Towards Green AI by reducing training effort of Recurrent Neural Networks using Hyper-Parameter Optimization with dynamic stopping criteria

Vili Podgorelec*, Iztok Fister Jr., Grega Vrbančič
Intelligent Systems Laboratory, Faculty of Electrical Engineering and Computer Science, University of Maribor, Slovenia
email: *vili.podgorelec@um.si

***Abstract*—Neural networks have become a leading model in modern machine learning, able to model even the most complex data. For them to be properly trained, however, a lot of computational resources are required. With the carbon footprint of ever-growing adoption of neural networks in mind, an approach to reduce the required training resources would be very welcome. We designed a new training effort reduction method based on the calculation of area under the normalized loss curve and assessed it on the electricity consumption forecasting problem with the recurrent neural networks. The results show that the proposed method was able to considerably reduce the amount of computational resources, while maintaining the predictive performance, and thus contributing towards the Green AI.**

***Index Terms*—machine learning, neural networks, hyper-parameter optimization, resource optimization, Green AI**

I. Introduction

To meet the challenges of successfully discovering knowledge and intelligently analyzing the ever-increasing amount of data available, researchers have developed increasingly complex Machine Learning (ML) approaches, algorithms and models. Such approaches attempt to relieve data scientists of the increasingly demanding tasks of preprocessing and transforming data. A typical example are Neural Networks (NNs) and Deep Learning (DL) methods, which, through representation learning, largely relieve experts of the time-consuming feature engineering and mapping of data spaces [1]. The trade-off for this type of automation is an immense increase in the complexity of the training process and the computational complexity of such algorithms, which require huge amounts of computing resources to run successfully [2].

Training deep NNs is extremely demanding in terms of resource consumption. With the ever-growing adoption of Artificial Intelligence (AI), its carbon footprint is no longer negligible [3], as AI could soon consume as much energy as a country the size of the Netherlands [4]. So, the question is how to reduce the need for computing resources when training complex ML predictive models without significantly reducing the performance of the trained models.

NNs are trained using non-deterministic, gradient-based iterative algorithms, involving randomized weight initialization, data ordering, and data augmentations [5]. With a large number of iterations, they gradually improve the fit of the trained model with the given training data, thereby reducing the model's error rate. The search space for possible solutions is extremely large and complex, with a large number of local optima [6]. The goal of training is to approach the global optimum, but iterative searches often end up in local optima that do not allow adequate global progress [7]. To avoid stagnation in local optima, the training is performed in several independent runs; before each run, we can change or adjust the parameters that affect the execution of the training algorithm. Because of the stochasticity of the training algorithms, each independent run of training produces a different network with better or worse performance than average [5]. The consumption of computational resources is directly related to the number of training iterations performed. Since we do not necessarily train a better predictive model with each successive run, the total number of training iterations, and thus the need for computational resources, could be reduced by prematurely terminating unpromising runs that are highly unlikely to improve the predictive model. However, care must be taken not to prevent runs that would otherwise lead to an optimal model by stopping training excessively.

In this paper, we present a possible new method for reducing the training effort of NNs by dynamically stopping individual training runs. We present and explain the metric, with the help of which we can assess whether it makes sense to continue with the training process. We tested the new method on the problem of predicting electricity consumption using Recurrent Neural Networks (RNNs).

The main contributions of this paper can be summarized as:

- a brief overview of existing resources optimization approaches in neural networks training,
- a proposed training effort reduction method based on the calculation of area under the normalized loss curve,
- a designed experiment to evaluate the suitability of the proposed training effort reduction method.

The next section of the paper briefly overviews related work on NNs training effort reduction approaches. In Section III, we present all the crucial components that constitute the proposed training effort reduction method, which is explained in detail in Section IV. The proposed method is evaluated on a set of prepared electricity consumption time series data in an experiment, presented in Section V, together with the evaluation of the obtained results. The last section concludes the paper with a summary of our findings.

979-8-3503-8560-1/24/$31.00 ©2024 IEEE

II. Background: the computational complexity of NN training and ways to optimize the resources

Several methods have been proposed to reduce the computational cost of training NNs and thus reduce the environmental impact. In [8] authors designed a ReLU-based multilayer NN that maps feature vectors to a higher dimensional space, reducing training costs as the number of layers increases. Another approach is to reduce the computational complexity of deep NNs by optimising and adapting them for resource-constrained platforms [9]. Implicit regularization techniques, such as using more aggressive learning rates and optimizing hyper-parameters, can also lead to faster convergence rates and computational savings [10]. Furthermore, dynamic data reduction techniques can be used to reduce the amount of work required during training, thereby reducing costs and environmental impact [11].

Frequently, various Hyper-parameter Optimization (HPO) approaches are used in training NNs, which can improve the predictive performance of NNs to be competitive with human experts, but at the expense of increased computational costs. A prevalent approach to speed up the training process with HPO is early training termination using some kind of stopping criteria [12]. It is not only a well-known regularization technique, but also provides an excellent mechanism to avoid wasting resources when training is not going in the right direction. For example, in [13] the authors mimicked the early termination of bad runs with the help of a probabilistic model that extrapolates performance from the first part of a learning curve to its remainder, enabling them to automatically identify and terminate bad runs to save time.

III. Materials and methods

A. Hyper-parameter Optimization in Machine Learning

In general, complex algorithms use various hyper-parameters, which determine their behavior and thus contribute to their result. HPO is the process of finding optimal values of hyper-parameters [14], such that will provide the most desirable result. It is used for various purposes where optimal values of the parameters are not known in advance or cannot be simply determined analytically. In ML, we use HPO to determine the values of hyper-parameters, which significantly contribute to the predictive performance of the trained predictive model. Thus, the set of possible values of hyper-parameters represents the search space, within which, using the search algorithm or optimization method, we are looking for such a combination of value settings (hyper-parameter configuration), which will enable the selected ML algorithm to build the best possible prediction model according to the given evaluation function [15]. The search for optimal values is usually iterative, where the setting of hyper-parameter values is being constantly changed and improved throughout iterations. The values can be either continuous, discrete, binary, or categorical and have different constraints, so their optimization is often a complex constrained optimization problem [16].

In general, for a HPO problem, the aim is to obtain:

$$x^* = \arg\min_{x \in X} f(x), \tag{1}$$

where $f(x)$ is the objective function to be minimized, such as the root mean squared error, x^* is the hyper-parameter configuration that produces the optimum value of $f(x)$, and a hyper-parameter x can take any value in the search space X.

In supervised ML, such as regression, the goal is to obtain an optimal predictive model function to minimize the cost function that models the error between the predicted output and original data [15]. If the mapping between input values and outputs in the predictive model is denoted as a model function f from a set of possible models F, the optimal predictive model can be obtained by [17]:

$$f^* = \arg\min_{f \in F} \frac{1}{n} \mathcal{L}(f(x_i), y_i), \tag{2}$$

where n is the number of training instances, x_i is the feature vector of the i-th instance, y_i is the corresponding actual output, and $\mathcal{L}$ is the cost function value of each sample.

There are different approaches to solve HPO tasks in ML, the most common being various grid and random search approaches [18]. Various advanced metaheuristic algorithms have shown some very good results when optimizing hyper-parameters in deep neural networks recently [14].

B. Random Search

Random Search (RS) and its variations are one of the most common metaheuristics algorithms for optimizing complex stochastic systems whose expected performance under any particular system design is unavailable in close form and must be instead simulated [19]. Thus, if θ represents the set of all possible system designs and $f(x) = \epsilon[Y(x, \xi)]$ refers to the expected system performance under each design $x \in \theta$, then the goal is to solve the optimization problem:

$$min_{x \in \theta} f(x) \tag{3}$$

In contrast to the deterministic methods such as interval analysis and tunneling methods, which topically guarantee asymptotic convergence to the optimum, RS algorithm and its variants ensure convergence in probability. The trade-off between the approaches is in terms of computational complexity [20]. In various studies [21] the RS methods have shown the ability to solve large-scale problems efficiently in a way that is not possible for deterministic algorithms. Whereas it is known that a deterministic method for global optimization is NP-hard [22], the studies show that a stochastic algorithm can be executed in polynomial time [20].

C. RNNs for time series forecasting

There are many methods, techniques and approaches for dealing with electricity consumption forecasting. They can be categorized into three groups: statistical analysis, traditional ML, and deep learning. The DL algorithms, devised to automatically learn complex and highly nonlinear feature representations from time series data, generally outperform the rest

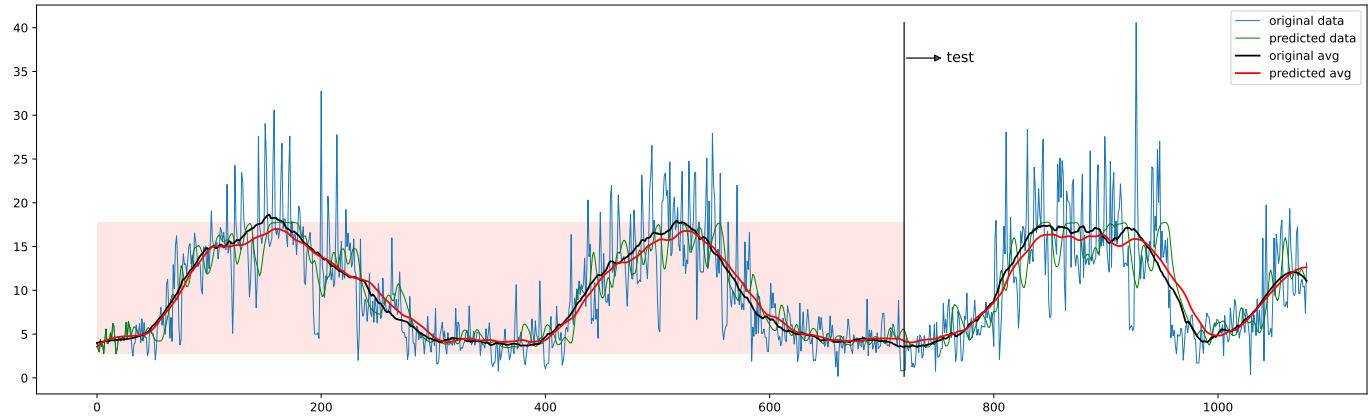

Fig. 1: An example of electricity consumption forecasting using a Recurrent Neural Network trained with the proposed method.

in predictive performance. A prevalent DL model architecture, known as the Recurrent Neural Network (RNN), is typically employed to tackle the challenge of such predictions, by utilizing recurrent units. The basic purpose of the recurrent unit is to compute the output based on the current input and the previous hidden state. The hidden state at some time step t is calculated using the current input x_t and the previous hidden state h_{t-1}, typically represented as $h_t = \sigma(W_h h_{t-1} + W_x x_t + b)$, where σ is the activation function, W_h and W_x are weight matrices, and b is the bias vector [23]. RNNs have demonstrated their efficacy in addressing various problems in time series forecasting (an example is presented in Fig. 1). However, a common issue arises due to the substantial increase in the number of inputs in RNNs, leading to a more complex weight adjustment process. This often results in the vanishing gradient problem [24], a significant obstacle in the optimization of DL models. The issue of capturing long-term dependencies in the data and alleviating the vanishing gradient problem was addressed with LSTM units, which introduced memory cells and gating mechanisms to control the flow of information. LSTM incorporates three different gates (input, forget, and output) to regulate information flow and prevent the vanishing gradient problem. The input gate decides which information to update, the forget gate decides which information to discard, and the output gate decides which information to output [23]. Because of their complex internal mechanism, however, the LSTMs tend to be computationally very expensive.

D. Electricity consumption dataset

The data for the electricity consumption dataset was collected from 15 distinct households in Slovenia. These real-world measurements represent household power consumption, captured at 15-minute intervals and then aggregated into daily consumption. Each interval corresponds to the power consumed in kilowatts per hour (kWh). The duration of power consumption measurement varies between 1.080 and 1.827 days, depending on the specific household. Some measurements (not more than 5% overall) might be missing.

E. Preparation of training data

For the experiment, presented in this paper, we used a univariate time series (electricity consumption measurements only) without external data. First, the missing values were replaced using the linear interpolation imputation method. The captured 15-minute interval measurements were then aggregated into a daily consumption.

The first 720 values (i.e. 720 days of data or approximately 2 years) were used to train the model with a sliding window size of 30 days (one month) with a step size of 1. An instance for the ML algorithm is thus composed of the 30 successive daily consumption values as input, while the following value in the time series is used as output. For the testing purposes, predictions for all the remaining values were made using the trained model to determine the accuracy of the predictions.

F. Evaluation of trained forecasting models

We used Mean Absolute Error (MAE) and Mean Root Squared Error (MRSE) metrics to evaluate the prediction models. The evaluations were performed using MAE and MRSE on test data over all 15 time series. The mean values of MAE and MRSE over all 15 time series in the dataset are generally reported.

IV. The Proposed Training Effort Reduction Method

The idea of a dynamic stopping criteria measure, derived from the loss value, which would help us determine whether it makes sense to continue with the training of a NN model, comes from our previous experience with developing, fine tuning, and optimizing various more or less complex deep neural network architectures applied against different domain problems. Commonly, in such processes various combinations of different parameters and architecture decisions are being tested in order to achieve the best possible performance. When such different parameter settings are being tested, most commonly the training loss is being observed during the training in order to identify whether the trained solution progresses as expected. Ideally, we would like to see the loss value at the

beginning of the training process to decrease rather quick and gaining the decrease momentum towards the middle of the training process, while at the end the decrease of the loss is slowed down before it stabilizes, reaching a (local) minimum. Based on the observation of the loss curve during the training process, we should be capable to detect whether the current parameter settings are promising to train the model well or not. Given the specific characteristics of the loss curve, which we are pursuing in the process of training a deep NN model, we would like to be able to assess the expected predictive performance of a fully trained model even before the full training is completed, as early in the training phase as possible and with high reliability.

There are two main factors that commonly indicate a good training process, which could end up in a well-trained model:

- The **absolute loss value**, indicating an amount of errors the model makes on the training data. The lower loss value generally indicates a more accurate model.
- The **trend-line of loss values** throughout the training process. The steepest it is, the better chance the model has to reach a global minimum.

To reliably determine whether it makes sense to continue with a specific training process, we need to consider both factors. The best potential has a model, already showing a rather good performance (represented by low loss value), where the improvement is still considerable (steep enough learning curve). A stagnating low loss will probably not improve significantly when a model is trained further. On the other hand, the improvement of a rather poor performing model has less chance to become an outstanding model as the improvement of an already good performing model.

For this purpose, we defined a composite measure which does not only focuses on the single value at the specific training step (a loss value), but takes into account all the previous ones and in such manner in early training stages tries to identify whether some NN architecture and/or parameter settings are promising or not.

A. *Area under normalized loss curve, AUNL*

The loss function L is the function that is most often used to monitor the performance of a trained NN model during the training process. In order to include preliminary values of the loss function, in the first step of developing the AUNL metric, the values of the loss function were normalized using min-max normalization and mapped to values within the interval [0, 1]. As a result, the normalized value of the loss function in the selected epoch NL_i represents a relative value with respect to the absolute maximum value of the loss function L and in this way indirectly includes information about the previous values of the loss function. Furthermore, by normalizing the loss values all training runs become directly comparable, although their absolute loss values might be different.

Since the metric derived in this way does not include information about the general trend of the loss function values throughout epochs, the idea of upgrading the metric arose, which works along the lines of the classification metric of the average area under the ROC curve. In our case, the area under the curve calculation method is used to calculate the area under the normalized loss function NL. Formally, the area under the curve of the normalized loss function NL can be defined as

$$AUNL = \int_a^b NL(x)\,dx = \frac{1}{n-1}\sum_{i=1}^{n-1}\frac{y_i + y_{i+1}}{2}\cdot h \qquad (4)$$

where a and b represent the first and the last epoch, y_i the value of NL in the i-th epoch, and h the step expressed as $x_{i+1} - x_i$. In our case, h is equal to 1, since the definition range of the function NL is equal to DNL = [1, n], and the points inside DNL are considered to be equidistant, i.e. $h = x_{i+1} - x_i = 1$ is valid for each i = 1, 2, ..., $n - 1$. A graphical representation of the area under the NL curve is presented in Fig. 2.

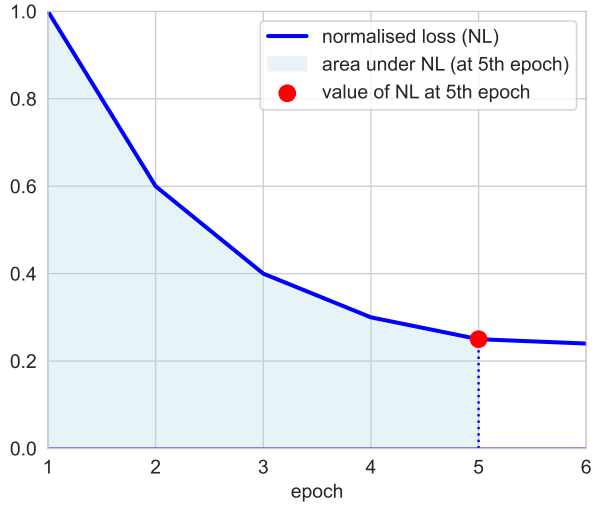

Fig. 2: Graphical representation of the AUNL metric (area under the normalized loss curve), calculated at the 5-th epoch.

B. *Training effort reduction*

Once the AUNL metric is defined, we shall define the procedure of how to (possibly) reduce the effort of training NN models. When training a complex NN architecture, several runs of training are performed, each providing a new model. In the context of HPO, (slightly) different setting of hyper-parameters is used for training, resulting in models of varying performance; even when the same setting is used for training, a non-deterministic nature of the training algorithm allows for different models.

The basic idea of the proposed training effort reduction method is to measure and monitor AUNL while training new models in successive runs. If the currently trained model has higher value of $AUNL$ at the same epoch as the best trained model so far, we can reasonably infer that further training will not result in a better performing model and can thus be terminated before the maximum number of epochs is reached. In order to give the training process a possibility to build up a model, some patience (a number of consecutive epochs before

$AUNL$ values are compared) is being used. In such a manner, the total number of epochs used for training are being reduced by premature termination, thus reducing the overall training effort. The downside of the proposed approach, however, is that a possible successful model can be lost by prematurely stopping the training.

C. HPO with the proposed training effort reduction method

Our proposed training effort reduction method for training the optimal RNN model is presented in Alg. 1. The method is based on the RS optimization approach, which provides different hyper-parameter settings and helps us find the optimal predictive model – in our case to find the appropriate number of neurons (LSTM units), dropout rate and batch size to be used in training the RNN models.

Algorithm 1 HPO approach with the TER method.

MP = Patience
NTR = Number of training runs
$AUNL_{current}$ = AUNL for currently trained model
$AUNL_{min}^{(j)}$ = Minimal achieved AUNL at j-th epoch
$X_{current}$ = Current model
X_{best} = Best model
$MAE_{current}$ = MAE of the current model on train data
MAE_{best} = MAE of the best model on train data
Initialize Random Search algorithm
while $i < NTR$ **do**
  Obtain new parameter settings S from Random Search
  Initialize $X_{current}$ based on S
  while $j <$ Maximum epochs **do**
    Train $X_{current}$ for one epoch
    Calculate $AUNL_{current}$
    if $j\%MP == 0$ **then**
      Calculate $AUNL_{current}$
      if $AUNL_{current} > AUNL_{best}^{(j)}$ **then**
        Stop training
      end if
    end if
    $j = j + 1$
  end while
  Calculate MAE on train data for $X_{current}$
  if $MAE_{current} < MAE_{min}$ **then**
    $MAE_{min} = MAE_{current}$
    $X_{best} = X_{current}$
    $AUNL_{best}^{(\cdot)} = AUNL_{current}^{(\cdot)}$
  end if
  $i = i + 1$
end while

As can be seen from Alg. 1, the used RS approach to optimize the RNN training is repeated for a definite number of training runs (NTR). In each run, the new solution vector is obtained from the RS, based on which the current RNN model is initialized and configured. After the initialization, the training process begins in which after each number of completed epochs (i.e. patience, MP) the $AUNL$ metric is calculated and compared with the $AUNL$ value of the best performing RNN model (according to MAE over train data) being trained so far. If the $AUNL$ value of the currently trained model is higher than the $AUNL$ of the best model, the training run is terminated. After all training runs are completed, the best performing model according to MAE over train data is chosen.

TABLE I: The obtained results for the full training in comparison with the proposed training effort reduction method.

metric	full training	using the proposed method
MAE	**6.385**	6.483
RMSE	**7.648**	7.784
epochs	7,200	**3,825**
used resources	100.00%	**53.125%**

V. Experiments and results

A. Experimental setup

The experiments were conducted on a dataset of 15 household electricity consumption time series. The first 720 values (consumption for 720 days) of each time series were used for training, while the remaining values (from 360 to 1.107 days, depending on the time series) were used for testing (see Fig. 1). The window size was set to 30 days. The LSTM models were trained for (up to) 30 epochs.

B. Results

With the experiment, we wanted to assess two key indicators – the final predictive performance of the trained model (using full training or the proposed training effort reduction method), measured with MAE and RMSE metrics, and the amount of training resources, measured with the overall number of epochs spent for training. For this purpose, we performed the full RNN training over all 15 time series from the prepared data set of electricity consumption. We performed HPO over each individual time series using RS, varying three hyper-parameters: the number of neurons (LSTM units) in each RNN layer, dropout rate and batch size. In each run, we trained the RNN for 30 epochs. As a final result, we used the model that achieved the highest MAE value over the training data (for each time series). For comparison, we then used the proposed training effort reduction method on the same training runs, which means that we stopped the training of an individual run prematurely (before the execution of all 30 epochs) in the case when the $AUNL$ value was worse (i.e. higher) than the $AUNL$ value in the best trained model so far (see an example in Fig. 3). Also in this case we used as the final result the RNN model that achieved the highest MAE value over the training data (for each time series).

The obtained results are summarized in Table I. We can see that in the case of full training the MAE was 0.098 kWh (or 1.5%) better than when the proposed training effort reduction method was used (1.7% better in the case of RMSE). However, using the proposed training effort reduction method the amount of needed resources was nearly halved (only 53.125% of resources were spent).

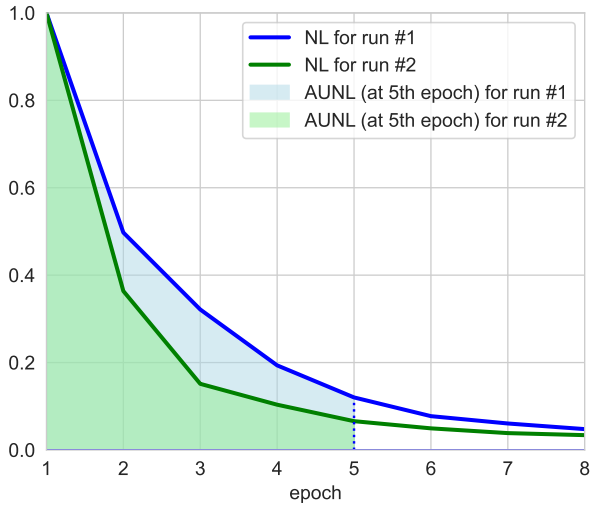

Fig. 3: Two models being trained in two runs. As the model in run #2 (green) achieved better AUNL value (at 5th epoch), the training continues; otherwise it would be terminated.

VI. Conclusion

With increasing demands for the use of complex ML models, the consumption of training resources also increases tremendously. In order to reduce the carbon footprint of AI, various attempts to adapt the NN training appear in the literature. Based on experience with optimization of deep NNs, we proposed in the paper a new training effort reduction method based on the calculation of area under the normalized loss curve. According to the results of the conducted experiment, the proposed method managed to reduce the consumption of computational resources by almost half, while the predictive performance of the trained models deteriorated only minimally. When used in a real-world setting, this could save us a lot of time and energy while bringing us significant financial benefits.

Although the presented training effort reduction method is a simple and quite naive approach to (deep) NN training optimization, it nevertheless addresses two key factors – optimization of predictive performance while simultaneously reducing the consumption of computational resources. With the increasing awareness of the importance of sustainable use of energy, we want to upgrade the presented approach in the future to further reduce the computational complexity of training complex ML models and nature-inspired algorithms to ensure the standards of Green AI.

Acknowledgment

The authors acknowledge the financial support from the Slovenian Research Agency (Research Core Funding No. P2-0057).

References

[1] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[2] Pedro Freire, Sasipim Srivallapanondh, Bernhard Spinnler, Antonio Napoli, Nelson Costa, Jaroslaw E. Prilepsky, and Sergei K. Turitsyn. Computational complexity optimization of neural network-based equalizers in digital signal processing: A comprehensive approach. *Journal of Lightwave Technology*, pages 1–25, 2024.

[3] Roberto Verdecchia, June Sallou, and Luís Cruz. A systematic review of green ai. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 13(4):e1507, 2023.

[4] Alex de Vries. The growing energy footprint of artificial intelligence. *Joule*, 7(10):2191–2194, 2023.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[6] Grega Vrbančič and Vili Podgorelec. Efficient ensemble for image-based identification of pneumonia utilizing deep cnn and sgd with warm restarts. *Expert Systems with Applications*, 187:115834, 2022.

[7] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *Advances in neural information processing systems*, 27, 2014.

[8] Alireza M Javid, Arun Venkitaraman, Mikael Skoglund, and Saikat Chatterjee. High-dimensional neural feature design for layer-wise reduction of training cost. *EURASIP Journal on Advances in Signal Processing*, 2020:1–19, 2020.

[9] Mee Seong Im and Venkat R Dasari. Computational complexity reduction of deep neural networks. *arXiv preprint arXiv:2207.14620*, 2022.

[10] IM Kulikovsvkikh. Reducing computational costs in deep learning on almost linearly separable training data. *Computer Optics*, 44(2):282–289, 2020.

[11] Dominic Sanderson and Tatiana Kalganova. Maintaining performance with less data: Understanding useful data. In *International Congress on Information and Communication Technology*, pages 1105–1127. Springer, 2023.

[12] Sajid Nazir, Shushma Patel, and Dilip Patel. Assessing hyper parameter optimization and speedup for convolutional neural networks. *International Journal of Artificial Intelligence and Machine Learning (IJAIML)*, 10(2):1–17, 2020.

[13] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[14] Vili Podgorelec, Špela Pečnik, and Grega Vrbančič. Classification of similar sports images using convolutional neural network with hyper-parameter optimization. *Applied Sciences*, 10(23):8494, 2020.

[15] Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020.

[16] Gonzalo I Diaz, Achille Fokoue-Nkoutche, Giacomo Nannicini, and Horst Samulowitz. An effective algorithm for hyperparameter optimization of neural networks. *IBM Journal of Research and Development*, 61(4/5):9–1, 2017.

[17] Claudio Gambella, Bissan Ghaddar, and Joe Naoum-Sawaya. Optimization models for machine learning: a survey. *arXiv preprint arXiv:1901.05331*, 2019.

[18] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

[19] Sigrún Andradóttir. A review of random search methods. In *Handbook of Simulation Optimization*, pages 277–292. Springer, 2015.

[20] Zelda B Zabinsky. Random search algorithms. *Wiley Encyclopedia of Operations Research and Management Science*, 2010.

[21] Martin Dyer, Alan Frieze, and Ravi Kannan. A random polynomial-time algorithm for approximating the volume of convex bodies. *Journal of the ACM (JACM)*, 38(1):1–17, 1991.

[22] Stephen A Vavasis. Complexity issues in global optimization: a survey. In *Handbook of global optimization*, pages 27–41. Springer, 1995.

[23] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[24] Yoshua Bengio, Paolo Frasconi, and Patrice Simard. The problem of learning long-term dependencies in recurrent networks. In *IEEE international conference on neural networks*, pages 1183–1188. IEEE, 1993.