optimization by default to tune the hyperparameters.

The CLA offers classifier performance indicators, including:

- Validation accuracy: Percentage of properly identified observations;
- Total cost: Misclassification cost;
- Prediction speed: Estimated speed for test data, founded on the prediction timings for the validation data. This speed may be impacted by background operations both within and outside of the app; therefore, we train models in equal states for more accurate comparisons;
- Training time: Duration of the model's training (train models in equal states for more accurate comparisons);

- Model size: the model's size if it were exported with no training data.

Experiment settings:

1. Data extraction.

The four gas ratios are the inputs (predictors), and the output classes (response) are the five transformer's incipient faults. The total number of data samples is 400.

2. Load data in the Classification Learner app.

The predictor data are combined into a table (350 × 5), and the other 50 samples constitute the test data, where the first four columns consist of the gas ratios and the last column comprises the five classes. Each row of the table indicates one observation, so we have 350 observations for the train set and 50 observations for the test set (Table 4).

Table 4. Dataset size.

Number of Predictors	Number of Observations	Number of Classes	Response
4	350	5	Faults

3. Select classifier options.

By default, the CLA uses the cross-validation method. According to this scheme, consider that there are no data for the testing process. The authors have selected to train the following classification models, as they are the most popular in the literature [39].

- Decision trees;
- Naive Bayes classifiers;
- Support Vector Machines (SVMs);
- Kernel;
- ANN.

The summary of our experimental results is presented in Table 5:

Table 5. Experiment summary regarding different classifiers.

Classifiers	Classifier Type	Accuracy% (Validation)	Cost (Validation)	Prediction Speed (obj/s)	Training Time (s)	Model Size (KB)
Decision trees	Fine Tree	78	77	6388.3	6.37	11.991
	Medium Tree	78	77	21,522.5	0.87	11.991
	Coarse Tree	67.4	77	20,652.2	0.37	11.991 5.011
SVM	Quadratic	77.7	78	9763.7	0.7	60.282
	Cubic	80.9	67	10,405.0	0.6	58.026
	Fine Gaussian	81.1	66	9530.5	0.7	65.810
	Medium G.	80.6	68	10,127.4	0.6	65.042
	Coarse G.	72.0	98	10,758.9	0.5	76.082
KNN	Fine	80.3	69	7547.1	0.7	28.934
		80.3	69	7547.1	0.7	28.934
	Medium	77.4	79	9726.3	0.4	28.934
		77.4				
	Cubic	76.9	81	10,022.9	0.4	28.934
	Weighted	81.4	65	10,189.9	0.4	28.952
ANN	Narrow	81.4	65	13,372.7	3.8	6.143
	Medium	82.9	60	21,741.1	2.5	7.343
	Wide	81.1	66	20,582.1	2.8	13.343
	Bilayered	82.0	63	22,651.2	3.6	7.943
	Trilayered	81.7	64	21,268.2	4.6	9.743
Naïve Bayes	Kernel	70.0	105	6570.1	1.1	92.444

Experimentation outcomes: As illustrated in Table 5, the ANN is the classifier with the highest performance metrics. It maintains the highest accuracy, the lowest total cost, the fastest prediction speed, the shortest training time, and a competent model size in Kb.

2.6.2. Second Experiment—Finding the ANN with the Best Performance

In this experiment, we train 30 ANNs with different parameters in the Experiment Manager app [39,40,43]. The Experiment Manager app (ExpM) facilitates building deep learning experiments for training ANNs under different initial states and evaluating the outcomes. The experiment is conducted as follows [39]:

- Finding the best possible training choices using Bayesian optimization;
- Operating the trainNetwork built-in function or designing a unique training function;
- Evaluating the performance of various network topologies by comparing the outcomes of utilizing distinct datasets.

In the Experiment Manager app, we train 30 NNs with the following parameters:
Number of layers: The ExpM quests among one, two, or three fully connected layers.
Layer size: The ExpM quests in the content of 1 to 300.
Activation functions: Tanh, ReLU, Sigmoid, none.
Regularization strength (L): The ExpM quests among the content $\{1 \times 10^{-5}/n, 1 \times 10^{-5}/n\}$, where n = 350, and states the number of observations.
Iteration limit: 1000.
Experimentation outcomes: Based on the findings collected from the previous training, we chose and presented the five models with the highest validation accuracy. The outcomes are illustrated in Table 6.

Table 6. The five ANNs with the best accuracy.

NumLayers	Activation Function	Layer 1 Size	Layer 2 Size	Layer 3 Size	Validation Accuracy
1	Sigmoid	240	5	0	75.0
3	Tanh	200	100	5	80.1
3	ReLu	300	200	5	80.2
3	ReLu	100	5	0	80.2
3	Tanh	100	50	5	80.2 80.9

We then conducted tests on these five ANNs using the ExpM app. We manually trained and tested them and ultimately selected the neural network with the best performance. We selected the ANN with the following parameters (Table 7) to optimize its hyperparameters and, conclusively, to present the model with the best accuracy.

Table 7. ANN parameters.

Num. of Layers	Activation Function	Layer 1 Size	Layer 2 Size	Layer 3 Size	Validation Accuracy (%)
3	Tanh	100	50	5	80.9

2.6.3. Third Experiment—ANN Hyperparameter Optimization

In the CLA, the last experiment is conducted with the prefinal ANN.

- Firstly, we systematically explore various values for each hyperparameter using the grid, random, and Bayesian search approaches to identify the optimal combination of hyperparameters that maximizes the performance on the validation set (search strategy).
- Secondly, we tune the hyperparameters for the three optimizers according to the optimal combination of hyperparameters we find with the previous search strategy.

1. Hyperparameter Tuning Approaches

Various searching strategies are used to assess the performance of the models on the validation dataset [25,38]. We employ the subsequent search techniques that have been extensively discussed in the academic literature: manual search, grid search, random search, and Bayesian optimization [39,43].

In grid search (Model 4 for our experiment), various models are formed based on a grid of data prior to the searching process, whereas other methods specify the models repeatedly based on a specific approach during the searching process. The process includes the establishment of a grid with various hyperparameter values, followed by a comprehensive exploration of all potential combinations. This approach may be computationally demanding. The objective is to identify the optimal combination of hyperparameters for a certain model, and the performance of each combination is assessed using cross-validation.

The random search method (Model 8) involves the specification of a distribution or range for each hyperparameter, followed by the random sampling of values from these specified distributions or ranges. Every combination of hyperparameters that is sampled is assessed using either cross-validation or a distinct validation set in order to determine its performance. The efficiency of the hyperparameter optimization process may be enhanced by assigning varying degrees of importance to different hyperparameters.

Bayesian optimization (Model 12) is a cutting-edge approach to optimizing costly black-box functions globally. It comprises two fundamental elements: a probabilistic surrogate model and an acquisition function [25]. In general, Bayesian optimization is a robust and adaptable optimization model that has shown success in the realm of hyperparameter optimization as well as other optimization challenges. The algorithm effectively examines the whole range of possible solutions, adjusts its approach based on the observed values of the objective function, and ultimately reaches optimal performance.

Manual search or trial-and-error method (Model 2) is an approach where machine learning and deep learning methods are configured manually by users based on their own experience and trial and error. In this approach, users rely on their expertise and knowledge of the data, methods, and parameters to determine reasonable hyperparameter values.

Experiment 3A

In the CLA, we determine our experiment in the subsequent stages. We train the neural network for each hyperparameter setting and assess its performance by measuring the validation accuracy and loss. Furthermore, the model's performance was evaluated using the confusion matrix, the ROC curve, and the minimum classification error plot of the different search methods.

1. Define the Hyperparameters: Identify the tuning hyperparameters: learning rate (LR), regularization strength (L2), momentum (M), and batch size (BS).
2. Define the Search Space: Determine the range of values that each hyperparameter can take:
    - LR search space between 0.001 and 0.1;
    - L2 search space between 0.00001 and 0.1;
    - M search space between 0.5 and 0.99;
    - BS search space between 32 and 4096.
3. Choosing a Search Strategy: Random search, Bayesian optimizer, grid search, random search.
4. Train and Evaluate: For each hyperparameter setting, we train the neural network and assess its performance by measuring the validation accuracy and loss.

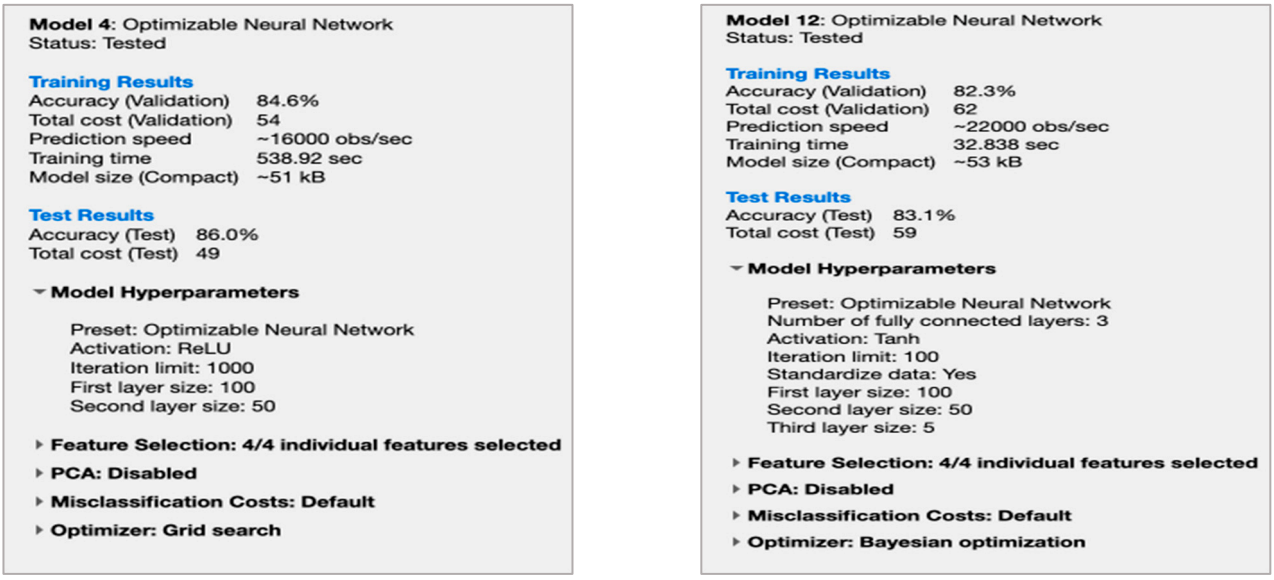
Various search strategies are used to assess the performance of models on the validation dataset, ultimately selecting the most optimal one. In Table 8 are presented the validation accuracy, the test accuracy, and the L2 norm for each algorithm. Experimentation outcomes:

Table 8. Automated optimization.

Algorithm	Validation Accuracy	Test Accuracy	L2
Random search (Model 8)	82	85	0.001
Bayesian optimizer (Model 12)	82.86	83.1	0.00002
Grid search (Model 4)	84.57	86	0.00002
Manual search (Model 2)	80.9	85	0.01

This experiment's results are illustrated in the following figures: Figures 4–11. We present the confusion matrix and the ROC curve for each model. Model 12 (Bayesian optimizer) and Model 4 (grid search) stand out as the most significant models in Table 8 because of their high validation and test accuracy. Their significance indicates their importance in the analysis, so we present a summary of the experimental process and the minimum classification error plot only for these two models.

Figure 4 reveals an overview of the training and test results according to the performance indicators for Model 4 and Model 12, as reported by the MATLAB R2023a software. These CLA classifier performance indicators have been described in Section 2.6.1.



(**a**) (**b**)

Figure 4. (**a**) Summary of Model 4; (**b**) Summary of Model 12.

From Figure 5, the minimum classification error plot (this plot was created automatically by the MATLAB R2023a software) for Bayesian optimization, we can observe the following, very important for our experiment, outcomes:

- The estimated minimal classification error is represented by each light blue point. This estimation is obtained using the optimization procedure, which takes into account all the combinations of hyperparameter values that have been estimated, involving the present iteration.
- The observed minimal classification error is represented in the graph by every dark blue point, which refers to the calculated error obtained during the optimization procedure.
- The optimized hyperparameters are represented by the red square, representing the iteration with the best performance. As we can see from Figure, the best point hyperparameter is the L2 with value 0.00000287, and the observed min classification error is 0.172 in the eighth iteration. The optimized hyperparameters may not consistently provide the reported minimal classification error. Here, we can also mention that the application employs Bayesian optimization for hyperparameter tuning. It selects a combination of hyperparameter metrics that eliminates the upper confidence range

of the objective model's classification error instead of minimizing the classification error itself.

- The hyperparameters that result in the smallest classification error are represented by the yellow point, indicating the corresponding iteration. Here, we observed that the L2 regularization with value 0.00000286 resulted in the min classification error 0.172 in the 6th iteration.

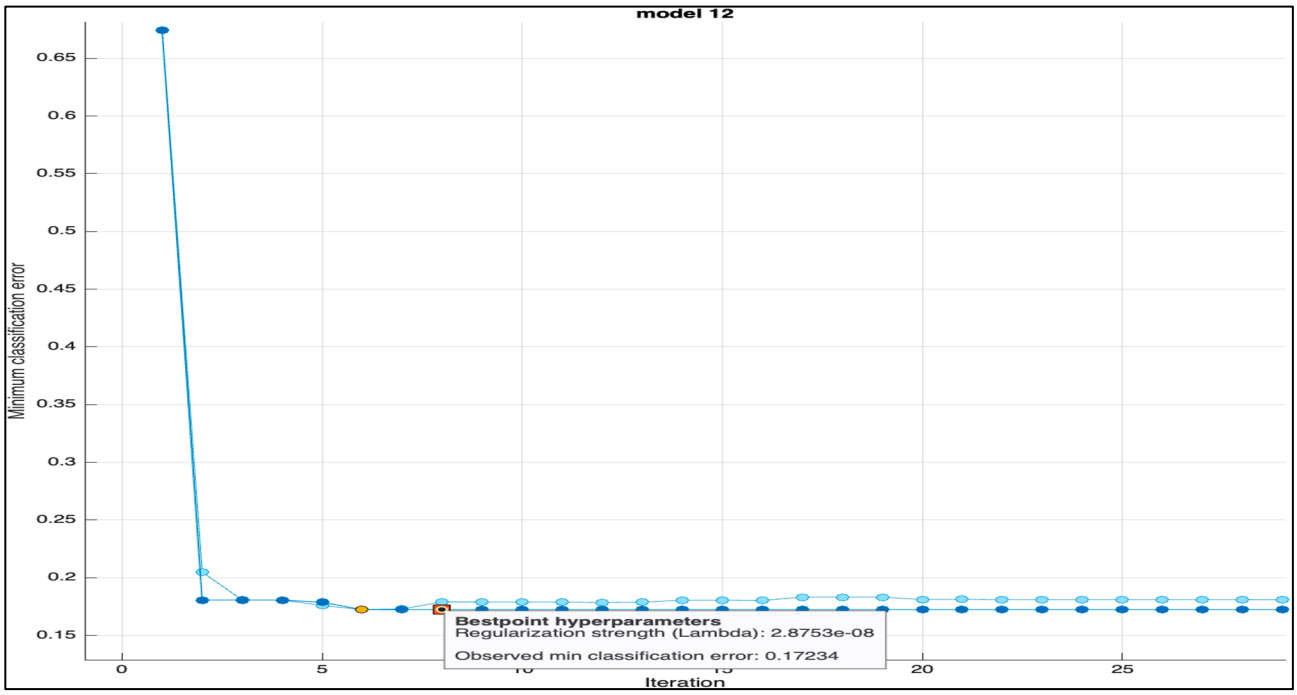


Figure 5. Min. Classification error plot of Model 12.

From Figure 6, for the grid search model, we can perceive that:

- The best point hyperparameter is the L2 regularization with value 0.00006155, the observed min classification error is 0.152 in the 55th iteration, and the L2 regularization with value 0.0000369 resulted when the min classification error occurs with value 0.152 in the 29th iteration. According to these two diagrams, we can conclude that grid search results in a more nominal classification error. Hence, it has better accuracy than Bayesian, as we have noticed from the confusion matrix and ROC curve. However, the minimum classification error plots indicate that Bayesian has a higher coverage speed as it reaches the minimum error in the sixth iteration.

Figures 7a, 8a, 9a and 10a illustrate the confusion matrix (CM) that summarizes the performance of each model. CM is a 5×5 matrix, as we have five classes for prediction in our ANNs. There is a general description of the CM in paragraph 2.5.2. and a precise description in articles [41,42].

The diagonal features (from top-left to bottom-right) define the correct predictions for each class, while the non-diagonal features denote misclassifications (incorrect prediction for each class). The rows designate the true (actual) classes, and the columns designate the predicted classes. The different colors in every orthogon are used to visually emphasize the various categories in the confusion matrix, making it easier to interpret the performance of the model at a glance. The choice of colors can differ based on the visualization tool or software being used. We will describe in detail the CM of Model 4 (Grid search) due to its superior validation and test accuracy performance.

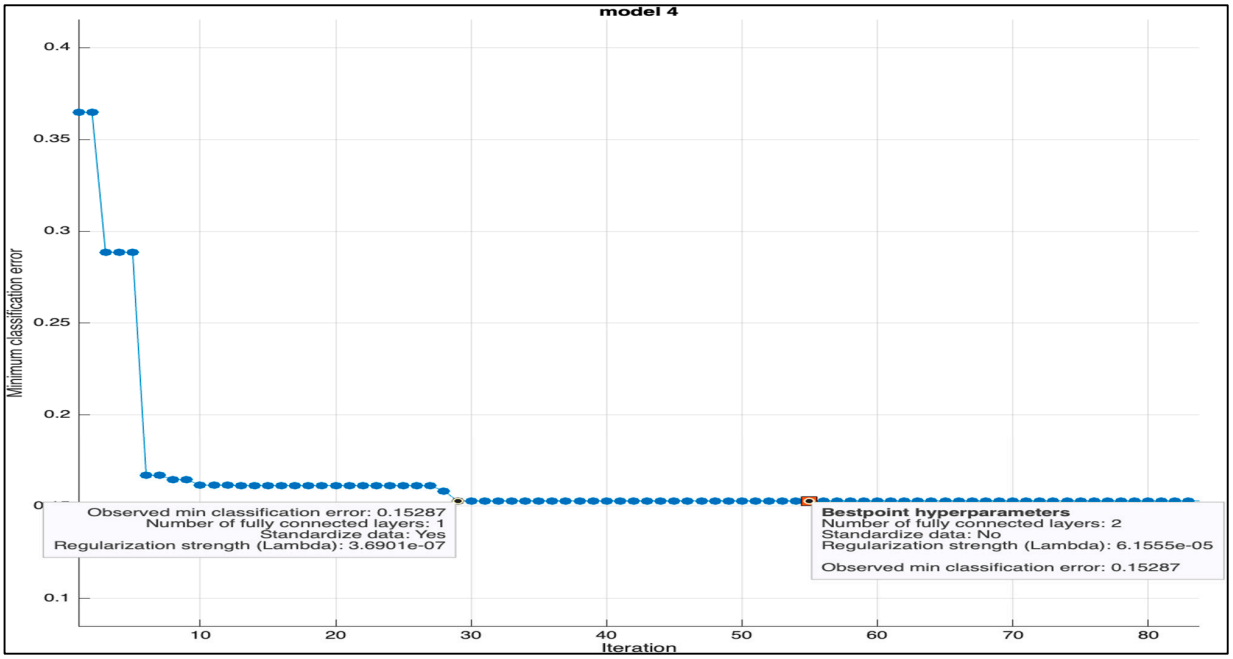


Figure 6. Min. Classification error plot of Model 4.

In the confusion matrix provided in Figure 7a, let us focus on the row corresponding to the actual Class 1 (or fault type 1). This row indicates the performance of the ANN in classifying instances where the true class is 1. The following is an explanation of the value of each element inside the given row:

- The first numerical value (83.1%) states that the neural network correctly predicted 83.1% of fault types as fault type 1. So, the true positives (TP) for Class 1 are 83.1% (blue orthogon).
- The second numerical value, denoted as 0, states that the neural network has not mispredicted type 1 as fault type 2. So, the false negatives (FN) for Class 1 are 0% (white orthogon).
- The third number (1.4%) states that the neural network incorrectly predicted type 1 as fault type 3 at a rate of 1.4%. The false negatives (FN) for Class 1 are 1.4% (light yellow orthogon).
- The fourth numerical value (8.5%) states that the neural network incorrectly predicted type 1 as fault type 4 at a rate of 8.5%. Here, the false negatives (FN) for Class 1 are 8.5 (orange orthogon).
- The fifth numerical value (7.0%) states that the neural network incorrectly predicted type 1 as fault type 5 at the rate of 7.0%. Here, the false negatives (FN) for Class 1 are 7.0% (orange orthogon).
- So, TP = 0.831, TN = 4.1, FP = 0.211, and FN = 0.169 for Class 1 in Figure 7a.

Therefore, the following metrics can be calculated to assess the performance of the ANN. The metrics for Class 1 are:

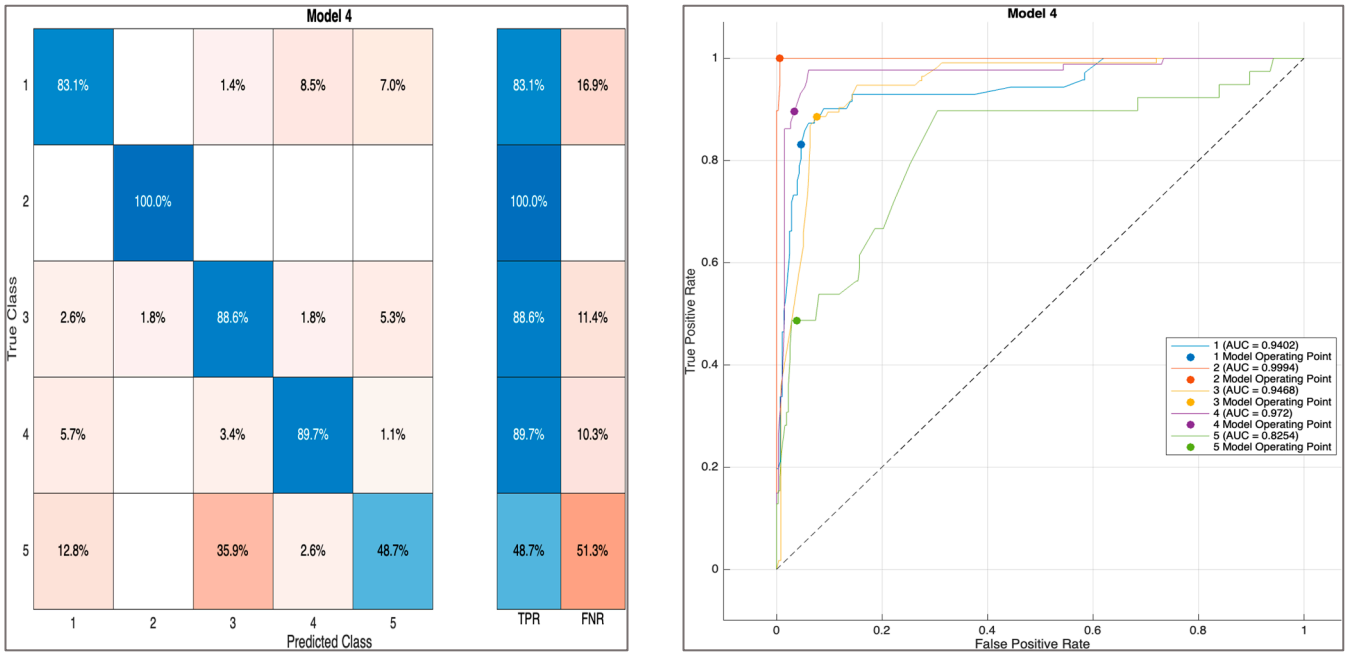
- Accuracy: (TP + TN)/(TP + TN + FP + FN) = 0.92;
- Precision: TP/(TP + FP) = 0.797;
- Recall = TP/(TP + FN) = 0.831;
- F1 Score = 2 ∗ (Precision ∗ Recall)/(Precision + Recall) = 0.813;
- Specificity = TN/(TN + FP) = 0.95.

The two columns, TPR and FNR, on the right of the CM show each class's total percentage rate of true positives (correctly classified) and the total percentage rate of false negatives (misclassified as other classes).

Figure 7b illustrates the ROC curves of Model 4. There is a general description of the ROC curves in paragraph 2.5.1. We are considering each class as the positive class and the

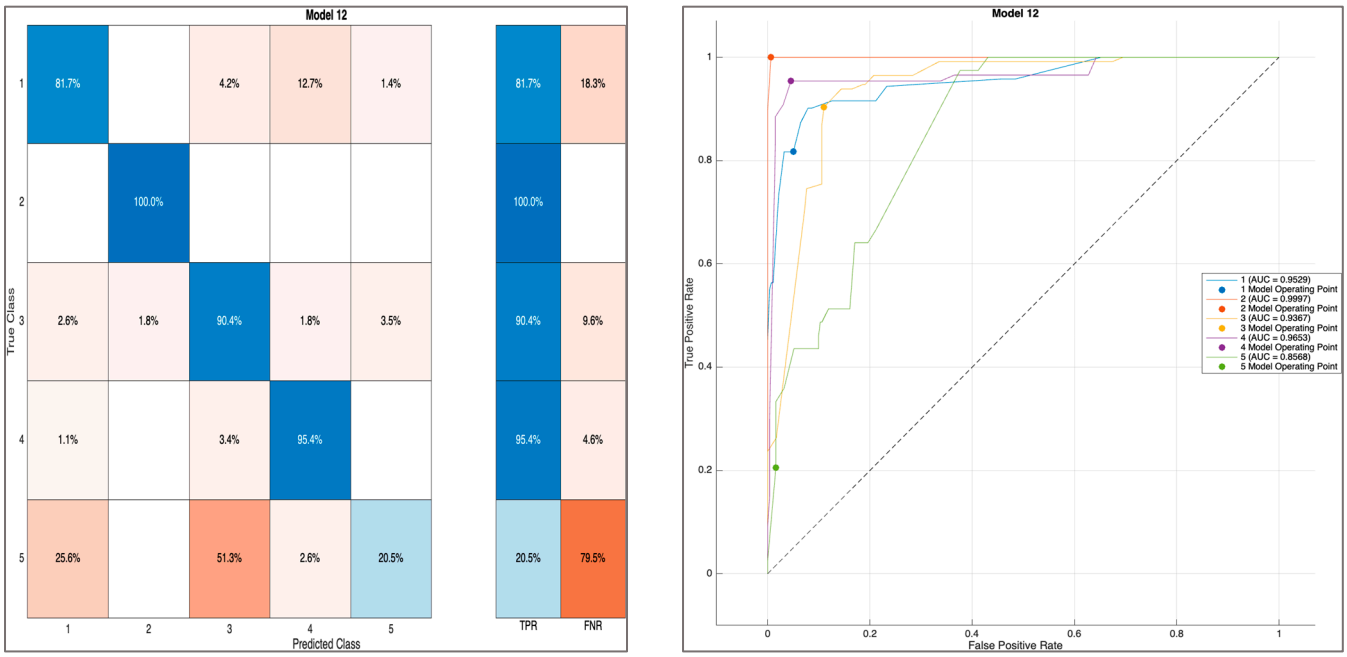
remaining classes are regarded as the negative class. We can perceive that the blue curve refers to Class 1, the red refers to Class 2, the yellow refers to Class 3, the purple refers to Class 4, and the green refers to Class 5. The operating point of each class indicates the TRP of the class. For example, the blue point of Class 1 has a value of 0.831, and each class have the same TRP value as in the CM.

The area under the curve (AUC) measures the classifier's total performance. Class 2 has an AUC = 0.9994, and Class 5 has an AUC = 0.825. AUC values vary from 0 to 1; a greater AUC value is indicative of superior performance.



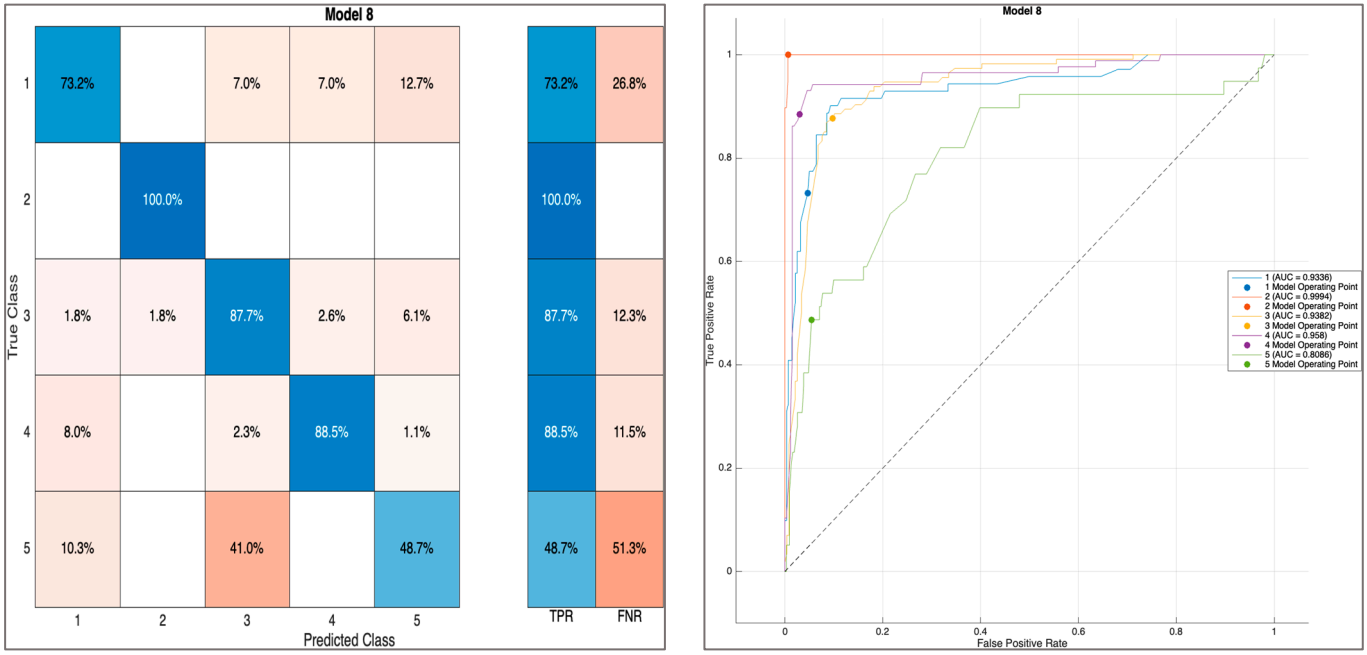
(**a**) (**b**)

Figure 7. (**a**) Validation confusion matrix for Model 4; (**b**) validation ROC curve for Model 4.



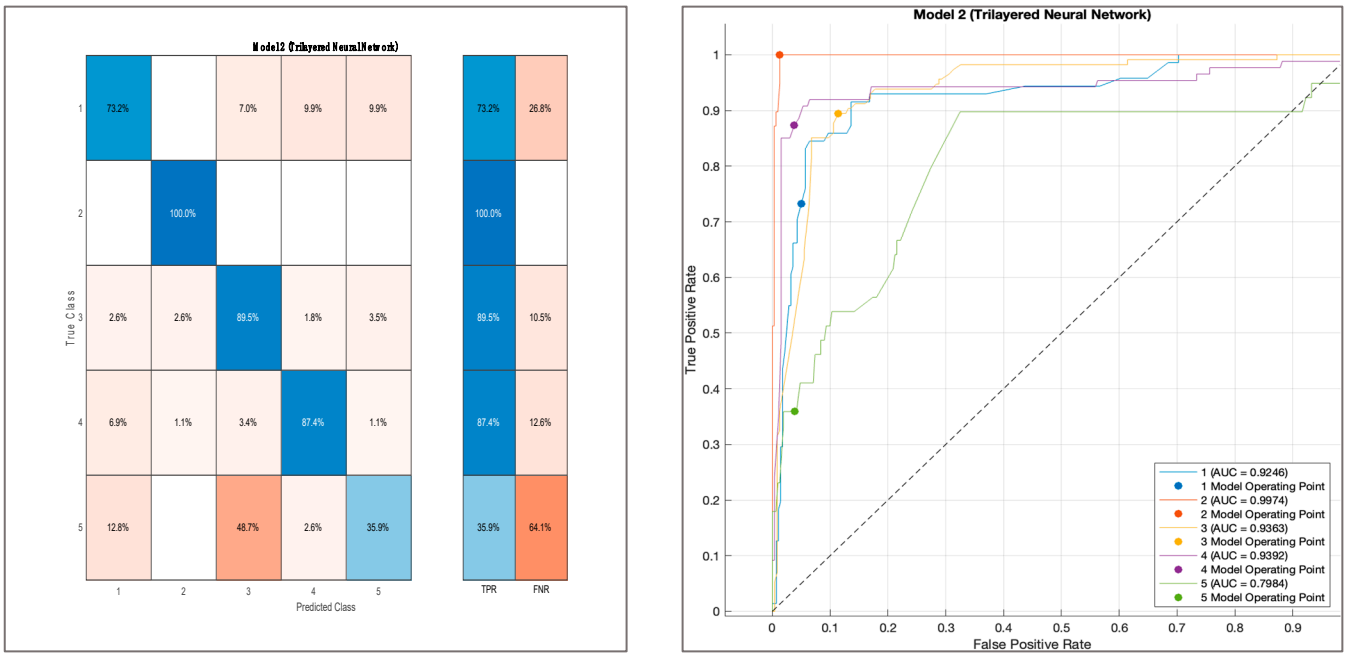
(**a**) (**b**)

Figure 8. (**a**) Validation confusion matrix for Model 12; (**b**) validation ROC curve for Model 12.



(**a**) (**b**)

Figure 9. (**a**) Validation confusion matrix for Model 8; (**b**) validation ROC curve for Model 8.



(**a**) (**b**)

Figure 10. (**a**) Validation confusion matrix for Model 2; (**b**) validation ROC curve for Model 2.

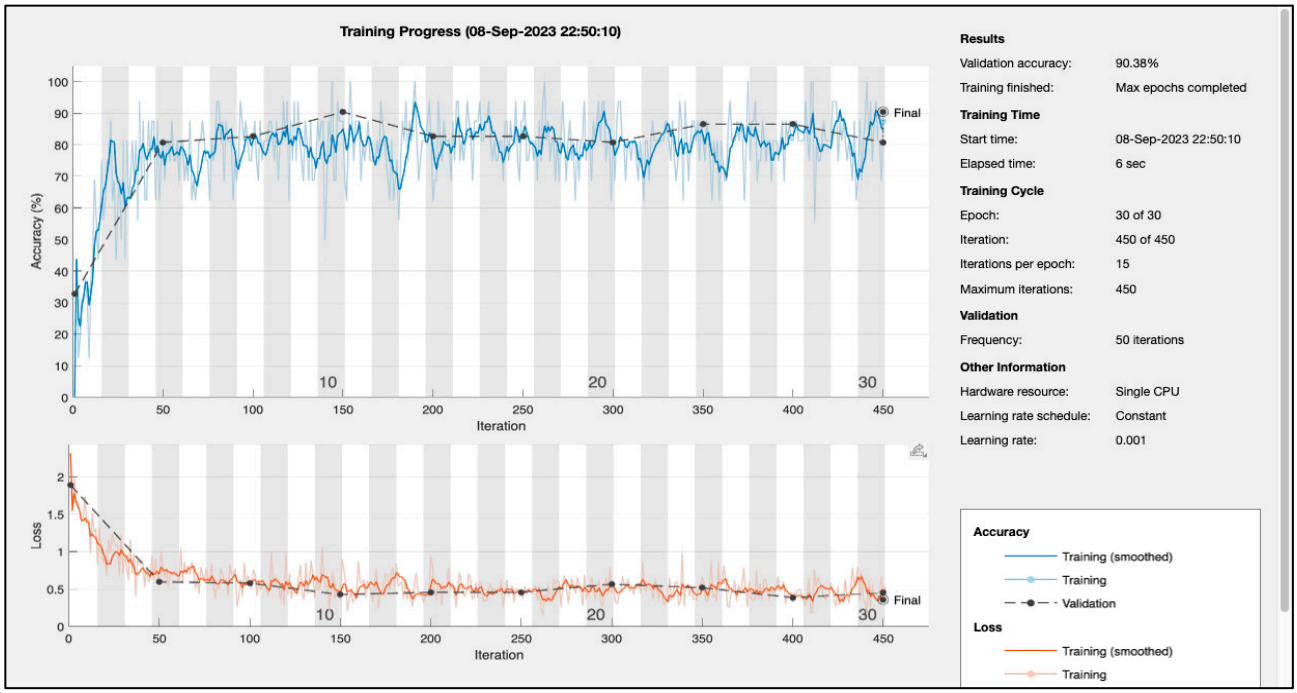


Figure 11. The progress of training for Adam.

2. Adaptive Learning Rate Method

The impact of the stochastic gradient descent (SGD) approach is significantly influenced by the manually controlled learning rate. Setting a suitable value for the learning rate is a difficult challenge. The learning rate may be automatically adjusted using several adaptive approaches [38,39]. These techniques do not need parameter adjustments, converge quickly, and often provide good outcomes. In this section, we examine three of them: Adaptive Moment Estimation (Adam), Mean Square Propagation (RMSprop), and Stochastic Gradient Descent with Momentum (SGDM).

Adam [37]: An improved SGD approach incorporating an adjustable learning rate for every parameter. Moreover, it combines the techniques of adjustable learning rate and momentum. The purpose of this architecture is to effectively modify the parameters of a model as it is being trained.

RMSprop: The fundamental concept behind RMSprop is to dynamically modify the learning rate assigned to each variable, by considering the size of the gradients. This concept is accomplished by continuously updating and calculating the average of the squared gradients for each parameter. The running average serves as a means of adjusting the learning rate, enabling it to be increased for parameters with lower gradients and decreased for values with higher gradients.

SGDM algorithm: In the conventional SGD approach, the model parameters are updated by considering the gradient of the loss function estimated on an individual training sample at each iteration. Noisy updates and weak convergence may occur, mainly when there is a substantial variation in the gradients. This problem is effectively addressed by introducing a momentum factor into the SGDM algorithm. The momentum term refers to a computed value representing the average of the gradients obtained from previous iterations. This approach facilitates the reduction of update irregularities and enables the ongoing progression of the optimization process in a desirable position, even amongst variations in the gradients. The update rule of SGDM has two primary elements: the present gradient and the momentum term. The existing gradient is scaled using a learning rate, which handles the magnitude of the update stage. The momentum term is multiplied by a coefficient that regulates the impact of past gradients on the present update. The model parameters are updated by combining these two components.

Experiment 3B.

In this section, we test the three optimizers, as mentioned above, in order to compare their classification accuracy. To achieve this, we adjust the hyperparameters (the learning rate, momentum, L2 regularization, and batch size) following a 'trial-and-error' process and notice which ones work best for the network [43,44]. There are specific hyperparameters for each optimizer that can be changed to improve the training performance. We will refer to them in the next paragraph. The momentum term and the learning rate are dynamically adjusted to improve the convergence speed. The L2 regularization term is adjusted to prevent over-fitting. Moreover, a proper batch size helps converge faster.

Adjustment of L2 regularization.

First, the standard cross-entropy function with the L2 regularization term is formed to prevent over-fitting. When the regularization parameter is set to zero, it poses the risk of overfitting and decreases the network's capacity to generalize. The regularization parameter is modified in order not to have an impact on the other precisely adjusted parameters inside the model. However, the consequences of it may be detected during the convergence of the loss function. The regularization parameter is often chosen from a set of commonly used values that are logarithmically distributed within the range of 0 to 0.1. We will adjust the L2 term as 0.1, 0.001, 0.00001.

After, we employ the same parameter initialization for comparing the three optimization algorithms. Finally, optimal values for the hyperparameters, such as the momentum and learning rate, are determined using exhaustive grid searching, and the resulting predictions are reported.

In the editor of MATLAB, we write the code for training and testing the three optimizers. We present an example of the code that processes the data for training the models in Box 1.

Box 1. Code for training.

```
InputTable = trainingData;
Predictor Names = {'INPUT1', 'INPUT2', 'INPUT3', 'INPUT4'};
predictors = inputTable(:, predictorNames);
response = inputTable.FAULT;
isCategoricalPredictor = [false, false, false, false];
classNames = [1; 2; 3; 4; 5];
```

A. Adam solver

We train the ANN using the Adam solver and set its properties.

The adjustable hyperparameters are:

- Squared Gradient Decay Factor;
- Learning rate;
- Gradient Decay Factor (controls the strength of L2 regularization factor);
- Epsilon (ε): A tiny value which is added to the denominator to prevent division by zero, usually in the range of 1×10^{-7} or 1×10^{-8}. Machine learning usually encounters Epsilon when computing ratios, gradients, or other mathematical procedures involving division. Adding Epsilon guarantees that the division stays defined even if the denominator is near zero and that the computation does not result in numerical instability or errors.

Set 'Gradient Threshold' as 1.

Experimentation outcomes:

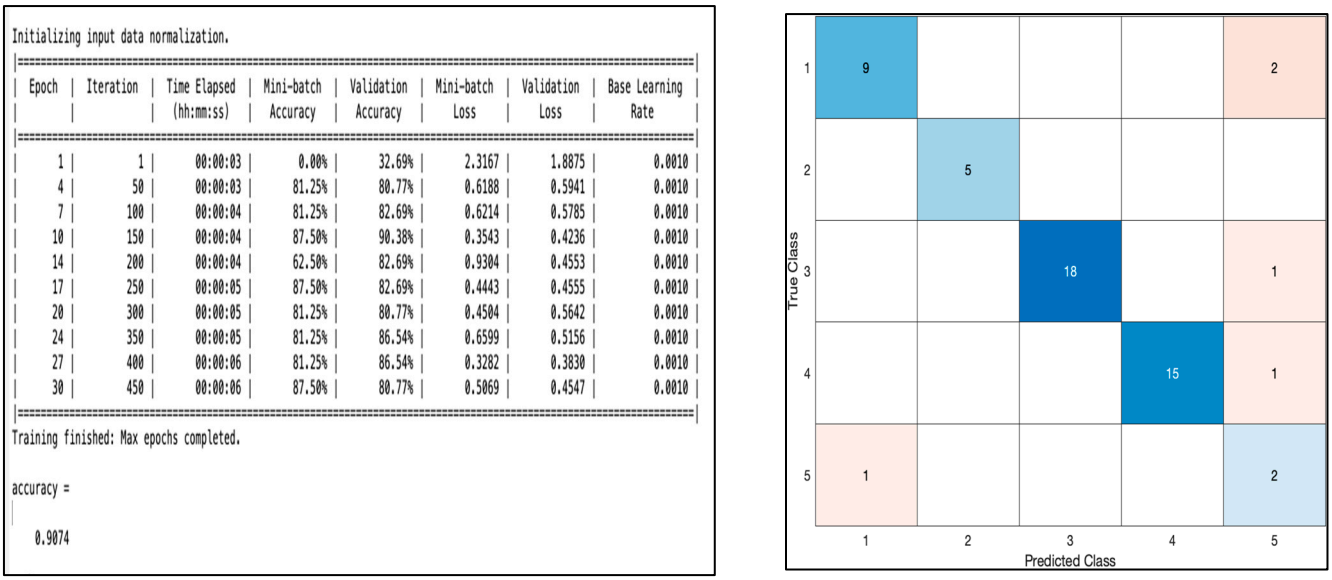
Following an intensive search based on the various criteria discussed above (paragraph 2. Adaptive Learning Rate Method), we have determined the following magnitudes of parameters provide the optimal results, as illustrated below:

The results obtained by employing the Adam optimizer are:

- Best validation accuracy: 90.4%

- Best test accuracy: 0.907
- Gradient Decay Factor: 0.999
- Learning rate: 0.001

We can observe the progress of the network's training validating and testing in Figures 11 and 12a and the confusion matrix in Figure 12b.



Initializing input data normalization.

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:03	0.00%	32.69%	2.3167	1.8875	0.0010
4	50	00:00:03	81.25%	80.77%	0.6188	0.5941	0.0010
7	100	00:00:04	81.25%	82.69%	0.6214	0.5785	0.0010
10	150	00:00:04	87.50%	90.38%	0.3543	0.4236	0.0010
14	200	00:00:04	62.50%	82.69%	0.9304	0.4553	0.0010
17	250	00:00:05	87.50%	82.69%	0.4443	0.4555	0.0010
20	300	00:00:05	81.25%	80.77%	0.4504	0.5642	0.0010
24	350	00:00:05	81.25%	86.54%	0.6599	0.5156	0.0010
27	400	00:00:06	81.25%	86.54%	0.3282	0.3830	0.0010
30	450	00:00:06	87.50%	80.77%	0.5069	0.4547	0.0010

Training finished: Max epochs completed.

accuracy =

0.9074

(**a**) (**b**)

Figure 12. (**a**) The progress of the network's training Adam; (**b**) test confusion matrix for Adam.

Figure 11 illustrates the training progress diagram, which displays various training metrics at each iteration. The diagram provides information on the training time, training cycle, and other parameters, which are displayed on the right side. The shaded backdrop serves as a visual representation of each training epoch (a full pass across the whole of the dataset).

The classification accuracy on each specific mini-batch is presented using a light blue curve; however, the program gives a smoothed version of the training accuracy (blue curve). We have provided a validation set, so the black curve shows the classification accuracy on the complete validation set. We set the validation frequency to 50 for estimating the ANN on the validation data every 50 iterations and the maximum quantity of epochs to 30. Moreover, we set the iterations to 450; every iteration comprises the estimate of the gradient and the subsequent updating of the network parameters. After the end of the training process, the figure displays the ultimate validation accuracy and the rationale for the termination of training, which is the completion of the maximum number of epochs (30). After this training process, the validation accuracy reaches 90.38% in the 10th epoch after 150 iterations.

The loss function or cross-entropy loss diagram illustrates the loss received on every mini-batch. Both the actual curve and its smoothed form are observable. Furthermore, we can see the loss curve on the validation dataset. The cross-entropy loss reaches 0.38 in the 27th epoch after 400 iterations.

Figure 12a exhibits a table created by the MATLAB software when we use the 'trainNetwork' function. It displays various training metrics every 50 iterations until 450 iterations according to the progress of the network's training. The max validation accuracy is 90.38% in the 10th epoch after 150 iterations and at the fourth second of elapsed time.

The minimum validation loss is 0.3830 in the 27th epoch after 400 iterations and at the sixth second of elapsed time. Moreover, the test accuracy reaches 0.907.

Figure 12b presents the CM for the test data. The dataset used for testing consists of 54 instances. Out of these, the correctly predicted classes are 49, and the misclassified classes are 5.

- We carry out an identical process for both the SGDM optimizer and the RMSprop opt. The findings are presented in the following tables and figures.

B. SGDM optimization

After conducting a thorough investigation using a range of criteria as previously described, we have identified the magnitudes of parameters that result in the most favorable results.

The adjustable hyperparameters are:
Momentum
Learning rate
The results obtained by employing SGDM optimizer are:

- Best validation accuracy: 82.7%
- Best test accuracy: 81%
- Momentum: 0.9
- Learning rate: 0.00008

Experimentation outcomes:
The magnitudes of the parameters that provide the optimum results are summarized in Table 9.

Table 9. Training option parameters for Adam, SGDM, and RMSprop.

Adam Training Options	SGDM Training Options	RMSprop Training Options
Gradient Decay Factor: 0.999	Momentum: 0.900	Squared Gradient Decay Factor: 0.990
Squared Gradient Decay Factor: 0.999	Initial Learning Rate: 0.0100	Epsilon: 1.0000×10^{-8}
Epsilon: 1.0000×10^{-8}	Learning Rate Schedule: 'piecewise'	Initial Learning Rate: 3.0000×10^{-4}
Initial Learning Rate: 1.0000×10^{-3}	Learning Rate Drop Factor: 0.2000	Learning Rate Schedule: 'none'
Learn Rate Schedule: 'none'	Learning Rate Drop Period: 5	Learning Rate Drop Factor: 0.1000
Learning Rate Drop Factor: 0.1000	L2 Regularization: 1.0000×10^{-4}	Learning Rate Drop Period: 10
Learning Rate Drop Period: 10	Gradient Threshold Method: 'l2norm'	L2 Regularization: 1.0000×10^{-4}
L2 Regularization: 1.0000×10^{-4}	Gradient Threshold: Inf	Gradient Threshold Method: 'l2norm'
Gradient Threshold Method: 'l2norm'	Mini-Batch Size: 64	Gradient Threshold: Inf
Gradient Threshold: Inf	Max Epochs: 100	Mini-Batch Size: 64
Mini-Batch Size: 64		Max Epochs: 20
Max Epochs: 30		

We can monitor the progress of the network's training validating and testing in Figures 13 and 14a and the confusion matrix in Figure 14b.

The max validation accuracy is 82.69% in the 17th epoch after 50 iterations and at the third second of elapsed time.

The minimum validation loss is 0.5371 in the 34th epoch after 100 iterations and at the third second of elapsed time. Moreover, the test accuracy reaches the 0.814.

Figure 14b presents the CM for the test data. The dataset used for testing consists of 54 instances. Out of these, the correctly predicted classes are 44, and the misclassified classes are 10.

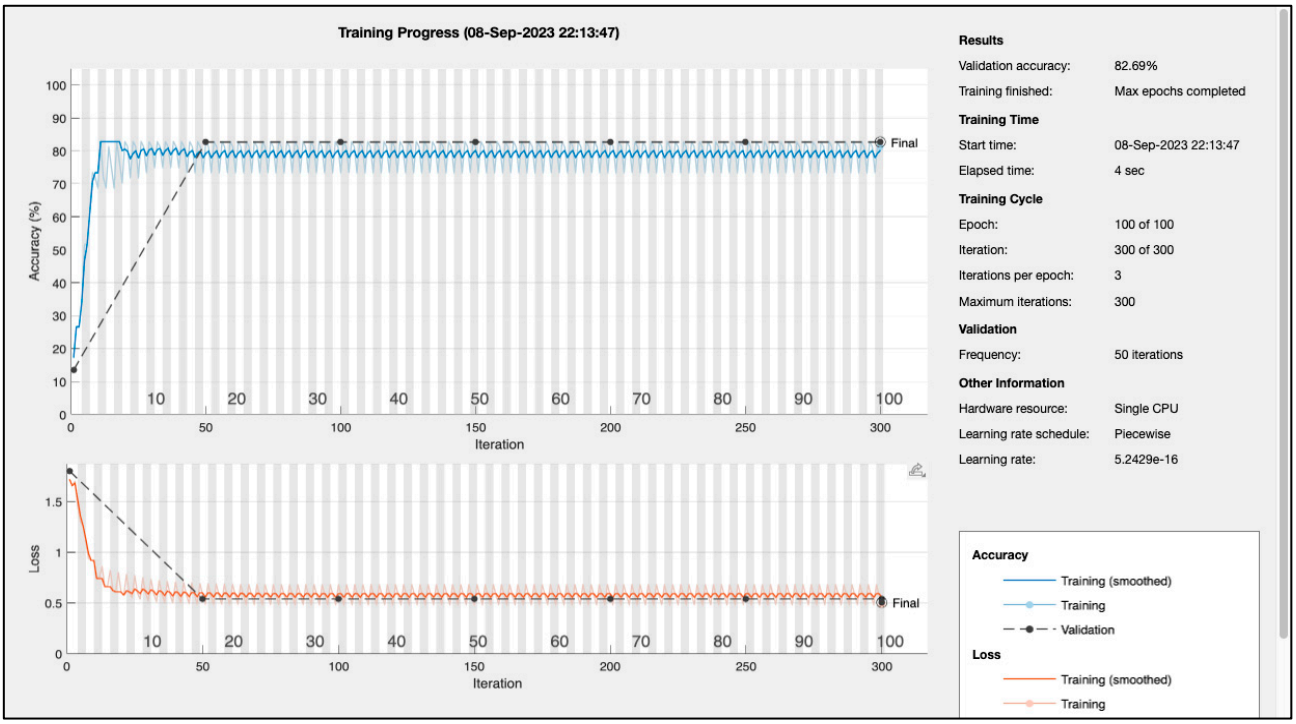
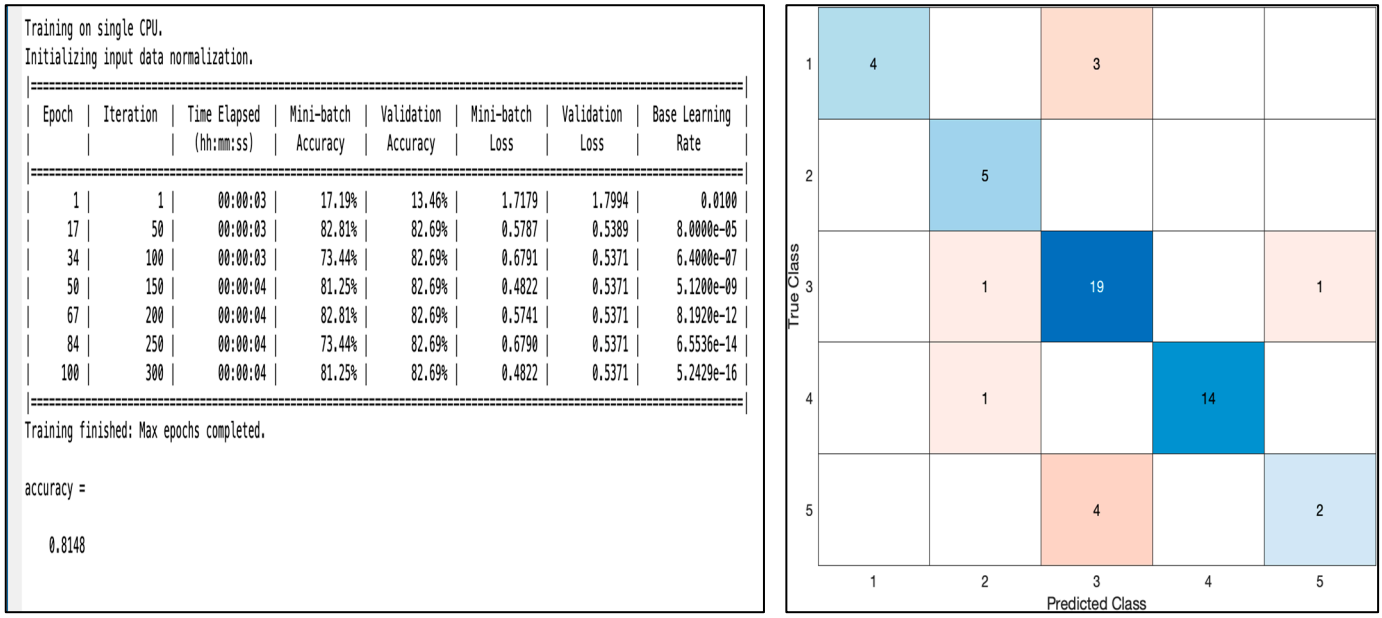


Figure 13. Training process for SGDM.



(**a**) (**b**)

Figure 14. (**a**) Progress of the network's training for SGDM; (**b**) test confusion matrix for SGDM.

C. RMSprop

We train the ANN using the RMSprop solver and set its properties.
The adjustable hyperparameters are:
Squared Gradient Decay Factor
Learning rate
Gradient Decay Factor (controls the strength of the L2 regularization factor)
Epsilon (ε): A very small value added to prevent dividing by zero when updating the parameters; typically, values are 1×10^{-7} or 1×10^{-8}.

Experimentation outcomes:

At the first training of the model, we observe an overfitting with these values:

- Validation accuracy: 67.31%
- Test accuracy: 0.75
- Momentum: 0.99
- Learning rate: 0.0003

In order to prevent overfitting, we adjust the momentum and learning rate hyperparameters. Firstly, we change the Gradient Decay Factor that controls the strength of L2 regularization to the value 0.999. After many training procedures with different sets of hyperparameters, we select those that provide the best accuracy to the model. The magnitudes of the parameters that provide the optimum results are illustrated in Table 9.

The best results obtained by employing the RMSprop optimizer are:

- Best validation accuracy: 86%
- Best test accuracy: 83%
- Momentum: 0.999
- Learning rate: 0.001

We can observe the progress of the network's training validating and testing in Figures 15 and 16a and the confusion matrix in Figure 16b.

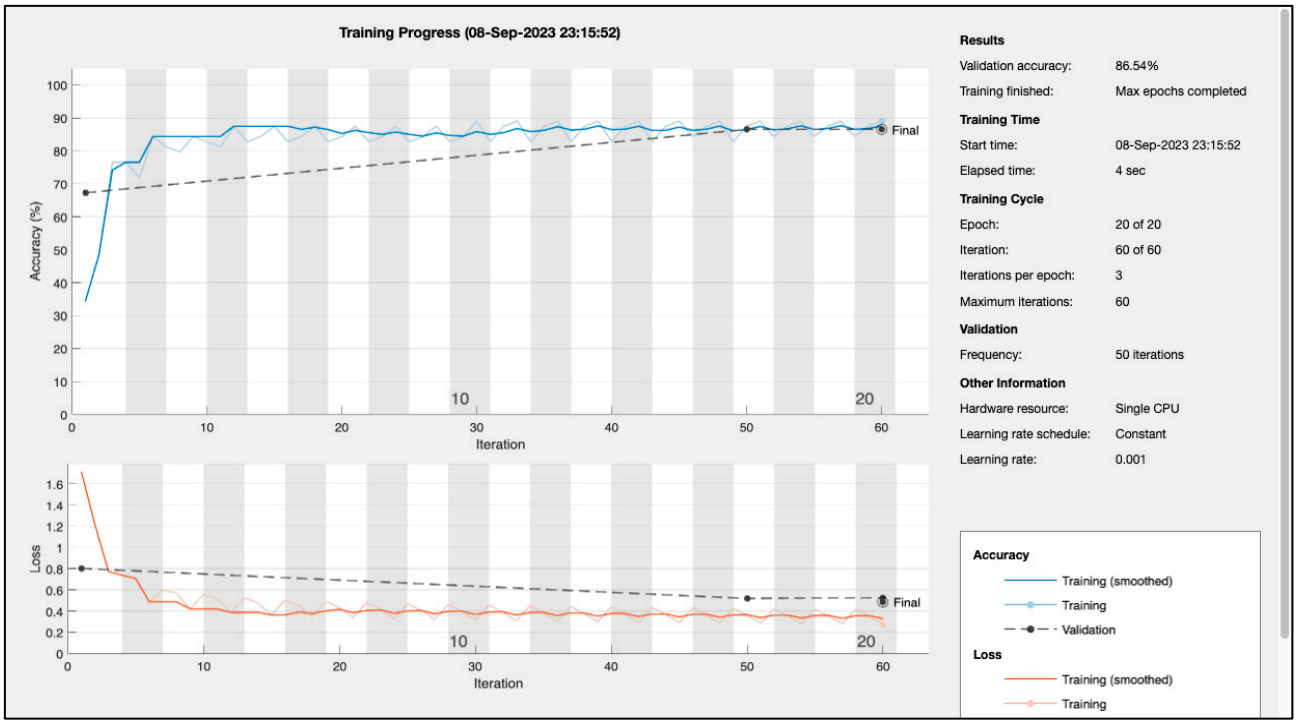
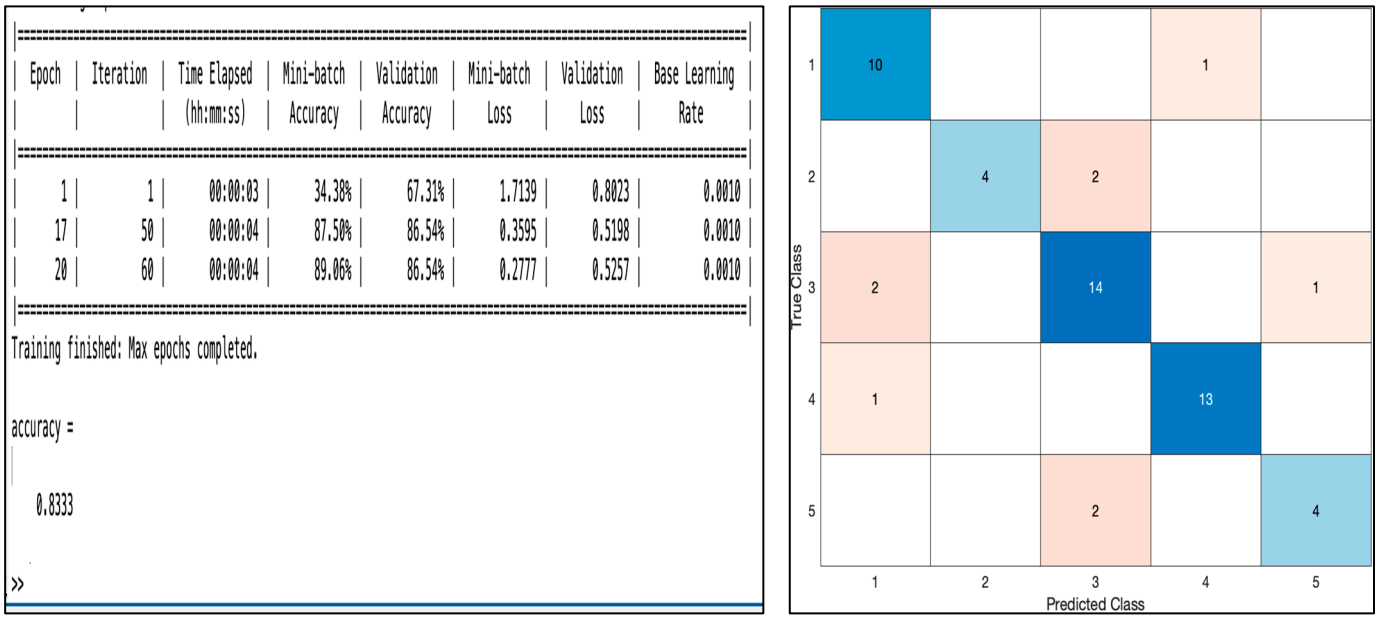


Figure 15. The training process for RMSprop.

The max validation accuracy is 86.54% in the 17th epoch after 50 iterations and at the fourth second of elapsed time.

The minimum validation loss is 0.5198 in the 17th epoch after 50 iterations and at the fourth second of elapsed time. Moreover, the test accuracy reaches 0.814.

Figure 16b presents the CM for the test data. The dataset used for testing consists of 54 instances. Out of these, the correctly predicted classes are 45, and the misclassified classes are 9.



(**a**) (**b**)

Figure 16. (**a**) Training progress for RMSprop; (**b**) test confusion matrix for RMSprop.

Table 10 illustrates the validation and testing accuracy of the Adam, SGDM, and RMSprop algorithms.

Table 10. Validation and training accuracy for Adam, SGDM, and RMSprop.

Algorithm	Validation Accuracy	Testing Accuracy
Adam	90.4%	0.907
SGDM	82%	0.814
RMSprop	86%	0.833

3. Results and Analysis

This manuscript proposes a method to achieve a MLNN with the lowest architectural complexity while indicating a high prediction accuracy for PTFD. The MATLAB R2023a software is applied to develop this model.

Firstly, the authors train five classifiers in the CLA to choose the most effective classification model. Table 5 employs the results from the CLA; it shows that ANN is the classifier with the highest performance metrics. It preserves the highest accuracy, the lowest total cost, the fastest prediction speed, the shortest training time, and the most competent model size in Kb. The ANN was expected to be the best classifier due to the simplicity of our classification problem and the limited size of the dataset, including the transformer gases that we have provided.

Secondly, we conducted an experiment using ExpM, training 30 ANNs under different initial states. Based on the ExpM result table, we selected the five models with the most increased validation accuracy. We then executed tests on these five ANNs using the ExpM pp, manually training and ultimately selecting the ANN with the best performance. As we can observe from Table 2, the model with the best accuracy has the following parameters:

- Number of layers: 3;
- Layers size: 100, 50, 5;
- Activation function: Tanh.

The next step is to tune the hyperparameters. The literature provides minimal information on the optimal tuning of hyperparameters. First, we systematically examine various values for each hyperparameter using the grid, random, and Bayesian search techniques to

identify the optimal combination of hyperparameters that maximizes the performance on the validation set (search strategy). In the CLA, we determine the range of values that each hyperparameter can take. For each hyperparameter setting, we train the neural network and assess its performance by measuring the validation accuracy and loss. Furthermore, the ANN's performance was assessed by employing an ROC curve, a confusion matrix, and a minimum classification error plot of the different search methods. As we can observe from these diagrams and from Table 5, the most accurate method is grid search, with a validation accuracy of 84.57% and test accuracy of 86%.

According to Figures 6 and 7, we can conclude that grid search results in a more nominal classification error. Hence, it has better accuracy than Bayesian optimization, as we have noticed from the confusion matrix and ROC curve. However, the minimum classification error plots indicate that Bayesian has a higher coverage speed as it reaches the minimum error in the sixth iteration. The grid search method is well acknowledged for its simplicity and efficacy in small datasets. Random search and Bayesian optimization have almost the same values of validation accuracy, 82%. Random search proves to be very advantageous in the exploration of innovative hyperparameter values. Bayesian optimization is a very efficient methodology for addressing complex and expansive search domains. Manual search has the lowest accuracy of 80%. It is time-consuming for users and introduces bias due to incorrect assumptions.

In the last section, we analyze Adam, SGDM, and RMSProp. We tune the hyperparameters for the three optimizers according to the best combination of hyperparameters we find with the previous search strategy. From the diagram of the training progress for each optimizer, we can observe the progress of the network's training, validation, and testing.

The outcomes of the Adaptive Learning Rate Method experiment are presented below:

The goal of comparing those three optimizers is to determine which optimization algorithm works best for our problem. The selection of the number of epochs is a part of hyperparameter tuning, combined with other hyperparameters like learning rate and batch size. It is a hyperparameter that is tuned using experimentation and validation to find the optimal value for each algorithm. The Adam, SGDM, and RMSprop algorithms have different convergence rates and dynamics. They update the model's parameters differently, affecting how fast they converge to an optimal solution. Adam converges faster with 20 epochs; similarly, RMSprop converges faster with 30 epochs, while SGDM requires 100 epochs to reach a similar level of performance. Comparing the algorithms under an equal number of epochs might not indicate their actual capabilities and may lead to an unfair evaluation.

A larger batch size can provide a more stable estimate of the gradient but may require more memory. On the other hand, a smaller batch size can introduce more noise but may converge faster.

When starting with a large learning rate of 0.9, the weights associated with each hidden unit will tend to converge toward extremely large positive or very large negative values. The derivatives of the error with regard to the hidden units will approach zero, resulting in no reduction in the error. We try different learning rates with low magnitudes to minimize the stochastic variations in the error caused by the varying gradients across distinct mini-batches.

At the first training of the RMSprop model, we observed an overfitting. In order to prevent overfitting, we adjust the momentum and learning rate hyperparameters. First, we change the Gradient Decay Factor to the value 0.999 to control the strength of L2 regularization. After many training procedures with different values or the momentum and learning rate, we select those that provide the best accuracy to the model. Comparing all the figures, we conclude that Adam introduces the best validation and test accuracy. The results obtained by employing the Adam optimizer are the following:

Best validation accuracy: 90.4%, best test accuracy: 0.907, Gradient Decay Factor = 0.999, learning rate = 0.001.

Based on the outcomes obtained from our experimentation, it can be inferred that the Adam optimization method has several benefits in comparison to other optimization techniques. The technique demonstrates computational efficiency, demands less memory compared to different approaches, and has shown quicker convergence in several instances. Additionally, the model exhibits robustness in selecting hyperparameters and demonstrates satisfactory performance when using default values. Consequently, the final model of our optimization approach is an ANN with the subsequent values:

- Number of layers: 3
- Layers size: 100, 50, 5
- Activation function: Tanh
- Best validation accuracy: 90.4%,
- Best test accuracy: 90.7%

Finally, in our model, we start with a validation accuracy of 80.9% with a test accuracy of 79%, and after the optimization method, the validation accuracy reaches 90.4% with a test accuracy of 90.7%. So, the optimized ANN with 90.7% accuracy is a superior method for PTFD.

This optimization of ANNs in a mathematical way is inspected to overcome the limitations of the IEC-599 standard, the Rogers Ratio Method. The optimized ANN is simulated and tested in MATLAB R2023a—Deep Network Designer. The dataset used in this study consisted of 60 data samples obtained from the IEEE DGA datasets [15] and the dataset used in our previous work [8].

The results are illustrated in Table 11. The accuracy of each approach is calculated using the formula (A) = (Total Samples Correctly Classified)/(Total Samples). Column (1) is given the serial number of every sample. Columns (2) to (6) indicate the concentrations of the gases in ppm. The real faults of the transformers under examination are provided in column (7).

Table 11. Results.

(1) S.N	(2) H_2	(3) CH_4	(4) C_2H_6	(5) C_2H_4	(6) C_2H_2	(7) Real Fault	(8) Rogers	(9) Agr.	(10) ANN	(11) Agr.
1	13	138	83	16	0	F4	F4	√	F4	√
2	762	93	38	54	126	F3	F3	√	F3	√
3	43	116	65	139	0	F4	F4	√	F4	√
4	179	306	73	579	0	F4	F4	√	F4	√
5	57	141	38	51	0	F4	F4	√	F4	√
6	40	8	34	15	0	F4	F4	√	F4	√
7	35	283	121	222	0	F4	U.F	—	F4	√
8	15	159	29	87	0	F4	U.F	—	F5	—
9	55	159	114	493	0	F4	F4	√	F4	√
10	37	123	67	52	0	F4	F4	√	F4	√
11	723	191	110	293	288	F3	F3	√	F3	√
12	7	15	78	58	0	F4	F4	√	F4	√
13	30	51	12	54	0	F4	F4	√	F4	√
14	31	56	33	77	0	F4	F4	√	F4	√
15	109	226	68	192	0	F4	F4	√	F4	√
16	137	279	66	505	0	F4	F4	√	F4	√
17	59	119	36	70	0	F4	F4	√	F4	√
18	151	242	68	232	0	F4	F4	√	F4	√
19	870	77	73	54	14	F2	F2	√	F2	√
20	376	575	146	1092	0	F4	F4	√	F4	√
21	269	1081	347	1725	25	F5	U.F	—	F5	√
22	10	10	8	1	0.01	F4	F4	√	F4	√
23	30	22	14	4.10	0.1	F1	F1	√	F1	√
24	2.90	2	2	0.3	0.1	F1	F1	√	F1	√
25	4	99	82	4	0.1	F4	F4	√	F4	√

Table 11. *Cont.*

(1) S.N	(2) H_2	(3) CH_4	(4) C_2H_6	(5) C_2H_4	(6) C_2H_2	(7) Real Fault	(8) Rogers	(9) Agr.	(10) ANN	(11) Agr.
26	21	34	5	47	62	F3	U.F	——	F3	√
27	50	100	51	305	9	F4	F4	√	F4	√
28	120	17	32	4	23	F1	U.F	——	F1	√
29	980	73	58	12	0.01	F2	F2	√	F2	√
30	1607	615	80	916	1294	F3	U.F	——	F3	√
31	14.7	3.7	10.5	2.7	0.2	F4	U.F	——	F5	——
32	181	262	41	28	0.01	F4	U.F	——	F4	√
33	173	334	172	812.5	33.7	F4	F4	√	F4	√
34	127	107	11	154	224	F3	F4	——	F3	√
35	60	40	6.9	110	70	F3	F4	——	F3	√
36	980	73	58	12	0.01	F2	F3	——	F2	√
37	86	187	136	363	0.01	F4	F3	——	F5	——
38	10	24	372	24	0.01	F4	U.F	——	F4	√
39	260	3	18	2	0.01	F2	F2	√	F2	√
40	586	19	77	6	0.01	F2	F4	——	F2	√
41	20	175	92	14	0.02	F4	F4	√	F4	√
42	801	87	45	62	150	F3	F3	√	F3	√
43	51	99	75	150	0.03	F4	F4	√	F4	√
44	200	298	69	602	0.05	F5	F4	——	F4	——
45	60	154	41	49	0	F4	F4	√	F4	√
46	40	8	34	15	0.2	F1	F4	——	F4	——
47	45	283	158	199	0	F4	U.F	——	F4	√
48	21	159	22	91	0.02	F4	U.F	——	F4	√
49	55	159	128	502	0	F5	F4	——	F5	√
50	41	223	71	52	0	F3	F4	√	F4	——
51	689	203	129	301	362	F2	F3	√	F2	√
52	10	24	95	45	0.02	F4	F4	√	F4	√
53	45	69	7	45	0.003	F5	F4	——	F5	√
54	45	59	45	89	0.01	F4	F4	√	F4	√
55	98	198	70	201	0.04	F4	F4	√	F4	√
56	204	302	57	495	0	F5	F4	——	F5	√
57	45	125	48	82	0	F4	F4	√	F4	√
58	201	256	54	224	0	F4	F4	√	F4	√
59	905	83	81	63	12	F2	F2	√	F2	√
60	402	604	99	998	0.02	F5	F4	——	F5	√

Key: UF = Unidentified fault, √ = fault type diagnosed correctly, —— = fault type not diagnosed correctly, Agr. = agreement with real fault.

In columns (9) and (11), the outcomes of each method are contrasted with the actual faults. In column (8), the faults are obtained using the conventional Rogers Ratio Method. This method cannot predict 11 faults (the primary limitation of the Rogers Method), and there are also 11 incorrect estimations, as we can notice in column (9).

The Rogers Ratio Method's accuracy is 63.3%, which is relatively low.

Column (10) depicts the output of the ANN. The ANN model can accurately predict 38 faults and incorrectly predict 6 instances, as seen in column (11). The level of precision the ANN shows is remarkably high; it obtains 90%. The aforementioned experiments proved that the test accuracy was nearly 90.7%, which is extremely close to 90%.

So, the ANN method's accuracy is 90%, which is remarkably high.

4. Conclusions

This research introduces an approach to improve the diagnostic precision of identifying power transformer faults by employing an optimized ANN. The selection of an optimizer is contingent upon the particular problem, the dataset, and the network's architecture; it often requires experimentation to find the most suitable one. Moreover, the parameters of

an ANN do not have definitive optimum values. The primary objective of ANN designers is to identify a proper learning algorithm that may enhance the generalization capability of an ANN model. Given this context, an intensive search is performed for the best training algorithm and its hyperparameters, including the neurons, learning rate, activation functions, L2 regularization factor, and momentum that may provide the desired results. As we can observe from the confusion matrices and ROC curves of all search methods, the class with the best predicted score (100%) is Class 2, Low Energy Discharge faults. As we can notice from the data samples, the majority of the samples are from Low Energy Discharge defects. Considering this point, we can determine that an increase in the number of samples within a given dataset leads to a higher classification accuracy. The number of samples in our project is 400 gas ratios, but achieving higher accuracies (observable from the literature and confirmed by our experimental findings) requires a sufficient number of training samples. However, obtaining a large amount of data is difficult in the field of PTFD. The limited data availability limits the generalizability of the results in PTFD using ANNs. There is a necessity for more extensive and diverse datasets to train and validate ANN models effectively.

Samples must be taken in order to accomplish better classification accuracy and reduce architectural complexity. Moreover, research focusing on enhancing data quality and managing inconsistencies is crucial for accurate fault diagnosis. Data consistency and the quality of DGA can differ significantly depending on the sensor station and maintenance operations. Another issue which is crucial for future work is multi-fault diagnosis. In this work (and most current studies), we concentrate on single-fault diagnosis, such as detecting a distinct fault type, e.g., a thermal fault. It would be valuable to design ANN-based models competent in diagnosing multiple faults that occur simultaneously or complicated faults.

It can be confirmed that hyperparameters have an essential role in controlling the learning process of a model. Furthermore, the specific values assigned to these hyperparameters have a significant impact on the overall performance of the model. Likewise, the optimization of hyperparameters and the augmentation of data samples are essential factors in improving the accuracy and reducing the complexity of the ANN.

Furthermore, the optimized ANN model is systematically compared to the Rogers Ratio Method, which has an accuracy of only 63.3%. In contrast, the ANN model demonstrated remarkable precision, yielding an accuracy of 90%. Conclusively, these findings highlight the potential of our optimized ANN model as an advanced and accurate solution for transformer health assessment and maintenance, effectively overcoming the limitations associated with the conventional DGA technique, the Rogers Ratio Method.

Author Contributions: Conceptualization, V.R. and S.D.K.; methodology, V.R.; software, V.R.; validation, S.D.K. and P.K.; formal analysis, V.R. and S.D.K.; investigation, V.R. and D.K.; resources, V.R. and D.K.; data curation, V.R. and S.D.K.; writing—original draft preparation, V.R.; writing—review and editing, V.R. and D.K.; visualization, V.R.; supervision, P.K.; project administration, S.D.K. and P.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data are available in a publicly accessible repository that does not issue DOIs. Publicly available datasets were analyzed in this study. This dataset can be found at https://ieee-dataport.org/open-access/botnet-dga-dataset (accessed on 24 January 2023).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Thango, B.A. Dissolved Gas Analysis and Application of Artificial Intelligence Technique for Fault Diagnosis in Power Transformers: A South African Case Study. *Energies* **2022**, *15*, 9030. [CrossRef]
2. Alsuhaibani, S.; Khan, Y.; Beroual, A.; Malik, N.H. A Review of Frequency Response Analysis Methods for Power Transformer Diagnostics. *Energies* **2016**, *9*, 879. [CrossRef]
3. Bhalla, D.; Bansal, R.K.; Gupta, H. Application of Artificial Intelligence Techniques for Dissolved Gas Analysis of Transformers—A Review. *World Acad. Sci. Eng. Technol.* **2010**, *62*, 221–229.

4. Singh, J.; Sood, Y.; Jarial, R. Condition Monitoring of Power Transformers Bibliography Survey. *IEEE Electr. Insul. Mag.* **2008**, *24*, 11–25. [CrossRef]
5. Papadopoulos, A.E.; Psomopoulos, C.S. The contribution of dissolved gas analysis as a diagnostic tool for the evaluation of the corrosive sulphur activity in oil insulated traction transformers. In Proceedings of the 6TH IET Conference on Railway Condition Monitoring (RCM), University of Birmingham, Birmingham, UK, 17–18 September 2014.
6. Koroglu, S. A Case Study on Fault Detection in Power Transformers Using Dissolved Gas Analysis and Electrical Test Methods. *J. Electr. Syst.* **2016**, *12*, 442–459.
7. Siva Sarma, D.V.S.S.; Kalyani, G.N.S. ANN approach for condition monitoring of power transformers using DGA. In Proceedings of the IEEE Region 10 Conference TENCON, Chiang Mai, Thailand, 24 November 2004.
8. Rokani, V.; Kaminaris, S.D. Power Transformers Fault Diagnosis Using AI Techniques. *AIP Conf. Proc.* **2020**, *2307*, 020056. [CrossRef]
9. Barkas, D.A.; Kaminaris, S.D.; Kalkanis, K.K.; Ioannidis, G.C.; Psomopoulos, C.S. Condition Assessment of Power Transformers through DGA Measurements Evaluation Using Adaptive Algorithms and Deep Learning. *Energies* **2023**, *16*, 54. [CrossRef]
10. Patel, D.M.K.; Patel, D.A.M. Simulation and analysis of dga analysis for power transformer using advanced control methods. *Asian J. Converg. Technol.* **2021**, *7*, 102–109. [CrossRef]
11. Ciulavu, C.; Helerea, E. Power Transformer Incipient Faults Monitoring. *Ann. Univ. Craiova-Electr. Eng. Ser.* **2008**, *32*, 72–77.
12. Dhini, A.; Faqih, A.; Kusumoputro, B.; Surjandari, I.; Kusiak, A. Data-driven Fault Diagnosis of Power Transformers using Dissolved Gas Analysis (DGA). *Int. J. Technol.* **2020**, *11*, 388–399. [CrossRef]
13. Rogers, R.R. IEEE and IEC Codes to Interpret Incipient Faults in Transformers, Using Gas in Oil Analysis. *IEEE Trans. Electr. Insul.* **1978**, *5*, 349–354. [CrossRef]
14. IEEE. *C57.104-1991-IEEE Guide for the Interpretation of Gases Generated in Oil-Immersed Transformers*; IEEE: New York, NY, USA, 1992. [CrossRef]
15. IEEE DataPort. Available online: https://ieee-dataport.org/documents/dissolved-gas-data-transformer-oil-fault-diagnosis-power-transformers-membership-degree#files (accessed on 17 April 2023).
16. Hussein, A.R.; Yaacob, M.; Othman, M. Ann Expert System for Diagnosing Faults and Assessing the Quality Insulation Oil of Power Transformer Depending on the DGA Method. *J. Theor. Appl. Inf. Technol.* **2015**, *78*, 278.
17. Sarma, J.J.; Sarma, R. Fault Analysis of High Voltage Power. *Int. J. Adv. Res. Electr. Electron. Instrum. Eng.* **2017**, *6*, 2411–2419. [CrossRef]
18. Zhang, Y.; Tang, Y.; Liu, Y.; Liang, Z. Fault Diagnosis of Transformer Using Artificial Intelligence: A Review. *Front. Energy Res.* **2022**, *10*, 1006474. [CrossRef]
19. Lopes, S.M.d.A.; Flauzino, R.A.; Altafim, R.A.C. Incipient Fault Diagnosis in Power Transformers by Data-Driven Models with over-Sampled Dataset. *Electr. Power Syst. Res.* **2021**, *201*, 107519. [CrossRef]
20. Bishop, C.M. Pattern Recognition and Machine Learning. In *Information Science and Statistics*; Springer: Berlin/Heidelberg, Germany, 2006; ISBN 978-0-387-31073-2.
21. Russell, S.; Norvig, P. Artificial Intelligence: A Modern Approach. In *Always Learning*; Pearson: London, UK, 2016; ISBN 978-1-292-15396-4.
22. Ling-fang, H. Artificial Intelligence. In Proceedings of the 2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE), Singapore, 26–28 February 2010; Volume 4, pp. 575–578.
23. Haykin, S.S. Neural Networks and Learning Machines. In *Pearson International Edition*; Pearson: London, UK, 2009; ISBN 978-0-13-129376-2.
24. Hutter, F.; Lücke, J.; Schmidt-Thieme, L. Beyond Manual Tuning of Hyperparameters. *KI—Kunstl. Intell.* **2015**, *29*, 329–337. [CrossRef]
25. Bartz-beielstein, E.B.T.; Zaefferer, M. *Tuning for Machine and Deep Learning with R*; Springer: Singapore, 2023; ISBN 978-981-19516-9-5.
26. Gridin, I. Hyperparameter Optimization. In *Automated Deep Learning Using Neural Network Intelligence: Develop and Design PyTorch and TensorFlow Models Using Python*; Apress: Berkeley, CA, USA, 2022; pp. 31–110. ISBN 978-1-4842-8149-9.
27. Bi, C.; Tian, Q.; Chen, H.; Meng, X.; Wang, H.; Liu, W.; Jiang, J. Optimizing a Multi-Layer Perceptron Based on an Improved Gray Wolf Algorithm to Identify Plant Diseases. *Mathematics* **2023**, *11*, 3312. [CrossRef]
28. Xu, T.; Gao, Z.; Zhuang, Y. Fault Prediction of Control Clusters Based on an Improved Arithmetic Optimization Algorithm and BP Neural Network. *Mathematics* **2023**, *11*, 2891. [CrossRef]
29. Asimakopoulou, G.; Kontargyri, V.; Tsekouras, G.; Asimakopoulou, F.; Gonos, I.; Stathopulos, I. Artificial Neural Network Optimisation Methodology for the Estimation of the Critical Flashover Voltage on Insulators. *IET Sci. Meas. Technol.* **2009**, *3*, 90–104. [CrossRef]
30. Kussul′, E.M.; Baidyk, T.; Wunsch, D.C. *Neural Networks and Micromechanics*; Springer: Heidelberg/Berlin, Germany; New York, NY, USA, 2010; ISBN 9783642025341.
31. Geman, S.; Bienenstock, E.; Doursat, R. Neural Networks and the Bias/Variance Dilemma. In *Neural Computation*; MIT Press: Cambridge, MA, USA, 1992; Volume 4, pp. 1–58.
32. Girolami, M. *A First Course in Machine Learning*; CRC Press: Boca Raton, FL, USA, 2015; ISBN 978-1-4987-5960-1.
33. Alemu, H.Z.; Wu, W.; Zhao, J. Feedforward Neural Networks with a Hidden Layer Regularization Method. *Symmetry* **2018**, *10*, 525. [CrossRef]

34. Sun, S.; Cao, Z.; Zhu, H.; Zhao, J. A Survey of Optimization Methods from a Machine Learning Perspective. *IEEE Trans. Cybern.* **2020**, *50*, 3668–3681. [CrossRef] [PubMed]
35. Bejani, M.M.; Ghatee, M. *A Systematic Review on Overfitting Control in Shallow and Deep Neural Networks*; Springer: Dordrecht, The Netherlands, 2021; Volume 54, ISBN 0-12-345678-9.
36. Tian, Y.; Zhang, Y.; Zhang, H. Recent Advances in Stochastic Gradient Descent in Deep Learning. *Mathematics* **2023**, *11*, 682. [CrossRef]
37. Kingma, D.P.; Ba, J.L. Adam: A Method for Stochastic Optimization. In Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015—Conference Track Proceedings, San Diego, CA, USA, 7–9 May 2015; pp. 1–15.
38. Beale, M.H.; Hagan, M.T.; Demuth, H.B. Deep Learning Toolbox TM User's Guide How to Contact MathWorks. 2020. Available online: https://www.mathworks.com/help/deeplearning/ (accessed on 17 April 2023).
39. Mathworks. Available online: https://www.mathworks.com/content/dam/mathworks/mathworks-dot-com/campaigns/por-tals/files/machine-learning-resource/machine-learning-with-matlab.pdf (accessed on 1 April 2022).
40. Kim, P. *MATLAB Deep Learning*; Apress: Berkeley, CA, USA, 2017; ISBN 978-1-4842-2844-9.
41. Biswas, S.; Nayak, P.K.; Panigrahi, B.K.; Pradhan, G. An Intelligent Fault Detection and Classification Technique Based on Variational Mode Decomposition-CNN for Transmission Lines Installed with UPFC and Wind Farm. *Electr. Power Syst. Res.* **2023**, *223*, 109526. [CrossRef]
42. Biswas, S.; Nayak, P.K. A New Approach for Protecting TCSC Compensated Transmission Lines Connected to DFIG-Based Wind Farm. *IEEE Trans. Ind. Inf.* **2021**, *17*, 5282–5291. [CrossRef]
43. Bouzar-Benlabiod, L.; Rubin, S.H.; Benaida, A. Optimizing Deep Neural Network Architectures: An Overview. In Proceedings of the 2021 IEEE 22nd International Conference on Information Reuse and Integration for Data Science, Las Vegas, NV, USA, 10–12 August 2021; pp. 25–32. [CrossRef]
44. Kamalov, F.; Leung, H.H. Deep Learning Regularization in Imbalanced Data. In Proceedings of the 2020 IEEE International Conference on Communications, Computing, Cybersecurity, and Informatics, Sharjah, United Arab Emirates, 3–5 November 2020; pp. 17–21. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.