Improving Automated Hyperparameter Optimization with Case-Based Reasoning*

Maximilian Hoffmann[1,2] and Ralph Bergmann[1,2]

[1] Artificial Intelligence and Intelligent Information Systems, University of Trier, 54296 Trier, Germany, `http://www.wi2.uni-trier.de`
[2] German Research Center for Artificial Intelligence (DFKI) Branch University of Trier, Behringstraße 21, 54296 Trier, Germany
`{hoffmannm,bergmann}@uni-trier.de;`
`{maximilian.hoffmann,ralph.bergmann}@dfki.de`

Abstract. The hyperparameter configuration of machine learning models has a great influence on their performance. These hyperparameters are often set either manually w. r. t. to the experience of an expert or by an *Automated Hyperparameter Optimization (HPO)* method. However, integrating experience knowledge into HPO methods is challenging. Therefore, we propose the approach HypOCBR (**Hyp**erparameter **O**ptimization with **C**ase-**B**ased **R**easoning) that uses *Case-Based Reasoning (CBR)* to improve the optimization of hyperparameters. HypOCBR is used as an addition to HPO methods and builds up a case base of sampled hyperparameter vectors with their loss values. The case base is then used to retrieve hyperparameter vectors given a query vector and to make decisions whether to proceed trialing with this query or abort and sample another vector. The experimental evaluation investigates the suitability of HypOCBR for two deep learning setups of varying complexity. It shows its potential to improve the optimization results, especially in complex scenarios with limited optimization time.

Keywords: Case-Based Reasoning · Automated Hyperparameter Optimization · Machine Learning · Deep Learning

1 Introduction

In recent years, machine learning and especially *Deep Learning (DL)* models have been used as a method of choice for solving various tasks, e. g., in decision support systems [18] and in helpdesk scenarios [2]. An important part of these models are the *hyperparameters* that are used to configure the model architecture or the learning process such as the number of layers or the learning rate. Hyperparameters are numerous in modern DL setups and there is an ongoing trend towards even more complex models, making it increasingly difficult to find

*The final authenticated publication is available online at `https://doi.org/10.1007/978-3-031-14923-8_18` suitable configurations for the large number of hyperparameters. Despite the existence of *Automated Hyperparameter Optimization (HPO)* methods [7,9,16], it is still common practice to tune these hyperparameters manually based on user experience. This task is, on the one hand, challenging since it requires in-depth knowledge of the underlying task and DL setup to set hyperparameter values appropriately. On the other hand, it is time-consuming as the process involves manually triggered iterations of selecting a hyperparameter configuration, training the parameterized model, and validating its performance. Whereas the latter aspect can be automated even by the simplest HPO methods, expert knowledge of the DL setup to be tuned can be hardly considered in current HPO methods (e. g., [8,12,15]). For instance, there might be knowledge in the form of a statement such as this: *The second convolutional layer has a great influence on the overall results.* To use this knowledge in current HPO approaches, it would be necessary to transform it into meta-parameters of the HPO method, such as the number of sampled hyperparameter vectors. However, this transformation is not trivial and requires in-depth knowledge of the HPO method, which the neural network modeler might be lacking.

To address these shortcomings, we propose an approach called HypOCBR (**Hyp**erparameter **O**ptimization with **C**ase-**B**ased **R**easoning) for combining HPO methods with a *Case-Based Reasoning (CBR)* [1] system that allows an explicit integration of expert knowledge into optimization procedures. The main assumption is that similar hyperparameter vectors lead to similar optimization results. Based on this assumption and modeled domain and similarity knowledge that expresses an expert's experience knowledge, HypOCBR uses a CBR system to filter sampled hyperparameter vectors from an HPO method and make decisions to proceed training with them or abort them. The proposed approach is designed to be used as an extension of existing HPO methods without required modifications to their core functionality. The remainder of the paper is structured as follows: Section 2 describes the fundamentals of HPO, including the hyperparameters of an example DL model and random search as a popular HPO method. Additionally, this section discusses related work about HPO in CBR research. Section 3 describes our approach by introducing the proposed system architecture and the used retrieval and decision-making process. The presented approach is evaluated in Sect. 4 where optimization procedures with and without CBR methods are compared on two DL setups of varying complexity. Finally, Sect. 5 concludes the paper and examines future research directions.

2 Automated Hyperparameter Optimization of Deep Learning Models

Our approach aims at optimizing *Deep Learning (DL)* models with the help of *Case-Based Reasoning (CBR)* methods that are built on top of existing *Automated Hyperparameter Optimization (HPO)* methods. In this section, we introduce the formal concepts and terminology of HPO (see Sect. 2.1). Additionally, different types of hyperparameters (see Sect. 2.2) are examined and random search, a popular HPO method (see Sect. 2.3), is presented. We further discuss related work in Sect. 2.4.

2.1 Formal Definition

Hyperparameter optimization involves two basic components in its simplest form, i. e., a training setup of a DL model including its hyperparameters and training data, and a search procedure for selecting a hyperparameter vector from the hyperparameter search space. Our introduction and notation of these components follows Feurer and Hutter [9]: A DL model $\mathcal{A}$ is parameterized by a set of hyperparameters N, where each hyperparameter $n \in N$ is defined by its domain Λ_n. The overall hyperparameter search space is given by $\Lambda = \Lambda_1 \times \Lambda_2 \times \Lambda_n$. We denote $\lambda \in \Lambda$ as a hyperparameter vector that represents a distinct sample from the hyperparameter search space. Instantiating the model $\mathcal{A}$ with this sample is denoted as $\mathcal{A}_\lambda$. The optimization procedure is defined as follows:

$$\lambda^* = \operatorname*{argmin}_{\lambda \in \Lambda} \mathcal{L}(\mathcal{A}_\lambda, \mathcal{D}) \tag{1}$$

Thereby, the optimal hyperparameter vector $\lambda^* \in \Lambda$ is determined by selecting a hyperparameter vector $\lambda \in \Lambda$ that minimizes the loss function $\mathcal{L}(\mathcal{A}_\lambda, \mathcal{D})$. The loss value is the result of training $\mathcal{A}_\lambda$ and validating its performance on the dataset $\mathcal{D}$. Therefore, $\mathcal{D}$ is usually split up into a training dataset for training $\mathcal{A}_\lambda$ and a validation dataset for computing the loss value. We refer to the training, validation, and loss computation with a single hyperparameter vector by the term *trial*. Please note that instead of loss values, which are always minimized, also other metrics, e. g., classification accuracy, can be used for optimization according to Eq. 1. The only necessary change is to compute argmax$(\cdot)$ instead of argmin$(\cdot)$ in some cases. When referring to loss values in this paper, we mean both types, i. e., metrics to minimize and to maximize.

2.2 Hyperparameter Search Spaces

The computational effort of an optimization as shown in Eq. 1 is mainly determined by the number of possible hyperparameter vectors $|\Lambda|$ and the time needed to perform trials with the parameterized models. But, even if the training time is small and it is a simple DL setup, Λ can become very large and the whole optimization process very slow. Consider the following example[3]: A DL setup for image classification is made up of two consecutive layers of convolution followed by pooling. The convoluted images are then flattened, i. e., reshaped to a one-dimensional vector, and fed into several fully-connected layers. The final output is a vector of ten elements that represents the probabilities for the individual classes of a classification task. The model is trained by processing batches of images and applying stochastic gradient descent according to the computed loss. A possible hyperparameter search space for this example setup is illus-

[3]The example is derived from an introduction on convolutional neural networks, accessible at `https://www.tensorflow.org/tutorials/images/cnn`.

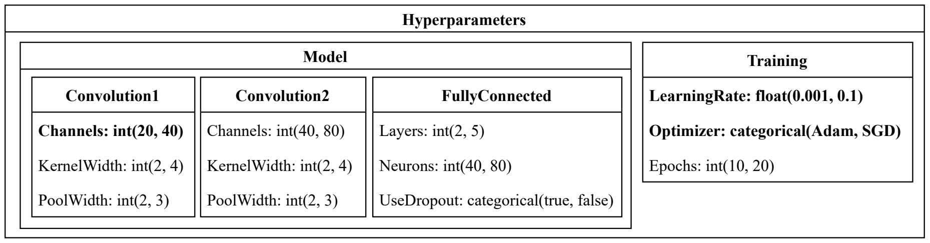

Fig. 1. Example Hyperparameter Search Space

trated in Fig. 1. The hyperparameters are structured hierarchically, with model and training hyperparameters forming the main groups. Each hyperparameter belongs to a certain type of search space (Λ_n) and is constrained to certain values. For instance, the number of channels in the first convolution is an integer value between 20 and 40 and the learning rate is a float value between 0.001 and 0.1. These types of search spaces influence the number of possible values of each hyperparameter. For instance, categorical and integer hyperparameters such as the optimizer or the number of channels represent a definitive set of possible values. Continuous search space types such as the learning rate, in turn, have an infinite number of theoretically possible values, assuming an infinitely small step size. This makes it infeasible to trial with every possible hyperparameter vector to compute argmin($\cdot$) and to find λ^* [6].

2.3 Random Search

Therefore, most search algorithms only examine a small subset of Λ that does not guarantee finding the optimal hyperparameter vector [9]. Random search (see Alg. 1 and cf. [6] for more information) is a simple example of these algorithms. The algorithm is an approximation of the function argmin($\cdot$) from Eq. 1 and works in the following way: Random hyperparameter vectors λ are selected from Λ by the function selectFrom($\cdot$) and trialed. If the loss of the selected sample is smaller than the loss of the current best hyperparameter vector λ^+, λ will be assigned to λ^+. This loop repeats until the search is finished (func-

Algorithm 1 Random Search

```
1: $\lambda \leftarrow \text{selectFrom}(\Lambda)$
2: $\lambda^+ \leftarrow \lambda$
3: while not isOptimized($\lambda^+, \mathcal{L}$) do
4:     $\lambda \leftarrow \text{selectFrom}(\Lambda)$
5:     if $\mathcal{L}(\mathcal{A}_\lambda, \mathcal{D}) < \mathcal{L}(\mathcal{A}^+_\lambda, \mathcal{D})$ then
6:         $\lambda^+ \leftarrow \lambda$
7:     end if
8: end while
9: return $\lambda^+$
```

tion isOptimized$(\cdot, \cdot)$) and the best selected sample λ^+ is returned. The used termination criterion is dependent on the underlying scenario and can be, for instance, the maximum search time, the maximum number of selected samples, or a threshold of the loss value. If random search has an unlimited budget, it will converge towards λ^+ being equal to the optimal hyperparameter vector λ^*.

2.4 Related Work

We discuss related work that covers popular HPO methods and CBR approaches in the field of HPO. Literature from the former category is discussed in detail in several survey publications (e. g., [9,16,26]), which is why we only want to highlight two popular example approaches: Multi-fidelity optimization aims at approximating the actual loss value of a setup by conducting low-fidelity and high-fidelity optimizations with a specific budget allocation, e. g., a maximum number of training iterations. Hyperband [15] uses an iterative process where the budget for each trial is determined based on the results of this run in the last iteration. Well-performing runs are allocated more resources and poorly-performing runs are terminated, until only the best-performing optimization is left. Model-based black box optimization methods aim to improve on other methods, e. g., Hyperband, that usually sample random hyperparameter vectors. Instead, promising hyperparameter vectors are chosen based on a surrogate model that is built during the optimization to copy the hyperparameter distribution of the black box method to optimize, e. g., a DL model. Bayesian optimization [24] methods use a surrogate model that is based on probabilistic methods, e. g., Gaussian processes. Our approach is compatible with both categories of HPO methods and, additionally, provides the opportunity of expert knowledge integration into the optimization process. Expert knowledge is directly integrated as CBR domain and similarity knowledge instead of being indirectly modeled in existing HPO methods.

The joint applications of CBR and HPO methods in literature are mostly concerned with hyperparameter tuning of CBR algorithms, e. g., feature weighting [25]. The opposite relation of CBR used for HPO is, to the best of our knowledge, not commonly discussed. We highlight some selected approaches: Pavón et al. [20] use CBR to enable automatic selection of Bayesian model configurations for individual problem instances. Their application builds up a case base of configurations, which is used to find the best-matching configuration for an upcoming problem instance. This idea is further pursued by Yeguas et al. [27] and applied to the task of tuning parameters of evolutionary optimization algorithms. Auslander et al. [3] explore a case-based approach of setting parameter values in the scenario of multi-agent planning. They retrieve parameter settings from a case base to parameterize upcoming planning problems. Molina et al. [19] use a set of past decision tree executions to predict suitable parameters for the application of the decision tree on new datasets. The parameter settings are retrieved based on the characteristics of the datasets. In contrast to the approach in this paper, most of the presented approaches can be classified as AutoML methods [13] that are used as a replacement for hyperparameter optimization. These methods automatically select a model configuration according to the training data or the learning task instead of tuning parameters from scratch. Our approach is novel in the sense that it aims to integrate CBR within the optimization procedure to make expert knowledge about the hyperparameter search space available and guide the optimization by eliminating unpromising hyperparameter vectors.

3 Automated Hyperparameter Optimization with Case-Based Reasoning

This section introduces our approach HypOCBR (**Hyp**erparameter **O**ptimization with **C**ase-**B**ased **R**easoning) that aims at improving *Automated Hyperparameter Optimization (HPO)* of *Deep Learning (DL)* hyperparameters with *Case-Based Reasoning (CBR)* methods. Our goal is to accelerate the optimization process by utilizing HypOCBR as a filter for the hyperparameter vectors that classifies whether a vector should be considered for trialing or not. Thereby, the strategy is to disregard hyperparameter vectors in situations where they are highly similar to known low-performing examples that are collected in a case base throughout the optimization procedure. Figure 2 shows an architectural overview of the three

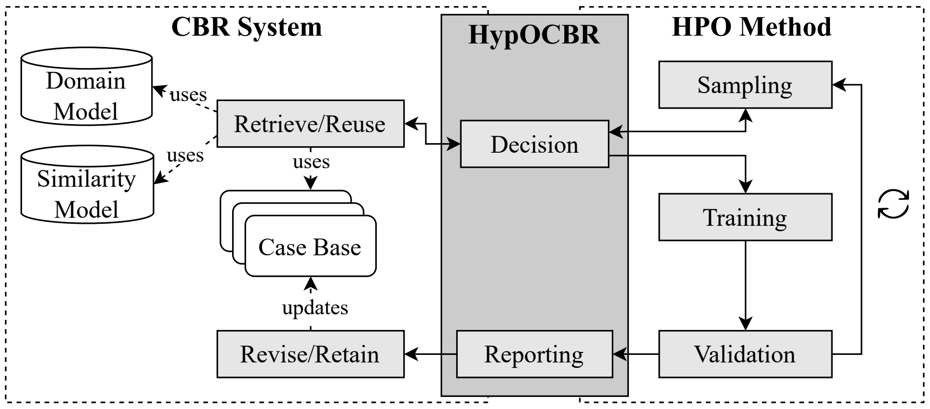

Fig. 2. Architecture of the CBR system, the HPO method, and HypOCBR

involved components, i. e., the CBR system, the HPO method, and HypOCBR. The HPO method carries out the optimization procedure by sampling hyperparameter vectors from the search space, performing training, and validating the model's performance (see Sect. 2.3). The architecture, in this regard, allows using arbitrary HPO methods, since their functionality does not have to be fitted to HypOCBR. The CBR system supports the optimization procedure by maintaining a case base with completed trials of hyperparameter vectors and their corresponding loss values as well as domain and similarity models to include domain expert knowledge. HypOCBR connects the CBR system and the HPO method by making use of a lightweight implementation of the 4R cycle [1]: After a new hyperparameter vector is sampled during the optimization, HypOCBR makes a decision to proceed with this vector or abort it early and sample another vector. Each sampled vector serves as a query to the CBR system based on which the case base is searched for similar hyperparameter vectors and their corresponding loss values, i. e., retrieving trials. HypOCBR then reuses the most similar trials and makes the decision based on their loss values. Please note that the retrieved hyperparameters are not adapted but their loss values are used to assess the potential of the queried hyperparameter vector. Each optimization gradually builds up a case base to use by starting with an empty case base and retaining each completed trial therein. Thereby, we do not consider an explicit revision of cases in the approach but rather store all reported validation results from the HPO component. Please note that this can lead to large case bases in long optimization runs, such that methods for speeding up retrieval (e. g., [17]) or decreasing the size of the case base (e. g., [22]) can be necessary but are not further discussed in this work. In the following, we will examine how a retrieval in the CBR system is performed by inspecting the case representation as well as the similarity definitions (see Sect. 3.1). The retrieval is part of the decision-making process of HypOCBR that is explained in Sect. 3.2.

3.1 Experience-Based Retrieval: Case Representation and Similarity Definition

The case representation models trials as pairs of their hyperparameter vector and the respective loss value. The case base, in this regard, stores information on the quality of hyperparameter vectors that were sampled during the optimization. The hyperparameter vectors are represented as object-oriented cases, as shown by the example in Fig. 1. The domain model describing these vectors is one point to integrate expert knowledge. The concrete form is highly dependent on the structure of the modeled hyperparameter vectors. For instance, simple knowledge about the data types, value ranges, and expected distributions of individual hyperparameters (see Sect. 2.2) can build the baseline. It can be extended by knowledge about the structure of individual hyperparameters (e. g., taxonomies of optimizer types) or the constraints among hyperparameters (e. g., the width of the kernel has to be less than the image width and height). Even more complex knowledge such as ontologies can be used to model the cases. The loss value that is also part of each case, usually is a single numeric value, e. g., accuracy, but can also be represented as a more complex object, e. g., a list of accuracy values from each epoch of training.

During retrieval, similarities are computed between the hyperparameter vectors of the cases and the hyperparameter vector that is given as the query by HypOCBR. We do not specify the similarity measures to use, but rather allow them to be customized regarding the use case and the available expert knowledge. Thereby a global object-oriented similarity between two hyperparameter vectors is usually put together from multiple local similarities between the individual hyperparameters, e. g., numeric measures for integer and float search spaces, binary measures for boolean search spaces, and tables or taxonomies for categorical values (cf. [4, pp. 93f.] for more information). This enables a precise customization of the global pairwise similarity based on the expert's experience: For instance, the weights of all hyperparameters from the training category (see Fig. 1) can be increased to focus more on this part of the hyperparameter vectors for similarity assessment.

3.2 Making a Decision for New Trials

The decision task of HypOCBR applies an algorithm to make a decision between *proceeding* with a sampled hyperparameter vector or *aborting* the trial. The used approach is conceptualized in Alg. 2 and can be parameterized according to the well-known principles of *exploration* and *exploitation* (see, for instance, Leake and Schack [14] for more information). This means that it can be configured towards favoring hyperparameter vectors that are different from all previously encountered vectors (exploration) or similar to known good hyperparameter vectors (exploitation). The meta-parameters to influence the behavior of the algorithm are given in Tab. 1.

The algorithm's first step is to check if the case base has reached a certain size (lines 1 – 3) which is given as the meta-parameter θ_1. This check is necessary as a case base is built up throughout the optimization to improve the quality of upcoming decisions. The case base, however, suffers from the cold start problem

Algorithm 2 Decision-Making for New Trials

```
Input: case base CB, similarity function $\mathrm{sim}(\cdot,\cdot)$, query $\lambda_q$
Output: PROCEED or ABORT
Meta-Parameters: $\theta_1$, $\theta_2$, $\theta_3$, $\theta_4$

1:  if $|\,\mathrm{CB}\,| < \theta_1$ then
2:      return PROCEED                    ▷ Minimum size of case base not reached
3:  end if
4:  $r \leftarrow \mathrm{retrieve}(\mathrm{CB}, \mathrm{sim}, \lambda_q, \theta_4)$
5:  $s \leftarrow \mathrm{aggregateSimilarities}(r)$
6:  if $s \geq \theta_2$ then
7:      $l \leftarrow \mathrm{aggregateLosses}(r)$
8:      $p \leftarrow \mathrm{percentile}(l, \mathrm{CB}, \theta_3)$
9:      if $p = true$ then
10:         return PROCEED                ▷ Exploitation
11:     else
12:         return ABORT                  ▷ Similar to underperforming cases
13:     end if
14: else
15:     return PROCEED                    ▷ Exploration
16: end if
```

[23] which can cause undesired behavior in CBR systems with little or no data at the launch of the application (e. g., [21]). If the case base has reached the desired size, it can be used for retrieving r (line 4) with the query λ_q, the case base CB, and the similarity measure for case pairs sim. The retrieval incorporates the meta-parameter θ_4 that defines the number of nearest neighbors to retrieve. Larger values of θ_4 help to prevent overfitting and make the retrieved cases more robust to outliers. These cases are then used to compute an aggregate similarity value s that measures how similar λ_q is to the retrieved cases from the case base r (line 5). The aggregated similarity is computed to ensure that the retrieved cases are sufficiently similar to make a decision. The retrieval is also the main part of the algorithm where expert knowledge is used. It determines the computed similarities, which direct the algorithm to the hyperparameter vectors and their loss values that, in turn, best match the queried vector. As the retrieved cases are the basis of the decision, the knowledge directly influences the measured potential of the query vector. If s does not exceed θ_2 (line 6) then the exploration strategy is followed and `PROCEED` is returned (see line 15). Otherwise, the retrieved cases are further inspected in lines 7 – 13. The loss values of the retrieved cases are aggregated to l (line 7) to get an average loss of the cases that are similar to the query. Given l, the case base CB, and the meta-parameter θ_3 as parameters, the function percentile$(\cdot, \cdot, \cdot)$ in line 8 returns a boolean value that expresses if the average loss value l is in the percentile range given by θ_3. For instance, setting θ_3 to a value of 55 expresses that p is true if l is among the best 45 % of the loss values in the case base. This gives a hint towards the potential quality of λ_q which is based on the average quality of cases similar to λ_q (retrieved cases r) w. r. t. to all cases in the case base CB. Eventually, `PROCEED` is returned if p is true (exploitation strategy; line 9) and `ABORT` is returned if p is false. The algorithm only aborts in one particular scenario, which highlights the main goal of the filter process, i. e., to abort trials if they are expected to lead to bad results.

Table 1. Meta-Parameters of HypOCBR

Meta-Parameter	**Description**	**Constraints**
θ_1	Minimum number of cases in the case base to avoid cold start	$\theta_1 > 0$
θ_2	Minimum similarity threshold of aggregated retrieved cases	$\theta_2 \in [0, 1]$
θ_3	Minimum percentile of aggregated metric of retrieved cases	$\theta_3 \in [0, 100]$
θ_4	Number of cases to retrieve	$\theta_4 <= \theta_1$

Furthermore, the meta-parameters (see Tab. 1) provide a straightforward way of configuring the behavior of the algorithm w. r. t. certain goals. For instance, it can be configured towards exploration by increasing θ_2 which requires a higher similarity of retrieved cases. A parameterization towards an `ABORT` decision is also possible by decreasing θ_2 and increasing θ_3. Additionally, pretests of an implementation of Alg. 2 with standard similarity measures of the process-oriented CBR framework ProCAKE [5] have shown that the performance is adequate, enabling working with case bases containing hundreds or thousands of trials within a few milliseconds.

4 Experimental Evaluation

The experimental evaluation compares the results and procedure of hyperparameter optimization with and without the usage of the proposed *Case-based Reasoning (CBR)* component HypOCBR. To include optimization tasks with different levels of complexity, we evaluate a rather simple baseline setup of an image classification task and a much more complex similarity learning task based on *Graph Neural Networks (GNNs)*. The goal is to analyze the influence of HypOCBR on the number of sampled hyperparameter vectors and the loss values of the trialed cases. The following hypotheses are examined:

H1 The computation overhead introduced by HypOCBR is negligible.
H2 The usage of HypOCBR in an optimization procedure leads to better results compared to an identical optimization without it.
H3 The benefit of integrating HypOCBR in an optimization procedure increases with the complexity, i. e., longer running trials with less overall optimization budget, of the setup to optimize.

Hypothesis H1 addresses the computation overhead introduced by the CBR system, which is expected to not influence the completed number of trials compared to HPO without HypOCBR. Hypothesis H2 examines the potential of the presented approach to conduct better trials w. r. t. their loss values. Hypothesis H3 aims at investigating the influence of the setup complexity on the benefit of HypOCBR.

4.1 Experimental Setup

The experiments feature two *Deep Learning (DL)* setups. The baseline setup (S - I) is a convolutional neural network classifying CIFAR10 images (derived from the example in Sect. 2.2). The model is fully trained for ten epochs within one optimization run with the goal of maximizing the percentage of correctly classified images (*accuracy*). The hyperparameter vectors of the model contain 11 parameters with only integer search spaces. The number of possible hyperparameter combinations, i. e., the number of unique hyperparameter vectors, is approx. $3 \cdot 10^7$. Additionally, we also want to look at a more complex problem by incorporating a second setup (S - II) that uses GNNs. S - II is based on our previous work on using graph embeddings for similarity learning [10,11]. Due to the required time for a full training run being approx. 12 – 18 hours, the iterations in each epoch are limited and the model is only trained for ten epochs in trials.

Each hyperparameter vector contains 26 hyperparameters, i. e., 18 integer hyperparameters, five float hyperparameters, and 3 categorical hyperparameters. This results in a total number of approx. $2 \cdot 10^{37}$ possible hyperparameter vectors. The target metric is the *Mean Absolute Error (MAE)* between the predicted similarities and the label similarities that should be minimized (see more details in [10,11]). Both setups are validated on a part of the dataset that is disjoint from the data that is used for training. We use simple domain models for both setups that include the data types and value ranges of all hyperparameters as well as an object-oriented structure of the vectors. The similarity model is also simple with linear numeric measures for integers and floats and binary measures for categorical hyperparameters. The similarity between two hyperparameter vectors is computed by a weight-adjusted average of the object-oriented structure of the hyperparameters.

The HPO method used in the experiments is random search (see Sect. 2.3) as it is a baseline method that is often used and performs reasonably well among other state-of-the-art optimization methods [9]. This way, we can focus on the effects of HypOCBR without the need for factoring in differently parameterized HPO methods which might be required, e. g., if using Bayesian methods. Each optimization procedure is conducted twice, with the only difference being that HypOCBR is just used in one procedure. All other factors are kept identical, which includes the dataset, the training procedure, and the HPO procedure. Randomized parts, e. g., the initial model weights and biases or the sampled hyperparameter vectors, are made deterministic by using random seed values. This means that it is possible to perform an optimization run with the same results if the same random seed is used. The used meta-parameters of HypOCBR are set according to results of pretests. For S - I, that is $\theta_1 = 20$, $\theta_2 = 0.81$, $\theta_3 = 50$, $\theta_4 = 15$ and for S - II that is $\theta_1 = 15$, $\theta_2 = 0.7$, $\theta_3 = 45$, $\theta_4 = 10$.

The termination criterion for each optimization with and without HypOCBR is the maximum elapsed time. A value of one hour is set for all optimizations of S - I and four hours for all optimizations of S - II. These values reflect the different complexities of both setups, where training and validation takes longer for S - II. After the optimization process is terminated, the trained and validated hyperparameter vectors are analyzed. We assume a scenario where an expert reviews the best hyperparameter vectors to make the final decision. Therefore, the validation results, i. e., the accuracy or MAE, of all trials and the ten best trials are compared among optimization procedures with and without HypOCBR. The implementation is realized as an extension of the open-source process-oriented CBR framework ProCAKE[4] [5]. All experiments are conducted on a computer with an Intel Xeon Gold 6138 CPU and an NVIDIA Tesla V100 SXM2, running Ubuntu 18.04 LTS.

[4] `http://procake.uni-trier.de`

Table 2. Evaluation Results for S - I

Trials		Loss (All)			Loss (Top)	
#	# (Proceed, Abort)	Best	Mean	Median	Mean	Median
161	620 (155, 465)	0.00%	0.50%	1.10%	0.70%	0.81%
152	609 (149, 460)	-0.44%	1.85%	1.60%	0.87%	0.99%
141	646 (143, 503)	0.00%	1.64%	1.50%	0.49%	0.66%
149	728 (141, 587)	1.37%	1.18%	1.29%	0.74%	0.55%
153	640 (149, 491)	2.33%	1.02%	0.72%	0.76%	0.57%
158	613 (158, 455)	-0.10%	0.90%	0.49%	0.29%	0.37%
161	456 (157, 299)	0.00%	1.49%	1.10%	1.00%	1.17%
164	413 (160, 253)	-0.43%	2.00%	1.94%	0.30%	0.41%
163	863 (159, 704)	-0.34%	1.96%	2.21%	0.52%	0.69%
159	690 (158, 532)	0.77%	1.13%	0.71%	0.37%	0.43%

4.2 Experimental Results

The results of the conducted experiments are depicted in Tab. 2 for S - I and in Tab. 3 for S - II. The information in both tables is structured analogously: The lines show ten optimization procedures, once conducted with and without HypOCBR under the same circumstances, i. e., with the same random seed. We give information on the number of trials that were evaluated (two leftmost columns), which is further split up into the trials with a *proceed* and an *abort* decision for the optimizations with HypOCBR. The loss values of the individual optimizations are percentage differences between optimizations with and without HypOCBR. A negative value corresponds to a negative impact of HypOCBR and vice versa. The table shows best, mean, and median values aggregated from all trials (left) and the ten best trials (right) of the individual optimizations. All presented differences are statistically significant ($p < 0.01$).

The results for S - I (see Tab. 2) show that the HypOCBR approach aborts many trials and only proceeds with a small share. The number of trials that are

Table 3. Evaluation Results for S - II

Trials		Loss (All)			Loss (Top)	
#	# (Proceed, Abort)	Best	Mean	Median	Mean	Median
105	685 (110, 575)	-0.19%	17.15%	16.21%	22.95%	22.00%
108	923 (103, 820)	2.97%	16.17%	34.29%	17.13%	27.17%
113	671 (111, 560)	0.38%	16.97%	17.47%	14.13%	11.70%
113	865 (103, 762)	0.00%	16.52%	12.31%	20.13%	17.71%
107	155 (116, 39)	0.00%	7.01%	21.39%	18.11%	19.73%
112	169 (118, 51)	2.75%	4.33%	21.23%	8.62%	14.06%
115	750 (112, 638)	22.60%	16.07%	14.66%	26.00%	49.14%
113	876 (105, 771)	0.00%	11.80%	9.76%	14.57%	4.52%
114	716 (114, 602)	-0.03%	15.58%	19.27%	-0.70%	3.48%
113	171 (110, 61)	0.00%	6.53%	22.57%	0.07%	0.00%

not aborted is in a similar range as the number of trials when HypOCBR is not used. However, the optimizations that use HypOCBR are capable of pre-filtering a larger number of trials (increase between 251 % and 529 %) which provides a better coverage of the search space. The aggregated accuracies (higher values are better) show better results of the mean and median values for HypOCBR on average, while the approach without found a better vector in four optimizations. All in all, the differences between both variants are small, ranging from 0.29 % to 2 % for the mean and median accuracies and from -0.44 % to 2.33 % for the best accuracy.

The results for S - II (see Tab. 3) show that the number of proceeded trials for optimizations with HypOCBR is approximately equal to the number of trials for optimizations without HypOCBR while pre-selecting numerous trials (increase from 144 % to 854 %). The aggregated MAE values (lower values are better) generally show improvements when using HypOCBR: The best MAE values show decreases between -0.19 % and 22.6 %. The mean and median MAE values mostly show large decreases of up to 49.14 %.

4.3 Discussion

The overall results of the optimization procedures that use HypOCBR are promising. Regarding Hypothesis H1, HypOCBR optimizations manage to train and validate approx. the same number of trials as the corresponding optimizations without HypOCBR. The small deviations of the results follow no trend and are most likely caused by computational variations of the underlying hardware. Thus, the introduced performance overhead is negligible and H1 is accepted. When analyzing the MAE and accuracy values, HypOCBR improves the optimization results in most cases with a few exceptions that stem largely from the best trials. The overall small improvements for S - I might be due to the full training run that is conducted during trialing. Thereby, the optimizer has the complete dataset to train the model and, thus, might be able to find suitable model weights even for inferior hyperparameters. The results for S - II show much clearer improvements than for S - I. With median MAEs being decreased by up to 49 %, HypOCBR significantly improves the optimization procedures. This especially reveals potential in scenarios with a limited time budget for optimization where only a small amount of trials is possible, e. g., in prototyping. There, the CBR component allows sampling more hyperparameter vectors that can be assessed for trialing. Although some of these trials might be misclassified by HypOCBR, it still improves the overall results of the optimization. The consistently improved mean and median results of all trials also hints at a success of eliminating bad trials. Therefore, we conclude that Hypothesis H2 is only partly accepted, since the integration of HypOCBR does not improve the optimization results in every case. The results are consistent with Hypothesis H3 due to the noticeably improved optimization for the more complex setup S - II compared to S - I. However, additional tests are needed to solidify the results.

5 Conclusion and Future Work

This paper introduced our approach called HypOCBR for enabling the use of *Case-Based Reasoning (CBR)* to improve *Automated Hyperparameter Optimization (HPO)*. HypOCBR acts as a filter for sampled hyperparameter vectors, deciding whether a vector should be considered for training and validation or not. The decision is based on a case base of hyperparameter vectors that were already trained and validated during optimization and on expert-modeled domain and similarity knowledge. These components are utilized in the following decision-making process: retrieve similar hyperparameter vectors given a query vector, aggregate their loss values, compare the aggregated loss value to the distribution of loss values, proceed training and validation with the query if it is similar to well-performing hyperparameter vectors. When evaluated for two *Deep Learning (DL)* setups of varying complexity, optimization procedures with HypOCBR show great potential: Due to the filtering function, more hyperparameter vectors can be analyzed, which leads to better optimization results for both setups.

A focus of future research should be on further optimizing, extending, and evaluating the approach. For instance, the approach might benefit from self-learning capabilities for the decision-making process that enable automatic tuning of the meta-parameters during optimization and easier application of the approach to new data. Further, the role of the case base can be extended to reuse existing case bases during startup and export the case base after the optimization procedure. It could also be beneficial to investigate more advanced methods for the reuse and revise phases of the described lightweight CBR cycle, which might, for instance, include methods for case adaptation or managing the growth of the case base (e. g., [22]). Additionally, HypOCBR could be evaluated on a larger scale by covering other HPO methods (e. g., [8,15]), other DL setups (see [10,11] for more examples), and the influence of differently modeled domain and similarity knowledge.

References

1. Aamodt, A., Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches. AI Communications **7**(1), 39–59 (1994)
2. Amin, K., et al.: Advanced similarity measures using word embeddings and siamese networks in cbr. In: Intell. Systems and Applications, Advances in Intell. Syst. and Comp., vol. 1038, pp. 449–462. Springer, Cham (2020)
3. Auslander, B., Apker, T., Aha, D.W.: Case-based parameter selection for plans: Coordinating autonomous vehicle teams. In: Case-based Reasoning Research and Development, LNCS, vol. 8765, pp. 32–47. Springer, Cham (2014)
4. Bergmann, R.: Experience Management: Foundations, Development Methodology, and Internet-Based Applications, LNCS, vol. 2432. Springer (2002)
5. Bergmann, R., Grumbach, L., Malburg, L., Zeyen, C.: ProCAKE: A Process-Oriented Case-Based Reasoning Framework. In: Workshop Proceedings of ICCBR. vol. 2567, pp. 156–161. CEUR-WS.org (2019)
6. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of machine learning research **13**(2) (2012)

7. Claesen, M., de Moor, B.: Hyperparameter search in machine learning. CoRR **abs/1502.02127** (2015)
8. Falkner, S., Klein, A., Hutter, F.: Bohb: Robust and efficient hyperparameter optimization at scale. ICML pp. 1437–1446 (2018)
9. Feurer, M., Hutter, F.: Hyperparameter optimization. In: Hutter, F., Kotthoff, L., Vanschoren, J. (eds.) Automated Machine Learning, pp. 3–33. Springer eBooks Computer Science, Springer, Cham (2019)
10. Hoffmann, M., Bergmann, R.: Using Graph Embedding Techniques in Process-Oriented Case-Based Reasoning. Algorithms **15**(2), 27 (2022)
11. Hoffmann, M., Malburg, L., Klein, P., Bergmann, R.: Using siamese graph neural networks for similarity-based retrieval in process-oriented case-based reasoning. In: Case-Based Reason. Res. and Dev., LNCS, vol. 12311, pp. 229–244. Springer (2020)
12. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of International Conference on Neural Networks (ICNN'95), Perth, WA, Australia, November 27 - December 1, 1995. pp. 1942–1948. IEEE (1995)
13. Leake, D., Crandall, D.: On bringing case-based reasoning methodology to deep learning. In: Watson, I., Weber, R. (eds.) Case-Based Reasoning Research and Development, LNCS, vol. 12311, pp. 343–348. Springer, Cham (2020)
14. Leake, D., Schack, B.: Exploration vs. exploitation in case-base maintenance: Leveraging competence-based deletion with ghost cases. In: Case-Based Reasoning Res. and Dev., LNCS, vol. 11156, pp. 202–218. Springer, Cham (2018)
15. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: A novel bandit-based approach to hyperparameter optimization. Journal of Machine Learning Research **18**(1) (2018)
16. Luo, G.: A review of automatic selection methods for machine learning algorithms and hyper-parameter values. Network Modeling Analysis in Health Informatics and Bioinformatics **5**(1) (2016)
17. Malburg, L., Hoffmann, M., Trumm, S., Bergmann, R.: Improving Similarity-Based Retrieval Efficiency by Using Graphic Processing Units in Case-Based Reasoning. In: Proc. of the 34th FLAIRS Conf. FloridaOJ (2021)
18. Mathisen, B.M., Bach, K., Aamodt, A.: Using extended siamese networks to provide decision support in aquaculture operations. Appl. Intell. pp. 1–12 (2021)
19. Molina, M.M., Luna, J.M., Romero, C., Ventura, S.: Meta-learning approach for automatic parameter tuning: A case study with educational datasets. In: Proc. of the 5th Int. Conf. on Edu. Data Mining. pp. 180–183. Chania, Greece (2012)
20. Pavón, R., Díaz, F., Laza, R., Luzón, V.: Automatic parameter tuning with a bayesian case-based reasoning system. a case of study. Expert Systems with Applications **36**(2), 3407–3420 (2009)
21. Quijano-Sánchez, L., Bridge, D., Díaz-Agudo, B., Recio-García, J.A.: A case-based solution to the cold-start problem in group recommenders. In: Case-based Reasoning Res. and Dev., LNCS, vol. 7466, pp. 342–356. Springer, Heidelberg (2012)
22. Roth-Berghofer, T.R.: Knowledge maintenance of case-based reasoning systems: The SIAM methodology, Dissertationen zur künstlichen Intelligenz, vol. 262. Akad. Verl.-Ges. Aka, Berlin (2003)
23. Schafer, J.B., Frankowski, D., Herlocker, J., Sen, S.: Collaborative filtering recommender systems. In: The adaptive Web, State-of-the-art survey, vol. 4321, pp. 291–324. Springer, Berlin and Heidelberg (2007)
24. Snoek, J., Larochelle, H., Adams, R.P.: Practical bayesian optimization of machine learning algorithms. In: Advances in Neural Information Processing Systems. vol. 25. Curran Associates, Inc (2012)

25. Wettschereck, D., Aha, D.W.: Weighting features. In: Case-based Reasoning Research and Development, LNCS, vol. 1010, pp. 347–358. Springer, Berlin (1995)
26. Yang, L., Shami, A.: On hyperparameter optimization of machine learning algorithms: Theory and practice. Neurocomputing **415**, 295–316 (2020)
27. Yeguas, E., Luzón, M.V., Pavón, R., Laza, R., Arroyo, G., Díaz, F.: Automatic parameter tuning for evolutionary algorithms using a bayesian case-based reasoning system. Applied Soft Computing **18**, 185–195 (2014)