computers

MDPI

*Article*

# Automatic Evaluation of Neural Network Training Results

**Roman Barinov *, Vasiliy Gai, George Kuznetsov and Vladimir Golubenko**

Department of Computing Systems and Technologies, Nizhny Novgorod State Technical University n.a. R.E. Alekseev, st. Minina, 24, 603155 Nizhny Novgorod, Russia
* Correspondence: mrroman152@gmail.com

**Abstract:** This article is dedicated to solving the problem of an insufficient degree of automation of artificial neural network training. Despite the availability of a large number of libraries for training neural networks, machine learning engineers often have to manually control the training process to detect overfitting or underfitting. This article considers the task of automatically estimating neural network training results through an analysis of learning curves. Such analysis allows one to determine one of three possible states of the training process: overfitting, underfitting, and optimal training. We propose several algorithms for extracting feature descriptions from learning curves using mathematical statistics. Further state classification is performed using classical machine learning models. The proposed automatic estimation model serves to improve the degree of automation of neural network training and interpretation of its results, while also taking a step toward constructing self-training models. In most cases when the training process of neural networks leads to overfitting, the developed model determines its onset ahead of the early stopping method by 3–5 epochs.

**Keywords:** learning curves; classification; training assessment automation



**Citation:** Barinov, R.; Gai, V.; Kuznetsov, G.; Golubenko, V. Automatic Evaluation of Neural Network Training Results. *Computers* **2023**, *12*, 26. https://doi.org/10.3390/computers12020026

Academic Editors: Mariofanna Milanova, Xavier Alameda-Pineda and Friedhelm Schwenker

Received: 13 December 2022
Revised: 16 January 2023
Accepted: 17 January 2023
Published: 20 January 2023



## 1. Introduction

Training of an artificial neural network is an iterative process which involves finding parameters of the network in which it achieves the optimal performance [1–3]. There exists a plethora of software designed to solve the task of automatic estimation of model parameters: Auto-sklearn, Auto-WEKA, TPOT, H2O AutoML etc [4–10].

During training, the optimization algorithm tries to minimize the cost function of the network; however, this process does not always proceed smoothly. As such, deep learning engineers try to detect problems occurring during this process as early as possible [11–13]. Missing or ignoring training issues can degrade the performance of a network or render it completely unusable for the task at hand, making retraining necessary and, thus, incurring additional costs in time and resources [14].

This article proposes a method for evaluating the training process of a neural network. Three states of the training process are considered in this work: underfitting, overfitting, and optimal training. Thus, the task of evaluation of the network's state is viewed as a multiclass classification problem [15].

We propose an algorithm for calculating feature descriptions of the loss and accuracy learning curves. The features calculated for these curves differ since the accuracy curve allows normalization of values to the [0; 1] interval, while the loss curve does not.

Use of the proposed approach allows one to increase the degree of automation in their network's training process. The proposed approach allows for a much earlier stopping point when compared to the classic early stopping method, while not allowing overfitting and providing an additional decision reinforcement tool for novice deep learning engineers.

## 2. Overview of Existing Methods for Detecting Model Training Issues

1. Expert method: learning curve—a graph showing the change in a particular metric as the machine learning model is being trained. There are two main types of learning curves.

Optimization curves are learning curves calculated on the basis of the metric by which model parameters are optimized, such as various types of error functions. Performance curves are learning curves calculated on the basis of the metric by which the model will be evaluated and selected later, such as accuracy, precision, recall, or a combined $F_1$-score.

The most popular example of a learning curve is the curve of a model's loss over time. Loss measures a model's error; consequently, a lower loss indicates higher performance of the model. In the long run, loss should decrease over time, indicating that the model is learning.

Another example of a learning curve is the accuracy graph, which reflects the performance of the model, whereby a higher accuracy indicates a more efficiently trained model. A model's accuracy graph rising over time indicates that the model is improving as it accumulates experience. Over time, the graph of the performance metric reaches a plateau, which means that the model is no longer learning, and the performance limit for a given configuration has been reached.

One of the most widely used combinations of metrics is training and validation loss over time. Training loss shows how well the model matches the training data, and validation loss shows how well the model matches new data that were not available during training.

As mentioned above, the main issues that experts seek to identify during model training are model underfitting and overfitting. To detect overfitting and underfitting, machine learning experts apply a set of rules [16–18] according to an understanding of the mathematical process of model training.

Basic case—absence of problems during model training. The training process stops when the trend of the loss function on validation changes from downward to upward.

In the absence of issues in the training phase, the values of the model error function on training data will almost always be lower than those on test data. This means that we should expect some discontinuity between training and validation loss curves. This discontinuity is one of generalization. The optimal case is considered to be one in which the training loss curve decreases until a point of stability, the validation loss curve decreases until a point of stability, or the generalization gap is minimal (almost zero in the ideal case).

If the values of the loss function are high and do not decrease with the number of iterations for both test and training curves, this indicates insufficient complexity of the model used in relation to the data, which leads to model underfitting.

If the learning curve is linear with a derivative close to zero or shows noisy values around some constant high loss value, this indicates that the model failed to reveal patterns during training, or in other words, the fact of learning is almost (or completely) absent.

The values of training and validation loss functions continuously decreasing in the last epochs of training indicate premature cessation of training and the model's inability to learn further.

If the values of the training loss function decrease over time, reaching acceptably low values, but the values of the validation loss function decrease only up to a certain point after which they begin to increase, then it can be concluded that the model reached an overfit.

It is worth noting that the learning curves can identify not only issues associated with the model, but also issues of unrepresentativeness of the data used to train or test the model. A representative dataset proportionally reflects the statistical characteristics of another dataset from the same subject area.

The issue of an unrepresentative training dataset occurs when the data available for training are insufficient for effective training of the model relative to the test dataset. This situation is also possible when the training dataset does not reflect the statistical parameters that are inherent to the data in a given subject area.

In such cases, the training and validation loss values decrease, but there is a large gap between the curves present, which means that the datasets used for training and validation belong to different probability distributions.

The issue of an unrepresentative validation dataset occurs if the dataset used to test the quality of model learning does not provide enough statistical information to assess the generalizability of the model. In such cases, the learning loss curve matches the basic case, while the validation loss curve shows highly noisy values in a region close to the learning loss curve.

It is possible that a validation loss may be much lower than training loss, reflecting the fact that the validation dataset is easier to predict than the training dataset. In this case, the validation dataset is too small and may be widely represented in the training dataset (i.e., there is an overlap between the training and validation datasets).

The expert method of neural network training assessment implies direct control of the network training process by an expert. In turn, this task imposes additional responsibilities and risks on the experts. The expert method depends on competence and experience of a decision maker, as well as increasing development time of a neural network. Additionally, a serious disadvantage of the method is a complete or partial lack of automation.

2. Keras API callbacks. Keras is a commonly used API for building and deploying deep learning models. A Keras API callback is a function that can be executed at different stages of the model's training pipeline. Such functions can be used for various tasks such as controlling the model's learning rate, which can affect the behavior of the training algorithm.

The main drawback of the described method is that the majority of parameters used to detect instances of a model's faulty behavior during training are constants specified in advance by the researchers or a set of rules formulated by researchers in advance. It is worth noting that the various events which can be detected by this method are not classified. An event detected by a Keras callback, be it overfitting or underfitting, is assumed to be hypothetical (with the final decision being made by an expert overseeing the training process) and only valid for that specific stage of model training. Another disadvantage of these callbacks is the lack of premade automatic tools for detection of a supposed model underfit [19].

3. In [20], the authors proposed an approach to accelerate the search for optimal hyperparameters of the neural network through an early stopping of training. The criterion for such stopping is extrapolated learning curves.

In general, the process of extrapolation of a learning curve from an arbitrary number of initial values to final values is as follows: during optimization of the neural network's parameters using the stochastic gradient descent algorithm, regular measurements of the model's performance function are taken. Let $y_{1:n}$ be the observed values of the performance function for the first $n$ iterations of stochastic gradient descent. Then, while observing $y_{1:n}$ values of the performance function, it is necessary to predict the performance $y_m$ at step m, where $m >> n$. In this approach, such a problem is solved using a probabilistic model.

The basic approach is to model the partially observed learning curve $y_{1:n}$ using a set of parametric functions $\{f_1, \ldots, f_K\}$. In turn, each of these parametric functions $f_k$ is described by a set of parameters $\theta_k$. Assuming Gaussian noise $\varepsilon \sim N(0; \sigma^2)$, we can use each function $f_k$ to model network's performance at timestep $t$ as $y_t = f_k(t \mid \theta) + \varepsilon$. The unit observation probability $y_t$ is, thus, defined as

$$p(y_t \Big| \theta_k, \sigma^2) = N(y_t; f_k(t \Big| \theta_k), \sigma^2). \quad (1)$$

The authors of [20] also presented a set of models of parametric curves, the shape of which coincides with the general ideas about the shape of performance curves. Usually, such curves represent an increasing, saturating function. In total, $K = 11$ different parametric functions are presented in the paper. It is worth noting that all the models presented cover only certain aspects of learning curves, but none of them can fully describe all of the possible curves. Therefore, there is a need to combine these models.

The stronger, combined model is a weighted linear combination of simple models:

$$f_{comb}(t|\xi) = \sum_{k=1}^{K} w_k f_k(t|\theta_k), \tag{2}$$

where the new combined vector of parameters,

$$\xi = (w_1, \ldots, w_K, \theta_1, \ldots, \theta_K, \sigma^2), \tag{3}$$

includes the weight $w_k$ for each model, the individual model parameters $\theta_k$, and the noise variance $\sigma^2$; then, $y_t = f_{comb}(t \mid \xi) + \varepsilon$.

Given such a model, it is necessary to model uncertainty and, hence, adopt a Bayesian perspective by predicting $y_m$ values using the Monte Carlo method with Markov chains.

The early stopping method using extrapolation of learning curves provides one with an opportunity to estimate the value of an accuracy function $y_m$ at step $m$; however, the decision about the model being subject to overfitting or the number of training epochs $m$ being insufficient (i.e., underfitting) has to be made by an external mechanism or an expert [21,22]. Consequently, with this approach, the automation of the learning assessment process is only partial and requires additional methods to support itself.

4. The authors of [23] described three criteria for an early stopping of the training process. The first criterion of early stopping is the suggestion to stop training at the point when the generalization loss exceeds a certain threshold. Let $E$ be the target (error) function of the training algorithm. $E_{tr}(t)$ is the training error, an average over the training set measured after epoch t. $E_{va}(t)$ is the error on the validation set. $E_{te}(t)$ is the error on the test set. In real life, the generalization error is usually unknown, and only the validation error $E_{va}(t)$ can be used to estimate it.

The value $E_{opt}(t)$ is defined as the smallest error on the test set obtained before epoch $t$. The generalization error at epoch $t$ is then defined as the relative increase in error on the test set compared to the minimum error at that point in time (as a percentage):

$$GL(t) = 100 \cdot \left( \frac{E_{va}(t)}{E_{opt}(t)} - 1 \right). \tag{4}$$

A high loss of generalization is one obvious possibility for stopping the training, as it directly indicates overfitting. The criterion itself can be described as

$$GL(t) > \alpha. \tag{5}$$

The second early stopping criterion differs from the first in that the generalization error is averaged over $k$ previous epochs and is compared to the minimum error at those $k$ epochs:

$$P_k(t) = 1000 \cdot \left( \frac{\sum_{t'=t-k+1}^{t} E_{tr}(t')}{k \cdot \min_{t'=t-k+1}^{t} E_{tr}(t')} - 1 \right). \tag{6}$$

It is assumed that, with large changes in the error function on a small interval ($k$ of about five epochs), there is a greater chance of subsequently obtaining a smaller value of the generalization error. The second early stopping criterion is formulated as

$$\frac{GL(t)}{P_k(t)} > \alpha. \tag{7}$$

The third criterion of early stopping is where training is halted when the generalization error increases over $s$ consecutive sequences of $k$ epochs. The idea behind this definition is that, according to the authors' assumption, when the validation error increases not only

once, but over $s$ consecutive sequences, such an increase indicates the beginning of final overfitting, no matter how large the actual increase is.

Choosing a particular stopping criterion, in essence, involves a tradeoff between training time and generalization error.

## 3. Proposed Model for Automatic Assessment of Network Training

This paper proposes a model for automatic evaluation of the training results of a neural network (Figure 1) based on quality metrics derived from its training process [24].

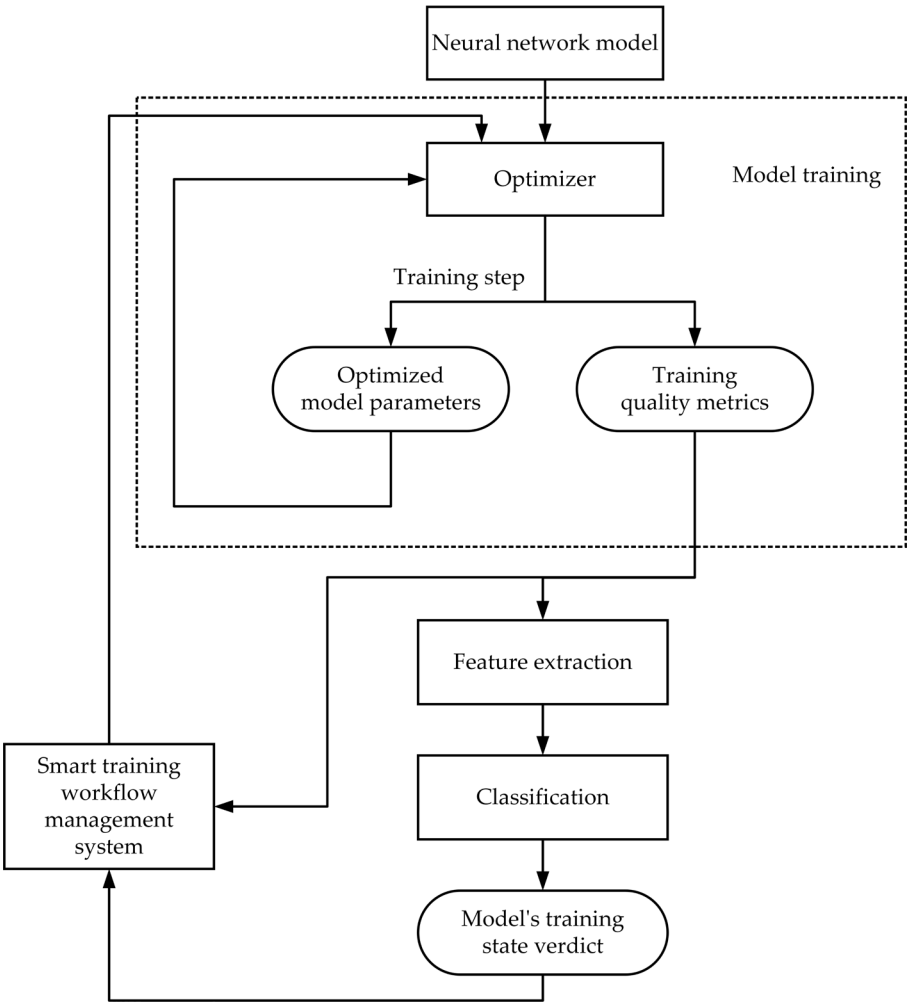
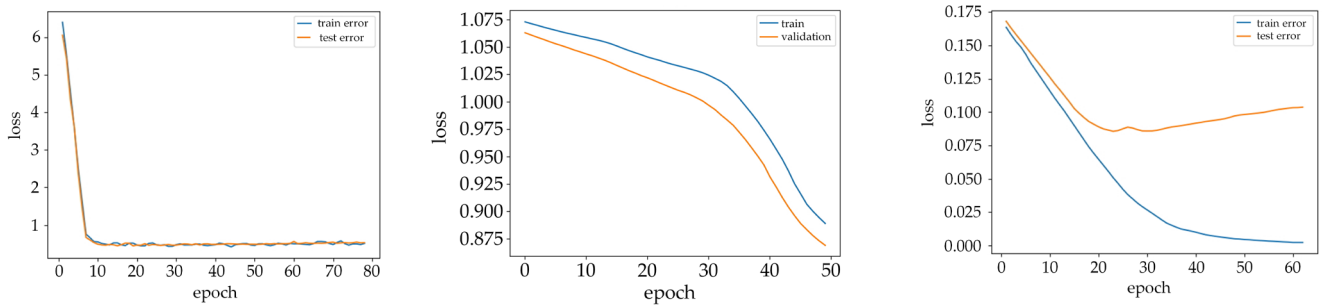



**Figure 1.** A model for automatic evaluation of neural network training results.

At each step of model training, the optimizer calculates given network performance metrics. For classification, the main metrics used are cost functions, such as binary or cross-entropy, and the rate of correct predictions. Thus, it is reasonable to consider these metrics as a time series with a step of one epoch of model training [25].

The metrics obtained from the optimizer are used to calculate a series of features that are passed as input to a classification network. The classifier then determines the state of a training process at a given epoch. Three states are defined for the model output: normal training, underfitting and overfitting. Examples of learning curves representing the dependence of the cost function on the number of training epochs are shown in Figure 2.




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

**Figure 2.** Example of learning curves: (**a**) normal state; (**b**) underfitting; (**c**) overfitting.

The classification result, i.e., an assessment of the learning state at the current epoch, is passed to a management system which, combined with the original performance metrics, determines whether to adjust the learning process, stop it, or continue without changes.

Thus, the task of automatic estimation of neural network training results is reduced to a task of multinomial classification into three classes.

## 4. Data Collection and Sampling

The main criteria for data selection were the types of neural network performance metrics. Only learning curves for the cost and accuracy functions that depend on the training epochs passed were selected [26].

After the data collection phase, the obtained learning curves were converted from graphical representation into time series.

It is worth noting that, due to the fact that training data were collected from multiple sources, it was not possible to collect a single coherent set of data. Therefore, two samples were generated—one including data from cost function learning curves and one including data from accuracy function learning curves.

## 5. Feature Descriptions of Learning Curves

It is useful to divide descriptions of the learning curves into two types: those calculated from the learning curves for both cost and accuracy functions, and those obtained from accuracy functions only.

This distinction is based on the fact that it makes sense to compute some of the proposed attributes using only normalized data. The time series obtained using learning curves of an accuracy function with values in the range [0, 1] are considered normalized. On the other hand, time series obtained from cost function curves are not normalized. We propose to introduce the common features described below for all considered learning curves, regardless of the type of function (cost or accuracy).

1. Standard deviation of the difference between the functions of a given metric on training and validation sets:

$$f_1 = \sigma(F_{t_i} - F_{v_i}), i = \overline{1, N}, \tag{8}$$

where $F_t$ is the metric function for training, $F_v$ is the metric function for validation, and $N$ is the number of training epochs of a neural network model. Using this feature, we can infer how much the values of a given metric differ between training and validation, and which function has a larger average. If the values of the training function nearly match the values of the validation function, it will produce a value of $f_1$ that is close to zero.

2. Standard deviation of the training metric function:

$$f_2 = \sigma(F_{t_i}), i = \overline{1, N}. \tag{9}$$

This feature can be used to understand how much the metric in question changes during training. A feature value close to zero can indicate that the neural network is not learning.

3. Standard deviation of the validation metric function:

$$f_3 = \sigma(F_{v_i}), i = \overline{1, N}. \tag{10}$$

This feature can be used to understand how much the metric in question changes during validation. For cost functions, if the initial value is unsatisfactory with respect to the intended task, the lack of change during validation may indicate that the neural network model has not learned, i.e., is not able to detect patterns

4. Average value of a metric's derivative function at extreme training epochs:

$$f_4 = \frac{1}{N(1-k)} \sum_{i=N(1-k)}^{N} {F_{t_i}}', \tag{11}$$

where $k$ is the percentage of extreme training epochs at which the feature is calculated. The parameter $k$ was empirically set to 10%. The sign of the feature serves as an indicator of a function's tendency toward increasing or decreasing. The value of the derivative can be used to see how much a given metric changes by the end of training, which, together with the sign, gives us an idea of the model's trend towards overfitting or underfitting due to an insufficient number of training epochs.

5. Average value of a metric's derivative function at extreme validation epochs:

$$f_5 = \frac{1}{N(1-k)} \sum_{i=N(1-k)}^{N} {F_{v_i}}'. \tag{12}$$

6. Standard deviation of a metric's function at extreme training epochs:

$$f_6 = \sigma(F_{t_i}), i = \overline{N(1-n), N}, \tag{13}$$

where $n$ is the percentage of extreme training epochs at which the feature is calculated. The parameter $k$ was empirically set to 20%. Using this feature, it is possible to infer how much the metric in question changes during the last training epochs. Values close to zero can indicate stabilization of the metric's function at the end of training.

7. Standard deviation of a metric's function at extreme training epochs:

$$f_7 = \sigma(F_{v_i}), i = \overline{N(1-n), N}. \tag{14}$$

This feature can be used to infer how much the metric in question changes during the most recent validation epochs. A value close to zero may indicate that there is no overfitting.

8. A number of discrete basis functions from active perception theory [27–29] calculated for both metrics on training and validation:

$$f_8 = -F_{t_i} - F_{t_{i+\frac{1}{4}N}} + F_{t_{i+\frac{1}{2}N}} + F_{t_{i+\frac{3}{4}N}}, i = \overline{1, \frac{1}{4}N}, \tag{15}$$

$$f_9 = -F_{t_i} + F_{t_{i+\frac{1}{4}N}} + F_{t_{i+\frac{1}{2}N}} - F_{t_{i+\frac{3}{4}N}}, i = \overline{1, \frac{1}{4}N}, \tag{16}$$

$$f_{10} = F_{t_i} - F_{t_{i+\frac{1}{4}N}} + F_{t_{i+\frac{1}{2}N}} - F_{t_{i+\frac{3}{4}N}}, i = \overline{1, \frac{1}{4}N}, \tag{17}$$

$$f_{11} = -F_{v_i} - F_{v_{i+\frac{1}{4}N}} + F_{v_{i+\frac{1}{2}N}} + F_{v_{i+\frac{3}{4}N}}, i = \overline{1, \frac{1}{4}N}, \tag{18}$$

$$f_{12} = -F_{v_i} + F_{v_{i+\frac{1}{4}N}} + F_{v_{i+\frac{1}{2}N}} - F_{v_{i+\frac{3}{4}N}}, i = \overline{1, \frac{1}{4}N}, \tag{19}$$

$$f_{13} = F_{v_i} - F_{v_{i+\frac{1}{4}N}} + F_{v_{i+\frac{1}{2}N}} - F_{v_{i+\frac{3}{4}N}}, i = \overline{1, \frac{1}{4}N}. \tag{20}$$

These features provide a relationship between the time series for which they are calculated. For example, features $f_8$ and $f_{11}$, which are calculated during the training and validation phases, respectively, show the tendency of learning curves to increase or decrease.

We also propose to introduce the following features for learning curves of the accuracy function specifically.

1. Difference between the initial and final values of the accuracy function during training:

$$f_{14} = F_{t_N} - F_{t_1}. \tag{21}$$

In addition to the standard deviation over the whole period of training, the value of this feature can be used to indicate how much the accuracy function has changed. A value close to zero can be interpreted as a lack of learning. More importantly, a positive sign for this feature's value shows a trend toward improvement in the accuracy metric, and a negative sign shows a trend toward its deterioration.

2. Difference between the initial and final values of the accuracy function during validation:

$$f_{15} = F_{v_N} - F_{v_1}, \tag{22}$$

interpreted similar to the previous feature.

3. Difference between final values of the accuracy function for training and validation:

$$f_{16} = F_{t_N} - F_{v_N}. \tag{23}$$

This feature can be used to determine how different the accuracy metrics are for training and validation at the end of training. With the value of the training accuracy function close to one, a value of around zero is an indication that the training was performed successfully.

4. Maximum value of the training accuracy function:

$$f_{17} = \max(F_t). \tag{24}$$

5. Maximum value of the validation accuracy function:

$$f_{18} = \max(F_v). \tag{25}$$

6. Difference between the initial and final values of the accuracy function at extreme training epochs:

$$f_{19} = F_{t_N} - F_{t_{N(1-n)}}. \tag{26}$$

The parameter $n$ was empirically set to 20%. A near-zero value of this feature, together with a near-zero standard deviation for the last training epochs, may indicate a stabilization of the accuracy function's values.

7. Difference between the initial and final values of the accuracy function at extreme validation epochs:

$$f_{20} = F_{v_N} - F_{v_{N(1-n)}}. \tag{27}$$

A near-zero value for this feature, together with a near-zero standard deviation for the last validation epochs, can indicate stabilization of the accuracy function for validation. If the value of the accuracy function is close to one, it can be used as an indicator that the learning process was performed successfully. If the value of the accuracy function is close to zero for validation, we can say that the model has not trained enough or that it is now in a stabilized overfit.

8. The area under the curve of the accuracy function for learning/validation ($f_{21}$). A near-zero value of this feature may indicate underfitting of the neural network.

9. The area under the learning curve of the accuracy function on validation ($f_{22}$). A near-zero value can indicate underfitting of the neural network, provided a maintained upward trend of the accuracy function, as well as overfitting in situations where the derivative of the accuracy function is negative for the last stages of validation.

Thus, the feature descriptions of learning curves consist of 13 features for the loss functions and 22 features for the accuracy function.

## 6. Classifying the State of the Model

On the basis of the generated data samples and their feature descriptions, we decided to create a classification algorithm based on two independent classifiers (Figure 3).

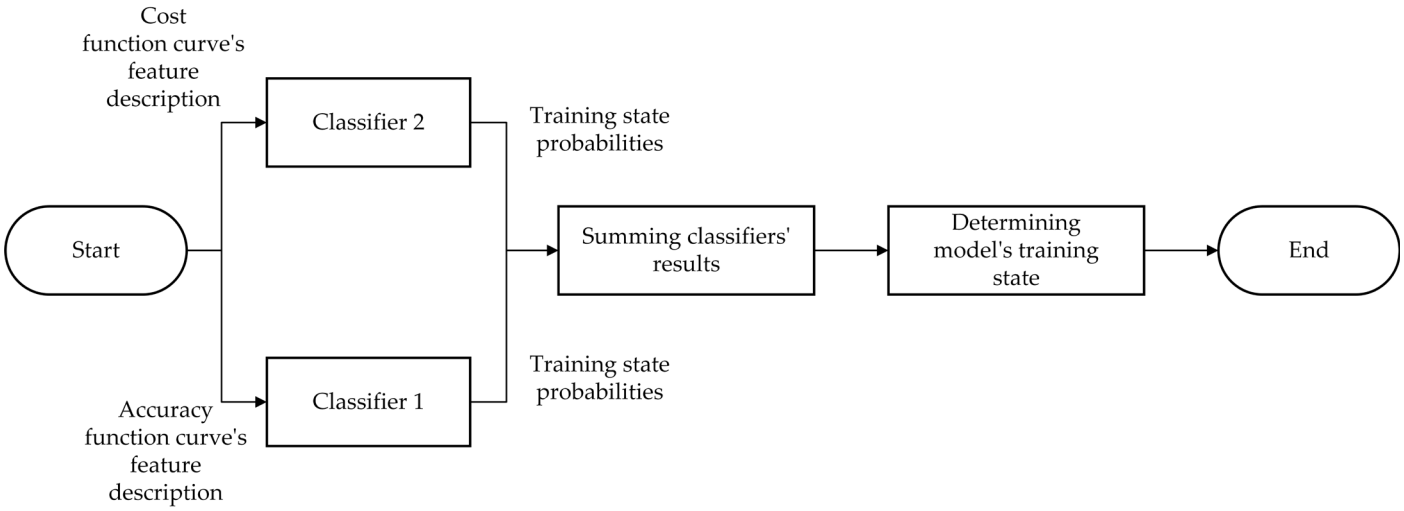



**Figure 3.** Training state classification algorithm.

This algorithm takes learning curve feature descriptions of both cost and accuracy functions obtained during the feature extraction process as input.

Next, two classifiers determine the probabilities that the learning curve feature descriptions indicate one of the three training states: normal training, underfitting, and overfitting.

After this, the maximum value across obtained coefficients is selected, and a decision about the state of model training is made. Thus, the classification algorithm consists of three main stages:

1. Direct classification of input feature descriptions of learning curves by two independent classifiers.
2. Combination of classification results.
3. Determination of the model's state.

## 7. Results

In this selection experiment, after the feature extraction process and the SMOTE [30–33] data augmentation algorithm, we obtained the following distribution of training samples: cost function learning curve data sample size—387 objects (129 objects for each class); accuracy function learning curve data sample size—225 objects (75 objects for each class). The experiment consisted of performing cross-validation (Table 1).

**Table 1.** Classifiers' performance quality metrics.

| **Datasets** | **Learning Curves' Cost Functions** | | | **Learning Curves' Accuracy Functions** | | |
|---|---|---|---|---|---|---|
| **Models** | **Accuracy** | **Precision** | **Recall** | **Accuracy** | **Precision** | **Recall** |
| Decision Tree | 0.734 | 0.733 | 0.734 | 0.791 | 0.795 | 0.791 |
| SVC (OvO) | 0.336 | 0.554 | 0.336 | 0.640 | 0.701 | 0.640 |
| SVC (OvO, polynomial features) | 0.337 | 0.328 | 0.337 | 0.724 | 0.719 | 0.724 |
| SVC (OvA) | 0.354 | 0.392 | 0.408 | 0.782 | 0.806 | 0.796 |
| SVC (OvA, polynomial features) | 0.440 | 0.417 | 0.411 | 0.810 | 0.824 | 0.822 |
| K-neighbors | 0.612 | 0.615 | 0.612 | 0.769 | 0.770 | 0.769 |
| Logistic regression | 0.292 | 0.293 | 0.292 | 0.649 | 0.701 | 0.646 |
| Gradient boosting | 0.780 | 0.789 | 0.788 | 0.879 | 0.872 | 0.872 |
| Random forest | 0.860 | 0.862 | 0.860 | 0.880 | 0.876 | 0.876 |

From the obtained results, it is clear that, with respect to selected performance metrics, the most efficient classifier models for determining the state of the training process were the random forest classifier models.

To conduct a computational experiment comparing the proposed model with its counterparts, it was decided to create and train 20 artificial neural networks to solve a classification problem.

The criterion by which the models were compared with each other was the average value of differences in the number of training epochs after which the training process was stopped. In the case of the proposed model, the epoch at which training is halted was the epoch at which the model first outputs "overfit".

$$Q = \frac{1}{M}\sum_{i=1}^{M} n_m^{(i)} - n_a^{(i)}, \tag{28}$$

where $M$ is the number of neural network models, $n_m$ is the index of the last epoch computed by the proposed model, and $n_a$ is the last epoch computed by the counterpart.

For each neural network model, a fixed maximum of 100 training epochs was specified. The system's performance was compared to the "early stopping" method provided by the Keras API, without any manual configuration on our part.

The obtained results showed a value of $Q = -0.15$ for API Keras callbacks. The negative and near-zero value of the $Q$ parameter indicates that, in most cases, the proposed model detected overfitting simultaneously with the "early stopping" method of API Keras and, occasionally, 3–5 epochs ahead of it.

However, it is worth noting that the API Keras callback method does not provide information to interpret the training results, unlike the proposed model.

Figures 4–6 show examples of the results of the computational experiment. The vertical line indicates the epoch at which the training process was stopped by the "early stopping" API Keras method.

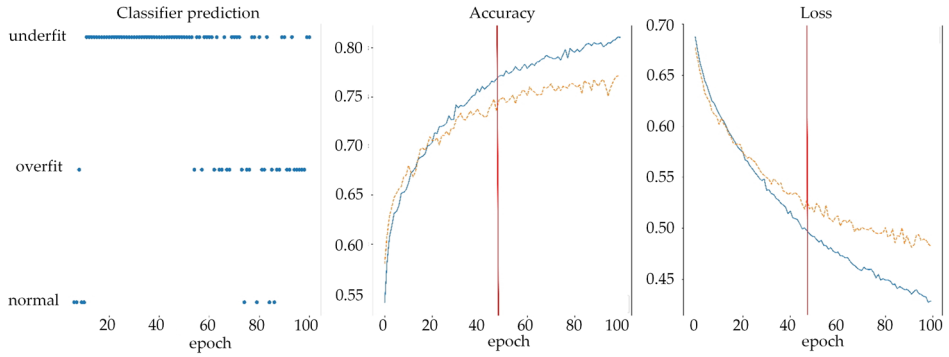



**Figure 4.** Results of computational experiment (overfitting; orange line—validation accuracy, blue line—training accuracy).

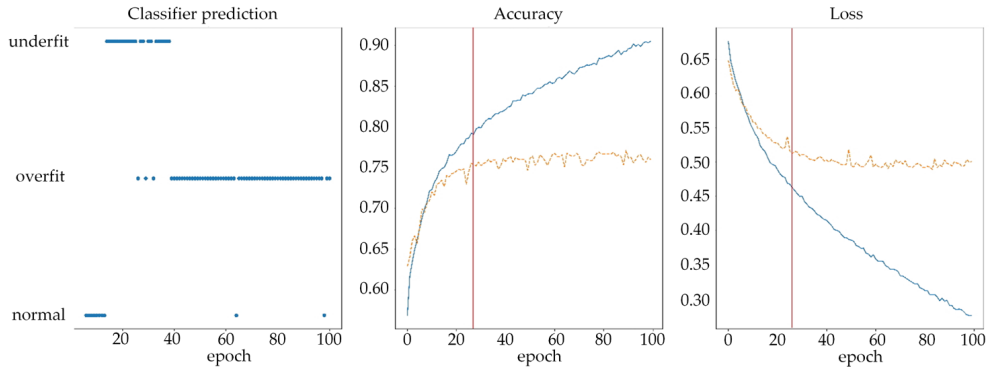



**Figure 5.** Results of computational experiment (overfitting).

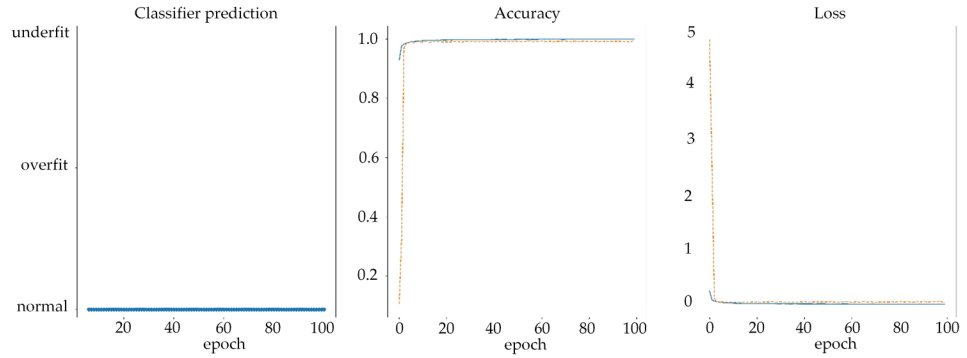



**Figure 6.** Results of computational experiment (no overfitting).

## 8. Discussion

We reviewed existing methods for detecting training issues in neural networks [34]. As a result, a number of conclusions could be made about the shortcomings of the reviewed methods. We then proposed a model for automatic evaluation of neural network training results.

This approach not only allows one to automate the process of identifying issues during training, but also provides a toolkit that allows nonexperts to train deep learning models without the need for additional consultation on the interpretation of their results.

The computational experiment showed that the performance metrics of the training state classification system were comparable to those of its counterparts.

In the future, the following improvements of the model are planned:

1. Developing a smart control system for the training process which allows one to change the optimizer's configuration during model training to influence its behavior.

2. Extending the set of models supported by the proposed method.
3. Developing an algorithm to identify complexity issues in a network's architecture for a particular dataset using the proposed approach.

**Author Contributions:** Methodology, R.B. and V.G. (Vasiliy Gai); software, V.G. (Vladimir Golubenko) and G.K.; data preparation, R.B. and G.K.; analysis and interpretation of results, R.B. and V.G. (Vasiliy Gai); draft manuscript preparation, R.B. and V.G. (Vasiliy Gai). 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 confirming the published results, as well as the source code, can be found in the repository at https://github.com/MrRoman152/learning-curves-analysis.git (accessed on 12 December 2022).

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

## References

1. Guo, R.; Shen, X.; Zhang, X. 3D ROC Histogram: A New ROC Analysis Tool Incorporating Information on Instances. *IEEE Access* **2019**, *7*, 183396–183404. [CrossRef]
2. Jalaeian Zaferani, E.; Teshnehlab, M.; Khodadadian, A.; Heitzinger, C.; Vali, M.; Noii, N.; Wick, T. Hyper-Parameter Optimization of Stacked Asymmetric Auto-Encoders for Automatic Personality Traits Perception. *Sensors* **2022**, *22*, 6206. [CrossRef] [PubMed]
3. Yotov, K.; Hadzhikolev, E.; Hadzhikoleva, S.; Cheresharov, S. Finding the Optimal Topology of an Approximating Neural Network. *Mathematics* **2023**, *11*, 217. [CrossRef]
4. Wever, M.; Tornede, A.; Mohr, F.; Hüllermeier, E. AutoML for Multi-Label Classification: Overview and Empirical Evaluation. *IEEE Trans. Pattern Anal. Mach. Intell.* **2021**, *43*, 3037–3054. [CrossRef] [PubMed]
5. Xin, H.; Kaiyong, Z.; Xiaowen, C. AutoML: A survey of the state-of-the-art. *Knowl. Based Syst.* **2021**, *212*, 106622.
6. Kotthoff, L.; Thornton, C.; Hoos, H.; Hutter, F.; Leyton-Brown, K. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *J. Mach. Learn. Res.* **2017**. [CrossRef]
7. Hutter, F.; Kotthoff, L.; Vanschoren, J. *Automated Machine Learning: Methods, Systems, Challenges*; Springer Nature: Berlin, Germany, 2019.
8. Celik, B.; Vanschoren, J. Adaptation Strategies for Automated Machine Learning on Evolving Data. *IEEE Trans. Pattern Anal. Mach. Intell.* **2021**, *43*, 3067–3078. [CrossRef]
9. Alsharef, A.; Sonia; Kumar, K.; Iwendi, C. Time Series Data Modeling Using Advanced Machine Learning and AutoML. *Sustainability* **2022**, *14*, 15292. [CrossRef]
10. Emmert-Streib, F.; Dehmer, M. Evaluation of Regression Models: Model Assessment, Model Selection and Generalization Error. *Mach. Learn. Knowl. Extr.* **2019**, *1*, 521–551. [CrossRef]
11. Chetoui, M.; Akhloufi, M.; Yousefi, B.; Bouattane, E. Explainable COVID-19 Detection on Chest X-rays Using an End-to-End Deep Convolutional Neural Network Architecture. *Big Data Cogn. Comput.* **2021**, *5*, 73. [CrossRef]
12. Algehyne, E.; Jibril, M.; Algehainy, N.; Alamri, O.; Alzahrani, A. Fuzzy Neural Network Expert System with an Improved Gini Index Random Forest-Based Feature Importance Measure Algorithm for Early Diagnosis of Breast Cancer in Saudi Arabia. *Big Data Cogn. Comput.* **2022**, *6*, 13. [CrossRef]
13. Dora, S.; Kasabov, N. Spiking Neural Networks for Computational Intelligence: An Overview. *Big Data Cogn. Comput.* **2021**, *5*, 67. [CrossRef]
14. Frank, M.; Drikakis, D.; Charissis, V. Machine-Learning Methods for Computational Science and Engineering. *Computation* **2020**, *8*, 15. [CrossRef]
15. Huang, Y.-C.; Hung, K.-C.; Lin, J.-C. Automated Machine Learning System for Defect Detection on Cylindrical Metal Surfaces. *Sensors* **2022**, *22*, 9783. [CrossRef] [PubMed]
16. Ghasemian, A.; Hosseinmardi, H.; Clauset, A. Evaluating Overfit and Underfit in Models of Network Community Structure. *IEEE Trans. Knowl. Data Eng.* **2020**, *32*, 1722–1735. [CrossRef]
17. Cho, H.; Kim, Y.; Lee, E.; Choi, D.; Lee, Y.; Rhee, W. Basic Enhancement Strategies When Using Bayesian Optimization for Hyperparameter Tuning of Deep Neural Networks. *IEEE Access* **2020**, *8*, 52588–52608. [CrossRef]
18. Nallakaruppan, M.; Ramalingam, S.; Somayaji, S.; Prathiba, S. Comparative Analysis of Deep Learning Models Used in Impact Analysis of Coronavirus Chest X-ray Imaging. *Biomedicines* **2022**, *10*, 2791. [CrossRef]
19. Gu, Y.; Wylie, B.K.; Boyte, S.P.; Picotte, J.; Howard, D.M.; Smith, K.; Nelson, K.J. An Optimal Sample Data Usage Strategy to Minimize Overfitting and Underfitting Effects in Regression Tree Models Based on Remotely-Sensed Data. *Remote Sens.* **2016**, *8*, 943. [CrossRef]
20. Domhan, T.; Springenberg, J.T.; Hutter, F. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence Buenos Aires, Buenos Aires, Argentina, 25 July 2015.

21. Li, Z.; Kamnitsas, K.; Glocker, B. Analyzing Overfitting Under Class Imbalance in Neural Networks for Image Segmentation. *IEEE Trans. Med. Imaging* **2011**, *40*, 1065–1077. [CrossRef]
22. Qian, L.; Hu, L.; Zhao, L.; Wang, T.; Jiang, R. Sequence-Dropout Block for Reducing Overfitting Problem in Image Classification. *IEEE Access* **2020**, *8*, 62830–62840. [CrossRef]
23. Prechelt, L. Early Stopping—But When? In *Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science*; Montavon, G., Orr, G.B., Müller, K.R., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7700, pp. 53–67.
24. Huo, J.; Gao, Y.; Shi, Y.; Yin, H. Cross-Modal Metric Learning for AUC Optimization. *IEEE Trans. Neural Netw. Learn. Syst.* **2018**, *29*, 4844–4856. [CrossRef] [PubMed]
25. Diaz, G.I.; Fokoue-Nkoutche, A.; Nannicini, G.; Samulowitz, H. An effective algorithm for hyperparameter optimization of neural networks. *IBM J. Res. Dev.* **2017**, *61*, 9:1–9:11. [CrossRef]
26. Kim, D.; Seo, S.B.; Yoo, N.H.; Shin, G. A Study on Sample Size Sensitivity of Factory Manufacturing Dataset for CNN-Based Defective Product Classification. *Computation* **2022**, *10*, 142. [CrossRef]
27. Utrobin, V.A. Elements of the study of image detection. *Trans. NNSTU N. A. R. E. Alekseev* **2010**, *81*, 61–69.
28. Wang, C.; Baratchi, M.; Bäck, T.; Hoos, H.H.; Limmer, S.; Olhofer, M. Towards Time-Series Feature Engineering in Automated Machine Learning for Multi-Step-Ahead Forecasting. *Eng. Proc.* **2022**, *18*, 17. [CrossRef]
29. Leite, D.; Martins, A., Jr.; Rativa, D.; De Oliveira, J.F.L.; Maciel, A.M.A. An Automated Machine Learning Approach for Real-Time Fault Detection and Diagnosis. *Sensors* **2022**, *22*, 6138. [CrossRef] [PubMed]
30. Pradipta, G.A.; Wardoyo, R.; Musdholifah, A.; Sanjaya, I.N.H. Radius-SMOTE: A New Oversampling Technique of Minority Samples Based on Radius Distance for Learning From Imbalanced Data. *IEEE Access* **2021**, *9*, 74763–74777. [CrossRef]
31. Chen, Y.; Chang, R.; Guo, J. Effects of Data Augmentation Method Borderline-SMOTE on Emotion Recognition of EEG Signals Based on Convolutional Neural Network. *IEEE Access* **2021**, *9*, 47491–47502. [CrossRef]
32. Dablain, D.; Krawczyk, B.; Chawla, N.V. DeepSMOTE: Fusing Deep Learning and SMOTE for Imbalanced Data. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**. [CrossRef]
33. GhoshRoy, D.; Alvi, P.A.; Santosh, K. Explainable AI to Predict Male Fertility Using Extreme Gradient Boosting Algorithm with SMOTE. *Electronics* **2023**, *12*, 15. [CrossRef]
34. Kumar, P.; Ali, I.; Kim, D.-G.; Byun, S.-J.; Kim, D.-G.; Pu, Y.-G.; Lee, K.-Y. A Study on the Design Procedure of Re-Configurable Convolutional Neural Network Engine for FPGA-Based Applications. *Electronics* **2022**, *11*, 3883. [CrossRef]