https://www.aimspress.com/journal/mine

Mathematics in Engineering, 8(2): 150–180.
DOI:10.3934/mine.2026006
Received: 02 July 2025
Revised: 03 February 2026
Accepted: 18 February 2026
Published: 03 March 2026

Research article

Hyperparameter optimization of an augmented autoencoder for 6D object pose estimation via neural architecture search

Matteo Lombardi[1,*], **Davide Sapienza**[1,2], **Elena Govi**[1] **and Giorgia Franchini**[1]

[1] Department of Physics, Informatics and Mathematics, University of Modena and Reggio Emilia, Via Campi 213/B, 41125, Modena (MO), Italy

[2] Artificial Intelligence Research and Innovation Center (AIRI), University of Modena and Reggio Emilia, Via Pietro Vivarelli 10, 41125, Modena (MO), Italy

* **Correspondence:** Email: matteo.lombardi@unimore.it.

Abstract: Deep learning has become a cornerstone in numerous applications, such as robotics, augmented reality, and autonomous systems, where accurate 6D pose estimation – determining an object's position and orientation in 3D space – is critical. Despite its success, designing optimal neural architectures and tuning hyperparameters remain computationally expensive and challenging, especially in high-variance and data-intensive domains like 6D pose estimation. To address this, we investigate the application of a Neural Architecture Search (NAS) technique guided by classical Machine Learning-based performance predictors. As these predictors estimate final model performance using early-stage training data, we demonstrate that leveraging such a strategy allows for efficient exploration of the hyperparameter space, significantly reducing the computational burden of exhaustive search while still achieving improved performance. Building on an existing NAS method, we introduce a novel modification that enhances the efficiency of the search process, further accelerating convergence by nearly 71% without sacrificing accuracy. Through extensive experimentation on the LineMOD dataset, we demonstrate that our method consistently discovers high-performing configurations of an Augmented Autoencoder for 6D pose estimation, outperforming benchmark models by almost 15% in pose accuracy and 42% in reconstruction loss. These results underscore the potential of predictor-based NAS as a powerful and computationally efficient tool for neural architecture optimization in complex, real-world tasks.

Keywords: deep learning; neural architecure search; augmented autoencoder; 6D pose estimation; LineMOD; support vector machines for regression; random forest

1. Introduction

Deep Learning (DL) has gained significant importance in recent years, becoming a foundation in engineering and industrial applications, such as robotics [11, 48], computer vision [40, 42], and autonomous systems [4,41]. Its rapid development has been possible due to progress in computational resources, the availability of large-scale data, and the design of high-performing neural network architectures [6, 13, 27]. These advancements have enabled Artificial Intelligence models to achieve remarkable levels of accuracy and speed in tasks once considered unsolvable by a computer, and have made these technologies widely accessible [5, 39]. Nevertheless, as the complexity and effectiveness of DL methods increase, likewise do their demands for computational resources and time [43, 47]. Training state-of-the-art neural networks requires extensive datasets, days of computation, and energy-demanding hardware. Developing high-performing neural networks comes with numerous challenges, including the fine-tuning of several hyperparameters, which are not learned during the training process, but must be set a priori [3].

The correct choice of hyperparameters is crucial as they affect not only the model's ability to generalize and perform well, but also its computational cost in terms of runtime and memory. Hyperparameters can be tuned manually or through automated processes. Manual setting requires a deep understanding of their effects on training dynamics and exhaustive experimentation, while automated methods simplify the process but may demand significant computational resources. For cases where manual tuning is impractical due to the complexity of the model or time constraints, automatic hyperparameter optimization is essential as it seeks the best hyperparameters with minimal human effort. However, there are still some challenges to face. Firstly, function evaluations can be extremely demanding for complex models and large datasets. Besides, the hyperparameter space is high-dimensional, and it is not always clear which hyperparameter we need to optimize, nor do we know a priori in which range.

NAS

All these challenges have led to the development of automated architecture engineering, commonly known as Neural Architecture Search (NAS) [7], which seeks an optimal architecture and a hyperparameter configuration that yield the best performance for the task at hand. In a typical NAS framework, a search strategy selects an architecture from a predefined search space. The architecture is evaluated by a performance estimation strategy, which returns a performance estimate that is used to guide the search strategy's subsequent exploration of the hyperparameter space. A wide range of methods, mainly based on Reinforcement learning [22] and Bayesian optimization [24, 53], have been developed to accelerate the performance estimation phase.

According to [38], it is possible to identify three NAS paradigms: (i) weight-sharing methods; (ii) gradient-based methods; (iii) prediction-based methods. Firstly introduced in Efficient NAS (ENAS) [37], the *weight-sharing* approach involves training a single model – usually called "super-network" – to convergence, allowing subsequent sampled architectures to inherit its weights. As a result, child networks require little to no additional training. Similarly, other examples of methods following this approach are Stochastic NAS (SNAS) [55], Continual Architecture Search (CAS) [35], and Curriculum NAS (CNAS) [12]. A subset of these methods models the search space as continuous, enabling the use of stochastic gradient descent and DL models to optimize it. The

most important method employing this approach is Differentiable Architecture Search (DARTS) [31], which has given rise to numerous extensions and variants, such as Improved DARTS (I-DARTS) [23], Differentiable Neural Architecture Search for Embedded Systems (EDNAS) [32], and NAS with Shrinking-and-Expanding Supernet (SE-NAS) [20]. In this framework, each layer of the super-network is characterized by a softmax over all candidate operations in the search space, allowing differentiable methods to explore and optimize the architecture space effectively.

However, due to their reliance on shared weights, these methods are limited to ranking sampled architectures relatively to one another, without providing accurate estimates of their absolute performance. This leads to the so-called *optimization gap*, where the performance of an architecture after being re-trained from scratch does not necessarily match the performance observed during the search phase. Additionally, there is also the risk that architectures inherit inappropriate weights, further compromising the reliability of performance estimates.

Performance prediction methods, such as Progressive NAS (PNAS) [30], Neural Architecture Optimization (NAO) [34], and NAS with Gradient Boosting Decision Tree (GBDT-NAS) [33], completely avoid these drawbacks and focus on evaluating a network's final performance using data collected during a brief initial training phase. In this approach, architectures from the search space are trained for a small number of epochs and used as training samples for a Performance Predictor (PP), which learns to predict their final performance and guides the search through the hyperparameter space. By providing such estimates, PPs narrow down the search space, making it possible to bypass full training for many potential architectures. Once trained, a PP can quickly evaluate multiple configurations, identifying the most promising ones for further examination and full training. Several methods [2, 19, 52] employ another neural network as the performance estimation strategy. However, this approach can be impractical, as neural networks typically require large datasets for effective training, thereby increasing the cost and complexity of the initial dataset creation phase. Moreover, hyperparameter optimization of the PP also needs to be tackled. To address this issue, [10] and its evolution [9] proposes a PP that employs conventional Machine Learning (ML) techniques, which ensure reliable and accurate predictions while keeping computational demands low. In this work, the authors collect a dataset of varied Convolutional Neural Network (CNN) hyperparameter configurations (e.g., filter counts, kernel sizes, learning rates, batch sizes) along with their performance after only a few training iterations. They then train lightweight regressors (Support Vector Machine for Regression – SVR – and Random Forest – RF) to predict the network's final accuracy based on these early training signals. Using this SVR-RF method is particularly strategic and advantageous as it reaches high predictive power even with limited datasets, unlike neural networks, which typically require large amounts of training data. Furthermore, standard ML models also involve significantly fewer hyperparameters. Since optimizing the PP's hyperparameters is a necessary step, this makes conventional methods easier and less demanding to tune compared to neural networks. The SVR-RF-based PP is integrated into a probabilistic exploration framework, which strategically focuses the search on promising configurations, reducing the number of unnecessary evaluations.

6D pose

6 Degrees of Freedom (6DoF – 6D for short) pose estimation is a fundamental task in computer vision that seeks to determine the position and orientation of an object in an image captured by a camera. According to [18], the problem can be formalized as follows: given an image I and N objects

$O = \{O_i \mid i = 1, \ldots, N\}$ with their corresponding 3D models $\mathcal{M} = \{\mathcal{M}_i \mid i = 1, \ldots, N\}$, we want to estimate the pose $\mathbf{P}$ of each object O_i in the scene I. The pose is defined by a 4×4 matrix composed by the rotation matrix $\mathbf{R}_{3\times3}$ and the translation vector $\mathbf{t}_{3\times1}$:

$$\mathbf{P} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}. \tag{1.1}$$

6DoF object pose estimation can be categorized into three main approaches based on object model availability ($\mathcal{M}$) during training and testing. Instance-level pose estimation provides the complete set of object models $\mathcal{M} = \{\mathcal{M}_0, ..., \mathcal{M}_m\}$ during both phases, allowing precise pose estimation of known objects [16, 45, 50]. Category-level estimation lacks the exact model $\mathcal{M}_{i_0}$ but has access to similar models $\mathcal{M}_{i_1}, ...\mathcal{M}_{i_s}$ from the same category during training, enabling generalization to new instances [49]. Zero-shot novel object pose estimation operates completely model-free, predicting poses without any prior object knowledge [29, 51].

Input modalities vary across methods, including monocular RGB, RGB with depth maps (RGB-D), or depth-only data. In this paper, we will delve into the 6DoF Object Pose Estimation instance level, which aims at retrieving poses of known object instances. Especially, we are interested in monocular RGB input methods, which are more challenging with respect to RGB-D inputs. In the past years, this field evolved from traditional template-based approaches [14, 21] using geometric features like Scale-Invariant Feature Transform (SIFT) to modern DL methods. Instance-level methods can be categorized in three primary strategies. Regression-based approaches directly predict 6DoF poses from RGB regions of interest, pioneered by PoseNet [26] as the first CNN-based end-to-end method, followed by PoseCNN [54], You Only Look Once for 6D Pose Estimation (YOLO6D) [46], CosyPose [28]. Hybrid approaches like Single Shot MultiBox Detector for 6D Pose Estimation (SSD-6D) [25] and Augmented Autoencoder (AAE) [45] combine traditional templates with DL. 2D-3D correspondence methods establish feature matches between images and 3D models, recovering poses through Perspective-n-Point (PnP) algorithms with Random Sample Consensus (RANSAC) or differentiable implementations. For example, Pixel-wise Voting Network (PVNET) [36] predicts unit vectors from pixels to keypoints, creating robust vector-field representations, and Estimating 6D Pose of Objects with Symmetries (EPOS) [16] handles symmetries using compact surface fragments through Encoder-Decoder structures. Geometry-Guided Direct Regression Network (GDR-Net) [50] combines correspondence establishment with a differentiable PnP/RANSAC algorithms compose by CNNs. A promising regressor, Zebrapose [44], is based on defining the matching of dense 2D-3D correspondence as a hierarchical classification task. The last category of RGB methods that should be included is about *refinements*. As demonstrated by some recent works, a pose refinement method could significantly improve performances of 6D pose estimation. For example, PoseCNN [54] and AAE [45] incorporate ICP algorithm (Iterative Closest Point) using depth maps for refinement.

To evaluate these methods, several metrics have been proposed [17]. The Visible Surface Discrepancy (VSD) measures the inconsistency between rendered object surfaces using the predicted and ground truth poses, focusing only on visible areas and thus being robust to occlusions and symmetries. Alternative pose distance metrics include the Maximum Symmetry-aware Surface Distance (MSSD) and the Maximum Symmetry-aware Projection Distance (MSPD), which account for indistinguishable symmetrical views by minimizing over the set of equivalent poses. Other common metrics are point-based distances, such as the Average Corresponding Point Distance (eACPD) and the

Maximum Corresponding Point Distance (eMCPD), which assess the mean or worst-case deviation between aligned 3D models.

One of the most used metrics in 6D pose estimation problems is the Average Distance of Model Points (ADD) [15]. For an object model $\mathcal{M}$ – typically a mesh represented by a set of points in $\mathbb{R}^3$ and triangles – without ambiguous views (the object does not present geometric or self-occlusion-induced symmetries), the ADD metric calculates the average distance between model points transformed by the estimated pose $\hat{\mathbf{P}}_{4\times4}$ and the ground truth pose $\bar{\mathbf{P}}_{4\times4}$:

$$e_{ADD}(\hat{\mathbf{P}}, \bar{\mathbf{P}}; \mathcal{M}) = \operatorname*{avg}_{\mathbf{x}\in\mathcal{M}} \left\| \bar{\mathbf{P}}\mathbf{x} - \hat{\mathbf{P}}\mathbf{x} \right\|_2. \tag{1.2}$$

In particular, the estimated pose is considered correct if $e_{ADD} \leq 0.1d$, with d being the object diameter. Specifically, to evaluate the performance of a 6D pose estimator, we consider the ADD recall defined as:

$$ADD\ recall = \frac{\#\ estimates\ with\ (e_{ADD} \leq 0.1 \cdot d)}{\#\ total\ estimates}. \tag{1.3}$$

Contribution

After presenting the core concepts of 6D pose estimation, the focus moves to the hyperparameter optimization of an Augmented Autoencoder (AAE) [45], detailed in Section 2.1. Our focus on this 6D pose estimation CNN is motivated by its ability to learn without requiring real, pose-annotated training data. Instead, the AAE learns implicit representations from rendered 3D model views. Through the Domain Randomization strategy, the model is trained on rendered views in various semi-realistic settings, bridging the domain gap between synthetic and real images. As a result, the AAE demonstrates robustness to view symmetries, occlusions, background clutter, and dynamic environmental changes. Additionally, it does not rely on depth information, although it can be included to refine its estimations.

Building on the SVR-RF-based method – selected for the previously discussed advantages of classical ML techniques used as PPs, such as their lightweight nature, robustness, and efficiency – this paper introduces a novel and effective NAS strategy to efficiently identify the optimal hyperparameter configuration for the AAE model, detailed in Section 2.3. After providing in Section 3 deeper insights into the interplay between hyperparameters, the loss function, and ADD recall, we propose not only an alternative performance measure to assess the quality of an AAE but also a different function to explore the search space more effectively. Through extensive experimentation presented in Section 3.1, we demonstrate that relying exclusively on the reconstruction loss minimized during AAE training provides a reliable basis for performance evaluation. We then introduce a novel exponential function to compute the final performance of an AAE based on the predicted loss at convergence, by enabling faster and more efficient exploration of the search space compared to the original SVR-RF-based framework, as proved with experiments in Section 3.4. Finally, we extend and generalize the optimal configurations identified by our methodology to the 6D pose estimation of other objects within the same dataset, as shown in Section 3.5, demonstrating the effectiveness and broader applicability of our proposed methodology.

2. Material and methods

In the following, we present the AAE model and the SVR-RF-based method used to optimize its hyperparameters.

2.1. Augmented autoencoder

Augmented Autoencoder (AAE) [45] is a DL method for 6D pose object estimation. Based on the classical structure of the Autoencoder, it is designed to learn a robust and meaningful compressed representation of high-dimensional data. This is achieved using a training strategy that promotes invariance not only to noise but also to a wide range of augmentations. During the Domain Randomization, which has been proposed and described in detail in Sections 2.1.3 and 3.3 of [45], a function f_{aug} applies various augmentations, such as geometric transformations and appearance variations, to the original image x, and adds real images as background, obtaining a new randomly augmented image $\tilde{x} = f_{aug}(x)$. The encoder ϕ, based on convolutional layers, is trained to map the augmented input $\tilde{x}$ from the high-dimensional space $\mathbb{R}^n$ into the same latent representation $z = \phi(x) \in \mathbb{R}^m$, with $m \ll n$, as the original, uncorrupted data x. The decoder ψ, based on deconvolutional (transpose convolutional) layers, reconstructs the original input x from the latent representation $\tilde{z}$, obtaining $\hat{x} = (\phi \circ \psi)(\tilde{x})$. The network's parameters are learned by minimizing the bootstrapped pixel-wise ℓ_2 loss:

$$\ell_2 = \sum_{i=1}^{n} \|\boldsymbol{x}_i - \hat{\boldsymbol{x}}_i\|_2 \,. \tag{2.1}$$

After the training phase, which differs for each type of object, a codebook is generated offline by encoding each one of the N object views into a latent representation $z_i \in \mathbb{R}^m$, each associated with its corresponding ground-truth rotation matrix R_i. At test time, an RGB scene containing the object is processed using a 2D object detector to localize and crop the object. The cropped image x_{test} is resized to match the input size of the encoder and passed through it to return its latent representation z_{test}. This representation is then compared to all codes z_i from the codebook using cosine similarity. The highest similarities are identified through a k-NN (k-Nearest-Neighbor) search. The corresponding rotation matrices R_i serve as the predicted orientations of the object.

2.2. NAS via standard ML methodologies

In the context of Performance Predictors (PPs), as mentioned before, [10] introduces a SVR-RF-based strategy to estimate the performance of specific CNN architectures using only their hyperparameters and training statistics from the initial epochs of a truncated training run. Specifically, after fully training a subset of CNNs, the method samples new hyperparameter configurations and predicts their final performance based on metrics collected during the early training steps. The core idea is to train a regression model that can generalize from previously seen architectures to unseen ones, based on early learning signals.

The entire methodology can be summarized in three main stages: (i) the creation of a dataset through full training of selected architectures; (ii) the training of a performance predictor model using this dataset; and (iii) the guided exploration of the hyperparameter space using the predictor. These phases are detailed below.

2.2.1. Dataset generation

The initial phase involves creating a dataset to train the PP. Given the large size of the search space, only a subset of configurations is sampled based on available computational resources. Each sampled configuration is used to train the corresponding CNN until numerical convergence, recording performance metrics during the initial training epochs, such as loss values and accuracies, as well as their final performance after convergence. Therefore, each dataset entry consists of a hyperparameter configuration, its corresponding early training statistics, and its final performance.

2.2.2. Performance predictor

This dataset is then used to train ML models capable of predicting the final performance of unseen configurations. The authors use SVR and RF, combining their outputs in a hybrid model (HYB) that averages the predictions from both. This ensemble approach benefits from the complementary strengths of SVR and RF, resulting in improved accuracy and reduced variance in performance prediction.

2.2.3. Hyperparameter optimization

Once trained, the performance predictor is used to guide the hyperparameter optimization process.

Each hyperparameter i has a set V_i of possible values and is associated with a probability vector P_i that reflects the likelihood of its values being employed in a high-performing CNN. These probabilities are initialized using the final performance of the networks from the generated dataset and updated according to Algorithm 2.2 with Option (b).

At each iteration, a hyperparameter configuration is sampled based on the current probability distributions P_i, and the corresponding CNN is trained for a few epochs. Early performance metrics collected during this training phase are fed into SVR and RF models to predict the CNN's final performance. The predicted performance value, along with the configuration and the initial training metrics, becomes a new entry in the dataset. Probability vectors are then updated using the new data, guiding the search toward better configurations.

The update of the probability vectors repeats until all probabilities meet a convergence threshold or a maximum number of iterations is reached. Finally, the top-performing configurations are fully trained, and the one that yields the best performance is selected as the optimal hyperparameter configuration for the CNN.

2.3. Proposed methodology

To perform the hyperparameter optimization of an AAE model using the NAS strategy presented above, we first generate a dataset as described in Section 2.2.1. Each randomly selected configuration is trained up to 40000 epochs, which is the default setting in the original AAE. During each training process, we propose collecting reconstruction loss values and averaging them every 1000 epochs, up to 6000 epochs – a value selected based on extensive experimentation, as shown in Section 3.3. As a heuristic analysis will highlight in Section 3.1, using mean values instead of single loss values at isolated epochs yields a more robust and representative early-stage metric of the training behavior, which will be employed during the performance prediction phase.

Moreover, since the AAE is trained to minimize a reconstruction loss function, the final performance metric is computed by averaging this loss over the last 500 epochs. Consequently, the objective of the NAS process becomes the identification of hyperparameter configurations that lead to accurate object reconstructions.

This strategy, where better reconstruction quality is assumed to imply better pose estimation performance, was originally adopted in the AAE original work and serves as the foundational assumption upon which we developed our own NAS methodology.

In our work, we follow the same principle, using the reconstruction loss as the primary signal for guiding the search process. This assumption is further examined in detail in Section 3.2, where we provide a quantitative analysis comparing the correlation between the reconstruction quality and the ADD recall metric. This comparison highlights the validity and potential limitations of relying solely on reconstruction quality as a proxy for pose estimation accuracy.

Each entry of the generated dataset includes a sampled hyperparameter configuration, mean loss values computed every 1000 epochs up to 6000 epochs, and the final mean loss, which is the AAE performance and the label for PPs. Given the robustness and reliability of the HYB method, as demonstrated in the original work and further confirmed by our extensive experiments in Section 3.3, the resulting dataset is used to train this PP that averages the predictions of SVR and RF models.

The hyperparameter optimization and the exploration phase are carried out as described in Section 2.2.3 and Algorithm 2.1, which uses Algorithm 2.2 with Option (a) to compute the probability vectors. Once the final mean loss of an AAE model is computed, we propose to evaluate its performance using the following function:

$$g(x) = exp\left\{-\beta \, \frac{x - m}{M - m}\right\}, \tag{2.2}$$

where m and M represent the lowest and highest final mean loss values observed in the generated dataset, and x is the final mean loss of the network. The parameter β controls the degree to which sampled networks are penalized or rewarded. The exponential formulation improves the ability to differentiate between well-performing and poorly-performing AAEs, in contrast to the linear function used in the original framework. A comparison between the two functions, supported by experimental evidence in Section 3.4.1, demonstrates the effectiveness of the novel function in achieving faster and more effective convergence during the exploration of the hyperparameter space.

In summary, our proposed methodology builds upon the reconstruction-based evaluation strategy originally introduced in [10], with key adaptations that enable efficient application to the AAE's longer training regime:

(1) **Reduced training dataset via early loss signals:** We collect mean loss values every 1000 epochs (only six data points up to 6000 epochs), and use these early-stage loss statistics to generate a small yet representative training dataset. Despite the limited sampling frequency, these statistics are shown to provide robust performance signals (Section 3.1).
(2) **HYB performance predictor:** Using the reduced dataset, we train the HYB performance predictor combining SVR and RF models (Section 3.3).
(3) **Novel exponential reward function:** We propose replacing the traditional linear reward function with a new exponential formulation (Eq (2.2)) that better highlights performance differences between models, enabling faster and more effective exploration during hyperparameter optimization (Section 3.4.1).

(4) **Justification for reconstruction loss:** We argue for using reconstruction loss instead of ADD recall when exploring the hyperparameter space, due to its advantages in computational efficiency and task relevance. This choice is supported by both heuristic reasoning and empirical results (Section 3.2).

Algorithm 2.1 The hyperparameter space exploration algorithm.

$t = 0$
while $t \leq maxiter$ **and** $convergence == false$ **do**
  Set the associated probabilities P_i of the hyperparameter V_i as specified in Algorithm 2.2 with Option (a).
  Generate a realization of the random state $x^{(t)}$ according to probabilities.
  Train AAE($x^{(t)}$) for 6000 epochs.
  Compute and record the average training loss every 1000 epochs.
  Predict final performances $S\hat{V}R = SVR(x^{(t)})$ and $\hat{RF} = RF(x^{(t)})$, and compute $H\hat{Y}B = \frac{S\hat{V}R+\hat{RF}}{2}$.
  Append a new entry to the dataset consisting of the configuration $x^{(t)}$, the performance metrics from the early epochs, and the final predicted performance $H\hat{Y}B$.
  $t = t + 1$
end while

Algorithm 2.2 Probability allocation algorithm.

Initialize each vector V_i with possible values of the hyperparameters chosen. Set $v_i = |V_i|$
Initialize each vector P_i of associated probabilities with the same size of V_i.
for $i \in \{1, \ldots, n_{param}\}$ **do**
  for $j \in \{1, \ldots, v_i\}$ **do**
    Find the set D_j^i of all the dataset elements that have the feature a_j^i.
    Option (a)*: use performance function $g(x) = exp\left\{-\beta \frac{x-m}{M-m}\right\}$.
    Compute and sum all the performances of the elements in D_j^i and store the sum in P_i^j.
  end for
  Normalize the vector P_i to obtain a probability vector.
end for

3. Numerical experiments

In this section, we tackle the hyperparameter optimization of an AAE through the NAS strategy presented above, employing $\beta = 5$ in the performance function Eq (2.2). The AAE framework used in these experiments is publicly available†. The AAE model that will be optimized is designed to estimate the 6D pose of the object *ape* of the LineMOD dataset [15]. The AAE generates its own set of training images via the Domain Randomization strategy: it randomizes object poses from the Computer

*Option (b): use the original performance function $f(x) = M - x$ from [10].
†`https://github.com/DLR-RM/AugmentedAutoencoder`

Aided Design (CAD) model, augments them with various transformations, and adds cluttered real-world scene backgrounds taken from the Pascal Visual Object Classes (VOC) dataset [8]. Examples of training images can be seen in Figure 1.

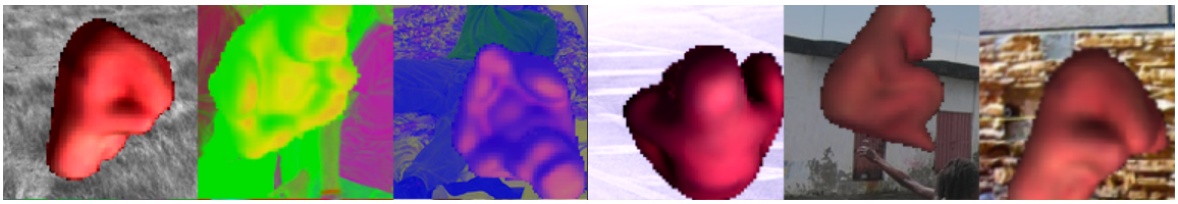


Figure 1. Examples of images created for the training dataset via the Domain Randomization strategy.

We address the optimization of key hyperparameters of an AAE model, related to both the architecture and the training process. For the architectural ones, we select the most impactful hyperparameters that alter the network's structure without compromising its conceptual integrity, whereas for the training-related ones, we focus on those that most significantly influence convergence.

The investigated values for each hyperparameter are listed in Table 1. The remaining hyperparameters, such as the number of layers or the latent dimension, are kept at their default values from the original architecture. This choice ensures a fair comparison between the NAS-designed networks and the original AAE, clearly highlighting the improvements achieved through the optimized hyperparameters.

Table 1. Investigated hyperparameters. The hyperparameter values used in the original AAE are highlighted in bold.

Hyperparameter	Values
Number of filters for the 1st layer	$V_{NK1} = \{32, 64, \mathbf{128}, 256\}$
Number of filters for the 2nd layer	$V_{NK2} = \{32, 64, 128, \mathbf{256}\}$
Number of filters for the 3rd layer	$V_{NK3} = \{128, \mathbf{256}, 512, 1024\}$
Number of filters for the 4th layer	$V_{NK4} = \{128, 256, \mathbf{512}, 1024\}$
Encoder filter size	$V_{KE} = \{3, \mathbf{5}, 7\}$
Decoder filter size	$V_{KD} = \{3, \mathbf{5}, 7\}$
Optimizer	$V_{OPT} = \{\mathbf{Adam}, Momentum, SGD\}$
Mini-batch size	$V_{MB} = \{32, \mathbf{64}, 128\}$
Learning rate	$V_{LR} = \{2 \cdot 10^{-2}, 2 \cdot 10^{-3}, \mathbf{2 \cdot 10^{-4}}, 2 \cdot 10^{-5}\}$

Since each one of the four layers can have $|V_{NK}| = 4$ different numbers of kernels, and kernels can have $|V_K| = 3$ possible dimensions both in the encoder and the decoder, the total number of configurations for the analyzed hyperparameters is given by:

$$|V_{NK}|^4 \cdot |V_K|^2 \cdot |V_{OPT}| \cdot |V_{MB}| \cdot |V_{LR}| = 82944.$$

A dataset is generated by uniformly sampling and fully training only a small fraction of the entire search space. Following the approach of [10], which employs 0.01%, the dataset would

consist of only 8 samples, which is insufficient given the greater complexity of our task. Leveraging available computational resources, we instead sample and fully train 100 networks, yielding a more informative dataset for training a more robust PP. This corresponds to approximately 0.12% of the total configuration space. Each AAE model is trained up to 40000 epochs. To incorporate early training statistics in the dataset, the loss is averaged every 1000 epochs during the first 6000 epochs. The final performance of each AAE is then computed by averaging the loss over the last 500 epochs of the training process.

Before presenting the results of the hyperparameter optimization, we first justify our choices and the modifications made to the original framework in the following sections.

3.1. Mean loss values as an early-stage metric

When relying solely on loss values, it is evident that we need a detailed description of the loss function's behavior throughout the entire truncated training process of each AAE. However, a straightforward extension of the original approach of [10] to a 6000-epoch training regime would require much more frequent sampling to accurately capture the training dynamics. Conversely, evaluating the loss function only at specific epochs might not be sufficiently informative of the overall trend of the loss function. Specifically, the points where the loss function is computed may exhibit spikes, or they may skip time frames in which the loss function has shown an inconsistent behavior, yielding unreliable information. Figure 2a,2b shows how loss values computed, for instance, at 2500, 5000, and 10000 epochs (orange line), as well as loss values computed every 1000 epochs (red line), exhibit an oscillatory behavior sensitive to possible spikes in the loss function. Figure 2c shows how considering loss values only at specific epochs may lead to ignoring important behaviors that could provide a more comprehensive understanding of the loss function's trend. Based on the information conveyed by the loss function computed at 2500, 5000, and 10000 epochs (orange line), one might conclude that the loss function is slowly descending. However, this does not consider the irregular behavior that suggests the network may not converge.

To overcome this problem, a heuristic analysis was conducted. Instead of considering loss values at specific epochs, experiments showed that the mean of the loss function values can be computed every 1000 epochs to provide a more reliable and detailed insight into the overall trend of the loss function, allowing the PP to yield better results. Figure 2 shows how this method (green line) is less sensitive to peaks and accurately describes irregular behaviors as well. Moreover, we compute the loss values averaged over the last 500 training epochs, in order to limit the influence of peaks on the final loss value. Considering only the loss values at 40000 epochs would be inconsistent in scenarios like the one shown in Figure 2d.

3.2. Unapplicability of the ADD recall

During the dataset generation, it was observed that ADD recall may not always be a perfect indicator of the overall quality of an AAE: a relatively high ADD recall can characterize some networks with poor reconstructed images. For instance, Figure 3 shows examples of inaccurate object reconstructions produced by the AAE model with the highest ADD recall (15) among the 100 sampled networks, yet a very large final loss value (0.228). As a result, a PP trained to forecast ADD recall might end up rewarding networks with a high ADD recall but inaccurate reconstructed images.

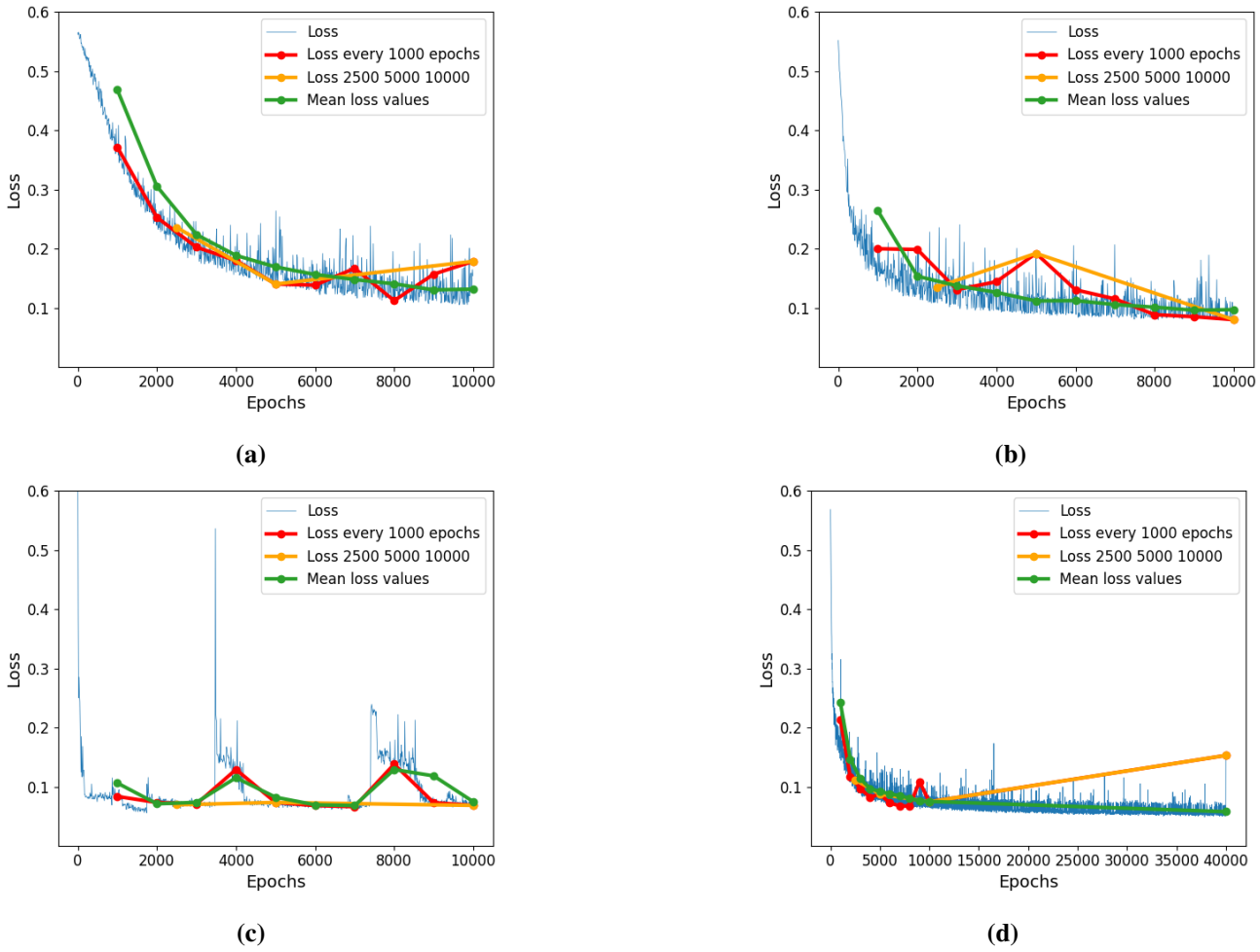


Figure 2. Spikes and inconsistencies in the loss function's trend.

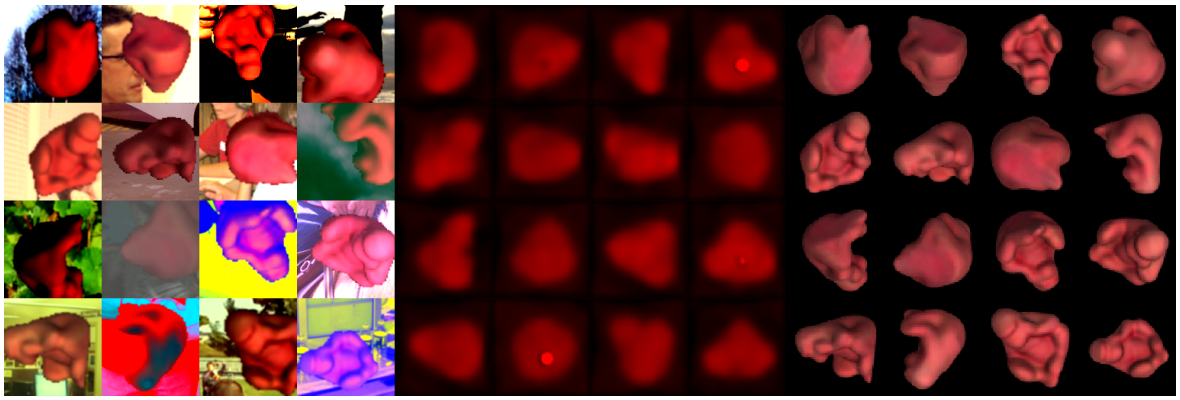
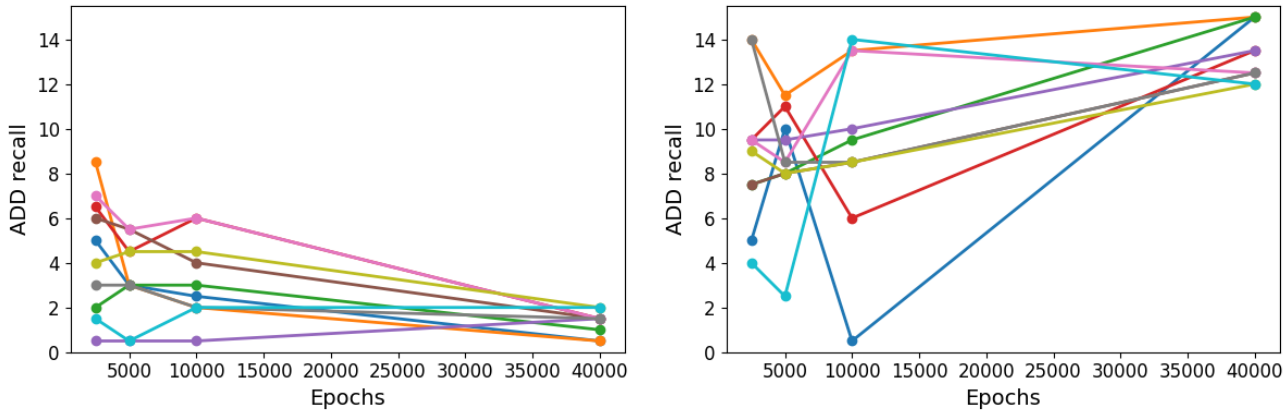


Figure 3. Inaccurate object reconstructions produced by an AAE model with high ADD recall and large final loss. On the left: augmented input images; in the center: reconstructed outputs; on the right: ground truth images.

Moreover, an unusual behavior emerges when examining the ADD recall trends of the ten best and ten worst networks in the generated dataset: the ADD recall does not always follow a monotonic trend. As shown in Figure 4a, some AAE models that perform poorly after 40000 epochs exhibit significantly higher ADD recall at earlier epochs. Conversely, as shown in Figure 4b, some of the best networks have a higher ADD recall at intermediate stages than at convergence. Other networks show an ADD recall with an oscillatory trend across the considered intervals, while only a few exhibit an ADD recall that increases as training progresses. This is heavily counterintuitive as, ideally, one may expect ADD recall to increase as training progresses. As evidence of this, Figure 5 highlights that ADD recall values computed at epochs 2500, 5000, 10000, and 40000 poorly correlate.



(a) Worst 10. **(b)** Top 10.

Figure 4. ADD recall trend.

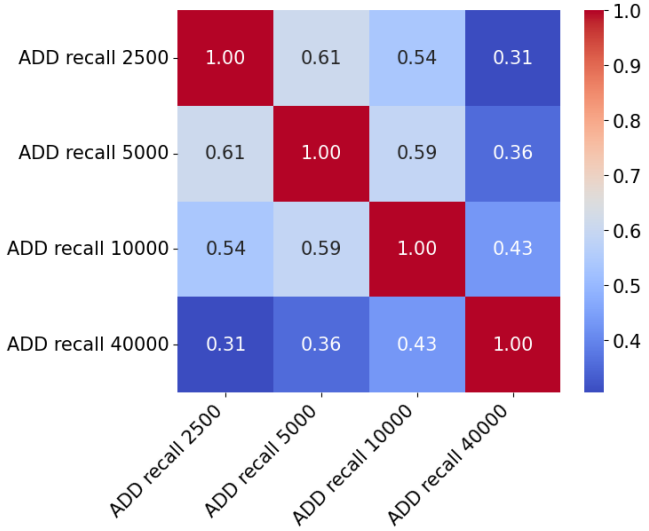


Figure 5. Kendall-τ correlation matrix between ADD recall values.

These considerations suggest that ADD recall values might introduce noise and inconsistencies,

hindering an accurate prediction of an AAE model's final performance. As we aim to obtain a reliable PP capable of assessing the final effectiveness of a new sampled architecture, it is well-posed to exclude ADD recall values and instead focus only on the information conveyed by the loss. By doing that, we avoid the need to compute embeddings, thereby saving both time and computational resources. This is because computing the ADD recall at a given epoch requires i) saving the network weights, which consumes storage, and ii) calculating the codebook, which is computationally expensive. Therefore, computing ADD recall too frequently would not be feasible.

While the global correlation between reconstruction loss and ADD recall is weak, we hypothesize that effectively minimizing the loss, and thus learning robust and meaningful latent representations, guides the search toward regions of higher pose accuracy. Empirical support for this hypothesis, based on a comparison between the initial dataset and the final optimized models, is provided in Section 3.4.2.

3.3. Training the performance predictor

The generated dataset is employed to train the HYB model, with the final performance, i.e., the final mean loss, serving as the target label for each sample. Since this PP combines the outputs of SVR and RF, both models require careful hyperparameter tuning before their use in the exploration phase. This can be effectively performed with Optuna [1]. Optuna is an automatic hyperparameter optimization framework that runs multiple trials during which a hyperparameter configuration is sampled and the corresponding model is evaluated with a specific objective function, which is typically a performance metric. After all the trials are completed, Optuna returns the set of hyperparameters that optimize the objective function.

The dataset is divided into a training set and a testing set. The training set is used by Optuna for hyperparameter tuning, while the testing set, consisting of 20% of the dataset, is used to evaluate the performance of the SVR and RF models.

The PPs are designed to estimate the final behavior of an AAE based on performance metrics collected during the early stages of the training process. As previously discussed, each newly sampled architecture is trained for up to 6000 epochs. Through extensive experimentation, we show that this represents the optimal number of epochs for training each sampled AAE, with additional training yielding no significant benefits.

For a fixed number of initial epochs, we first optimize the SVR and RF hyperparameters. Subsequently, we consider several training and testing sets on which the different PPs are trained, using the hyperparameter configurations found with Optuna, and then tested. Each testing set is created by randomly selecting 20 network configurations and their corresponding performance metrics for a fixed number of epochs, which serve as features in the PPs to predict the final performance. Predicted values are compared with the true labels, and their Mean Absolute Error (MAE) and R^2 are computed. For all training and testing set pairs, we determine the average MAE values, the corresponding standard deviations, and the average R^2 scores.

Considering 1000 distinct training and testing sets, the results of these experiments are reported in Table 2. The numerical outcomes indicate that the HYB methodology provides the best predictions, with the lowest average MAE, the smallest standard deviations, and the highest R^2 score, all of which contribute to a more robust and reliable model. Specifically, the HYB methodology exhibits the lowest MAE and the highest R^2 at 6000 epochs, while the smallest standard deviation is achieved at 5000 epochs. Since the standard deviation at 6000 epochs remains relatively low, the number of initial

epochs can be fixed at 6000. This value represents an optimal trade-off between PP accuracy and training time, as each new sampled hyperparameter configuration will be trained only up to 6000 epochs.

Table 2. Average MAE values, their related standard deviations, and average R^2 scores for 1000 random testing sets, obtained by the SVR, RF, and HYB methodologies, by taking into account performances at different numbers of initial epochs.

		1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
SVR	MAE	0.0171	0.0178	0.0151	0.0115	0.0138	0.0098	0.0100	0.0090	**0.0089**	0.0100
	STD	0.0084	0.0090	0.0068	0.0048	**0.0041**	0.0044	0.0046	0.0048	0.0043	0.0049
	R^2	0.5648	0.5537	0.7038	0.8401	0.8062	0.8804	0.8502	0.8730	**0.8925**	0.8586
RF	MAE	0.0147	0.0136	0.0112	0.0091	**0.0087**	0.0091	0.0099	0.0113	0.0102	0.0108
	STD	0.0063	0.0059	0.0064	0.0048	**0.0044**	0.0050	0.0047	0.0053	0.0051	0.0050
	R^2	0.7591	0.7570	0.7876	0.8759	0.8746	**0.8826**	0.8331	0.8038	0.8551	0.8300
HYB	MAE	0.0146	0.0144	0.0120	0.0091	0.0094	**0.0085**	0.0091	0.0094	0.0087	0.0097
	STD	0.0069	0.0071	0.0064	0.0046	**0.0035**	0.0042	0.0046	0.0049	0.0047	0.0050
	R^2	0.7205	0.7001	0.7757	0.8815	0.8854	**0.9021**	0.8529	0.8555	0.8891	0.8539

3.4. NAS results

NAS is performed using the PP introduced in the previous section, leveraging its predictions to evaluate the performance of an AAE model through the novel function presented in Section 2.3.

The exploration stops when either the maximum number of iterations is reached or the probabilities P_i for each hyperparameter i exceed a certain threshold s_i, which means that the corresponding value is the optimal hyperparameter and will, therefore, be chosen in all the following configurations. In our experiments, the maximum number of iterations is set to 400, and each threshold is defined as $s_i = \frac{2}{|V_i|}$. The threshold is equal to: $\frac{1}{3}$ for the mini-batch size, decoder kernel size, encoder kernel size, and optimizer; $\frac{1}{4}$ for the number of filters in each layer and the learning rate.

At the end of the NAS process, the 20 AAEs with the lowest predicted values are further trained up to 40000 epochs.

In the following, for some of the investigated hyperparameters, we include:

- a plot showing the probability trend for each of its possible values;
- a box plot illustrating the distribution of the final performance for all sampled AAEs across these values;
- a bar plot showing the occurrence counts of each hyperparameter value among the best 20 fully trained AAEs.

Number of filters. The probabilities of the possible values for the number of filters in each layer are all relatively close to the randomness baseline (black dashed line). This suggests that these hyperparameters may impact the AAE model's final performance less than others. Figure 6 shows the behavior of this hyperparameter for the 1st layer.

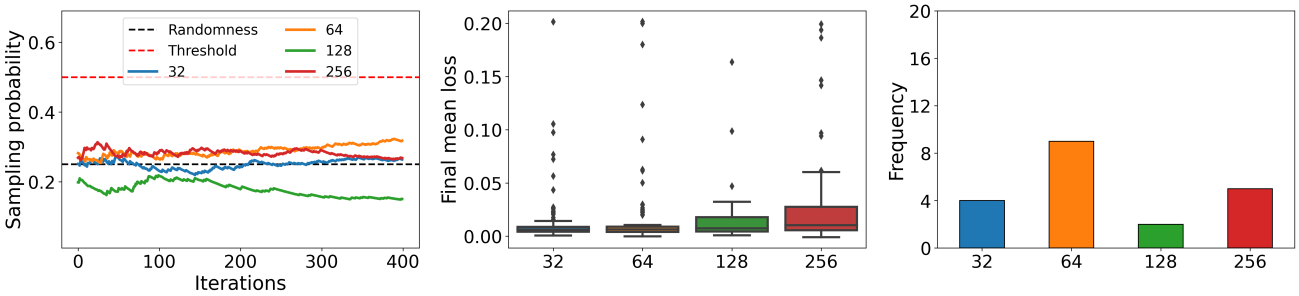


Figure 6. Number of filters for the 1st layer – proposed methodology (400 iterations).

Learning rate. The learning rate is the first hyperparameter to have one of its values reach the probability threshold, as shown in Figure 7. A learning rate of 0.002 is identified as the optimal value in just 100 iterations. Nineteen of the best networks use the optimal learning rate, including three AAEs sampled before it was even identified.

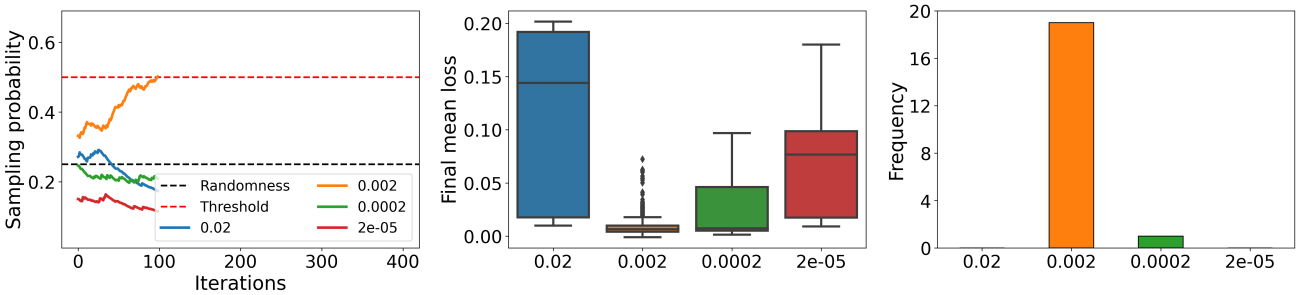


Figure 7. Learning rate – proposed methodology (400 iterations).

Optimizer. The optimizer is the second hyperparameter to reach its probability threshold. Although Adam starts with a probability of 0.5, it slowly reaches the threshold after approximately 190 iterations. This suggests that Momentum and SGD are sometimes associated with high-performing networks. However, Adam has the lowest minimum value, the lowest median, and the smallest interquartile range. All twenty of the best networks, including the six best ones sampled before identifying the best optimizer, employ Adam, as shown in Figure 8.

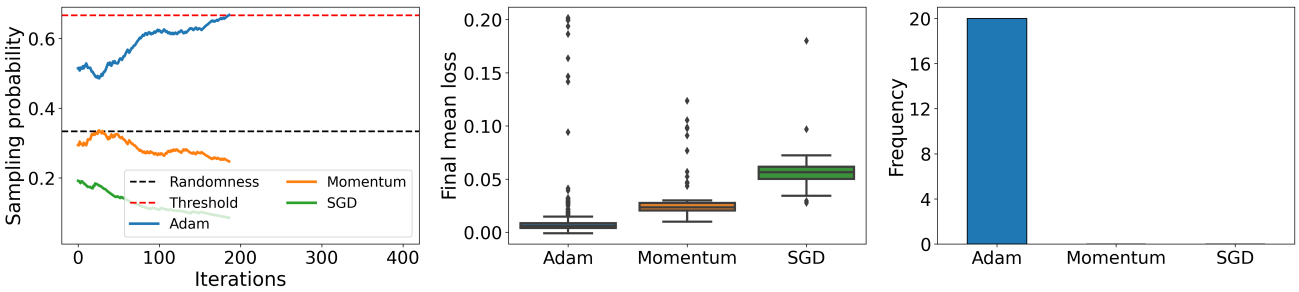


Figure 8. Optimizer – proposed methodology (400 iterations).

Convergence. Figure 9 demonstrates that updating probability vectors using the proposed formula Eq (2.2) has resulted in a fast convergence of the method. The predicted final performance shows a significantly descending behavior, highlighting the efficiency of the probability update mechanism, which has allowed the identification of two optimal hyperparameters. In fact, drastic improvements are observed when these optimal values are identified and subsequently fixed for the following sampled configurations, resulting in predictions with smaller values. While the method is not flawless, as some spikes are still evident throughout the process, they occur with much less frequency.

Supporting the idea that the method is converging effectively is the observation that:

- 17 out of the best 20 AAEs are sampled after the identification of the optimal learning rate, which occurs after 100 iterations.
- 14 out of the top-performing networks are identified after the best optimizer is determined, which occurs after 190 iterations.
- 12 of the top-performing networks are discovered within the last 150 iterations of the method.

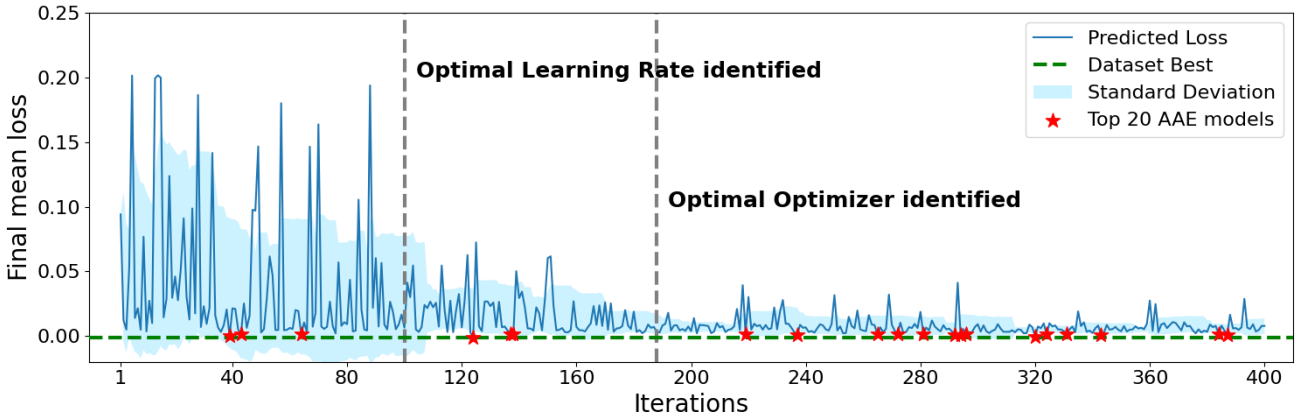


Figure 9. Predicted final performance trend of the proposed methodology – 400 iterations.

We compare the performances of these networks with two benchmark configurations: the one proposed in [45], hereafter referred to as *Sundermeyer*, and the publicly available[†] configuration, hereafter referred to as *Benchmark for 6D Object Pose Estimation (BOP)*, shown in Table 3. Although the 20 best AAEs were identified based on their final mean loss value, ADD recall remains the primary metric for 6D performance comparison.

Table 3. ADD recall and final mean loss values for Sundermeyer and BOP AAEs for object *ape*.

	NK1	NK2	NK3	NK4	KE	KD	MB	LR	OPT	ADD Recall	Final Mean Loss
Sundermeyer	128	256	256	512	5	5	64	0.0002	Adam	7.00	0.00455
BOP	128	256	512	512	5	5	64	0.0002	Adam	11.50	0.00356

Among the 20 best-performing networks, one falls short of the ADD recall achieved by Sundermeyer, and fourteen fail to meet the 6D pose performance of BOP. However, only one network has a higher final mean loss than Sundermeyer, and two exhibit a higher value than BOP.

The positive outcome of this experiment suggests potential for further improvement, therefore we expand the search by an additional 400 iterations, for a total of 800, to explore the hyperparameter space further and hopefully achieve even better results.

Figure 10 shows that no additional optimal hyperparameters are identified during the extra iterations. The predicted final mean loss continues to exhibit the same behavior observed after Adam is identified as the best optimizer.

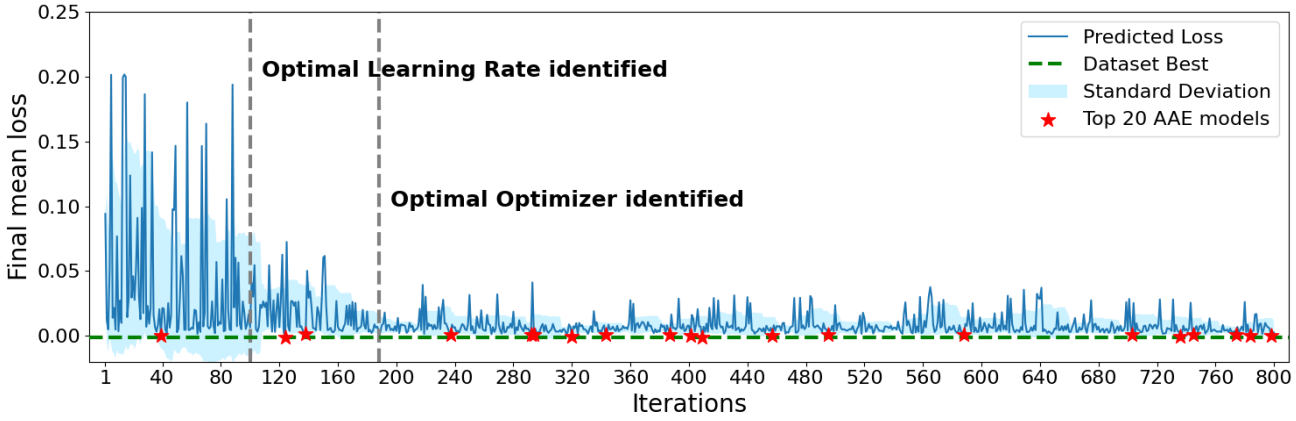


Figure 10. Predicted final performance trend of the proposed methodology – 800 iterations.

All of the 20 best-performing AAEs, reported in Table 4, surpass the ADD recall achieved by Sundermeyer, with 7 being the minimum value observed. However, fourteen of these networks still fall short of the ADD recall achieved by BOP. Nevertheless, every one of the 20 selected AAEs exhibits a lower final mean loss than both Sundermeyer and BOP, highlighting the superior reconstruction quality achieved by the optimal configurations.

The best network identified by the NAS process exhibits a negative predicted value. This is obviously an overestimation of the network's effectiveness, as a loss function cannot be negative. However, it suggests that the network may still be among the most high-performing. Indeed, it achieves an ADD recall of 9.5, and is the 2nd AAE with the lowest real final mean loss. The top-performing AAE based on the ADD recall is the 18th network with the lowest predicted final mean loss and the 7th with the lowest real final mean loss.

Table 4. 20 best AAEs obtained with the proposed methodology – 800 iterations.

NK1	NK2	NK3	NK4	KE	KD	MB	LR	OPT	ADD Recall	Predicted Loss	Real Loss
256	64	1024	1024	7	3	Adam	128	0.002	9.5	**-0.00106**	0.00215
256	128	1024	1024	7	3	Adam	128	0.002	11.0	-0.00092	0.00217
256	32	1024	1024	7	3	Adam	128	0.002	11.0	-0.00037	0.00249
256	128	512	512	7	3	Adam	128	0.002	10.0	-0.00027	0.00266
64	128	128	1024	7	3	Adam	128	0.002	13.5	-0.00010	0.00291
128	64	1024	512	7	3	Adam	128	0.002	11.0	0.00019	0.00271
64	128	256	1024	5	3	Adam	128	0.002	12.0	0.00030	0.00265
64	128	512	1024	7	5	Adam	128	0.002	13.0	0.00039	0.00226
256	256	512	1024	5	3	Adam	128	0.002	10.0	0.00044	0.00219
64	256	256	1024	5	3	Adam	128	0.002	10.5	0.00063	0.00261
64	128	1024	512	7	3	Adam	128	0.002	7.0	0.00066	0.00265
128	64	1024	1024	7	3	Adam	64	0.002	11.0	0.00072	0.00341
32	128	512	1024	5	3	Adam	128	0.002	8.0	0.00073	0.00263
128	32	512	512	7	3	Adam	128	0.002	10.0	0.00079	0.00319
32	128	256	1024	7	5	Adam	128	0.002	13.0	0.00082	0.00270
64	64	1024	1024	7	7	Adam	128	0.002	10.5	0.00087	**0.00200**
64	32	1024	512	7	3	Adam	128	0.002	9.0	0.00091	0.00286
256	256	512	1024	3	3	Adam	128	0.002	**16.0**	0.00093	0.00259
64	256	256	512	7	3	Adam	128	0.002	13.5	0.00104	0.00292
256	128	512	512	5	3	Adam	128	0.002	9.5	0.00116	0.00286
								Mean	10.95	0.00039	0.00263
								Median	10.75	0.00065	0.00265

3.4.1. Comparison with the original framework

According to the original SVR-RF-based framework, the performance of a network is assessed as the difference between the maximum loss value M observed in the original dataset and the current loss x, which can either be the final mean loss value computed for the 100 networks of the original dataset, or the one predicted by the PP for new configurations:

$$f(x) = M - x. \tag{3.1}$$

Figure 11 illustrates the randomness in the trend of the predicted final mean loss observed when using this function during the search space exploration. Contrary to expectations, the predicted final mean loss does not exhibit a strong descending behavior, suggesting that the probability update might not be sufficiently meaningful or effective within the 400 iterations considered. The final performance shows significant spikes, with their intensity remaining high even as the NAS progresses. A noteworthy improvement is observed only when the optimal learning rate is identified and fixed for the next sampled configurations, resulting in predictions with smaller values. Nevertheless, although to a lesser degree, a similar random behavior can be observed for the remainder of the NAS process.

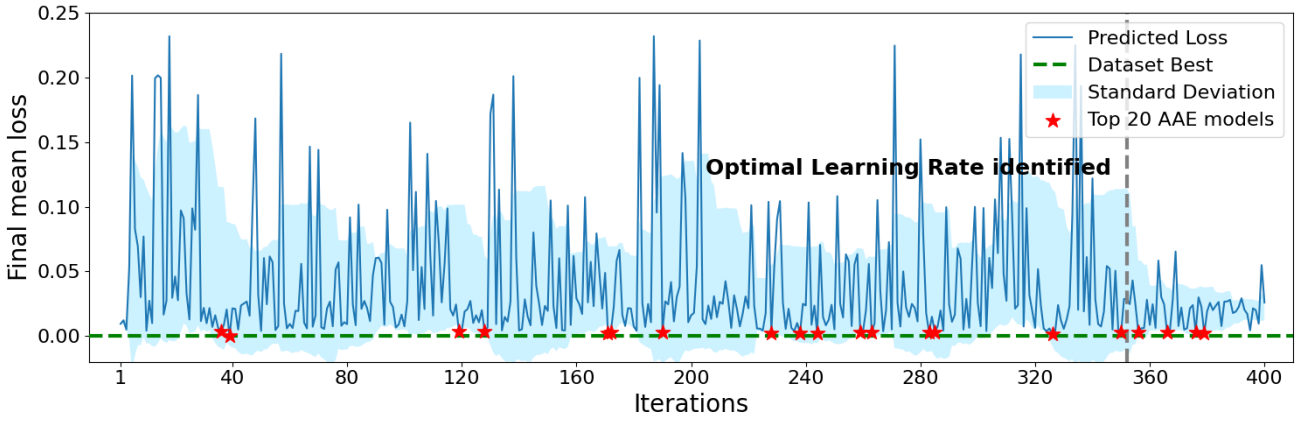


Figure 11. Predicted final performance trend of the SVR-RF-based original method – 400 iterations.

Inconsistencies in the results of certain hyperparameters suggest that the method might prioritize sampling frequency over the final performance of the sampled configurations. Even when a high final mean loss is predicted, the probabilities of the sampled hyperparameters are not properly penalized. Instead, they continue to increase. This phenomenon is evident, for instance, in the first 30 iterations, where the Momentum optimizer is sampled for half of the configurations. Despite relatively high predicted values, which would typically indicate poor performance, its probability rises and rapidly surpasses the fitness of the Adam optimizer, as can be seen in Figure 12.

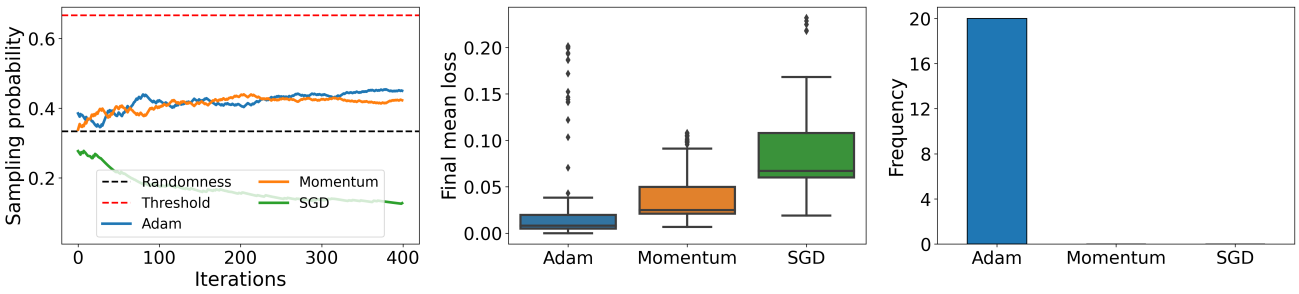
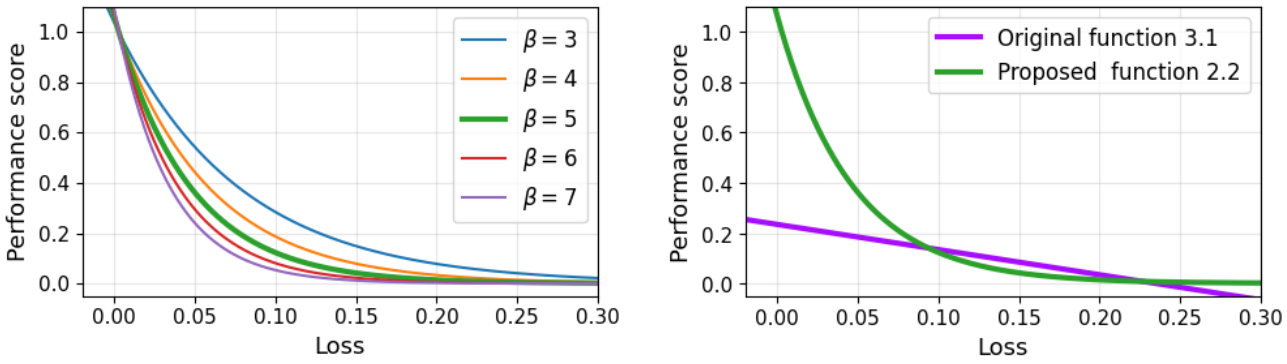


Figure 12. Optimizer – SVR-RF-based original method (400 iterations).

This irregular behavior underscores the importance of selecting an appropriate and meaningful performance function, as the entire NAS process relies on the probability of each hyperparameter value being associated with a high-performing AAE. An unsuitable choice may hinder convergence during the space search and compromise the overall effectiveness of the method.

The proposed function addresses this issue by providing greater separation between high-performing and low-performing networks, thereby facilitating a more efficient and reliable exploration of the hyperparameter space. Figure 13a illustrates the graph of this function for different values of β. After an in-depth analysis, among the tested values, $\beta = 5$ appears to strike the right balance: it is neither too strict, avoiding excessive penalization of moderately performing networks, nor does it reward exclusively the most outstanding AAEs. This function introduces a steeper slope, especially as x approaches 0, enabling clearer differentiation between high-performing and poor-performing AAEs.



(a) **(b)**

Figure 13. On the left: novel performance function (2.2) with different values of β. On the right: comparison between the original and the proposed performance function ($\beta = 5$).

Figure 13b illustrates the graph of both performance functions, offering a clear visual comparison of their behaviors and how each maps the loss values to performance scores. Moreover, Figure 14 presents the box-plots of the performance distributions of the generated dataset obtained using both the original performance function (3.1) and the proposed one (2.2), for different values of β. The results highlight the benefit of introducing the exponential formulation: the values are no longer clustered around M, and the distribution becomes more balanced. In particular, with $\beta = 5$, the interquartile range of the distribution is centered within the interval $[0, 1]$, avoiding skewness toward 0 (as observed for $\beta > 5$), indicating excessive strictness, or toward 1 (as observed for $\beta < 5$), indicating excessive leniency. This ensures that, from the very beginning of the exploration, the entire NAS framework is exposed to a balanced distribution of *good* and *bad* examples, thereby maximizing its discriminative capability.

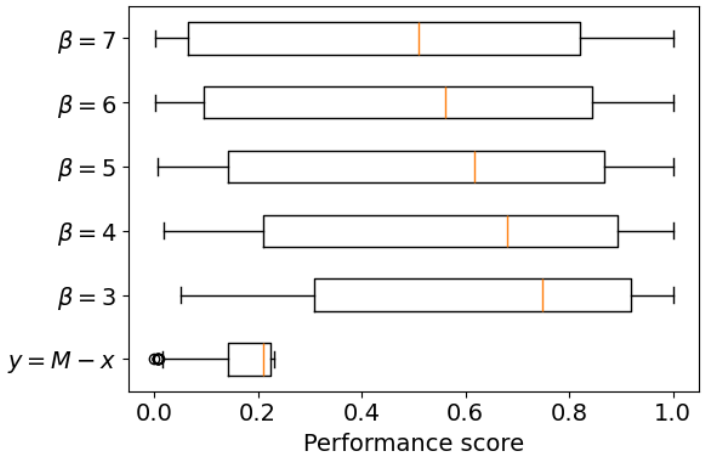


Figure 14. Comparison of the performance distributions of AAE models on the generated dataset.

This conclusion is further supported by the qualitative comparison presented in Figure 15, which correlates the visual quality of reconstructed images with their assigned performance scores. As

the reconstruction quality degrades from sharp (left) to blurry (center) and finally to unrecognizable (right), the performance metric must scale to reflect this deterioration accurately. The distribution for $\beta < 5$ proves more lenient, assigning high scores (approximately 0.8) even to noticeably blurry reconstructions, which risks rewarding sub-optimal architectures. Conversely, $\beta > 5$ behaves more strictly, causing the score to drop toward zero for intermediate-quality models, thereby suppressing useful signals during the search. The choice of $\beta = 5$ provides a reasonable balance: It correctly aligns the penalization rate with the observed visual degradation, ensuring that the search algorithm is guided by a signal that effectively distinguishes between accurate, mediocre, and poor reconstructions. Clearly, this represents just one possible suitable choice, and minor variations in β are likely to primarily affect the speed of convergence during exploration, rather than the ultimate quality of the discovered architectures.

Networks that produce poorly reconstructed images and exhibit high final loss values can still yield relatively high performance scores in the SVR-RF-based original framework. Consequently, even poorly performing networks are assigned high performance values, which significantly inflate the probabilities of hyperparameters. A penalty is applied to a hyperparameter only when it is not sampled or when the PP detects a significantly divergent AAE, leading to an extremely high predicted value. In contrast, the proposed function introduces improved differentiation among networks, enabling a more accurate assessment of performance.

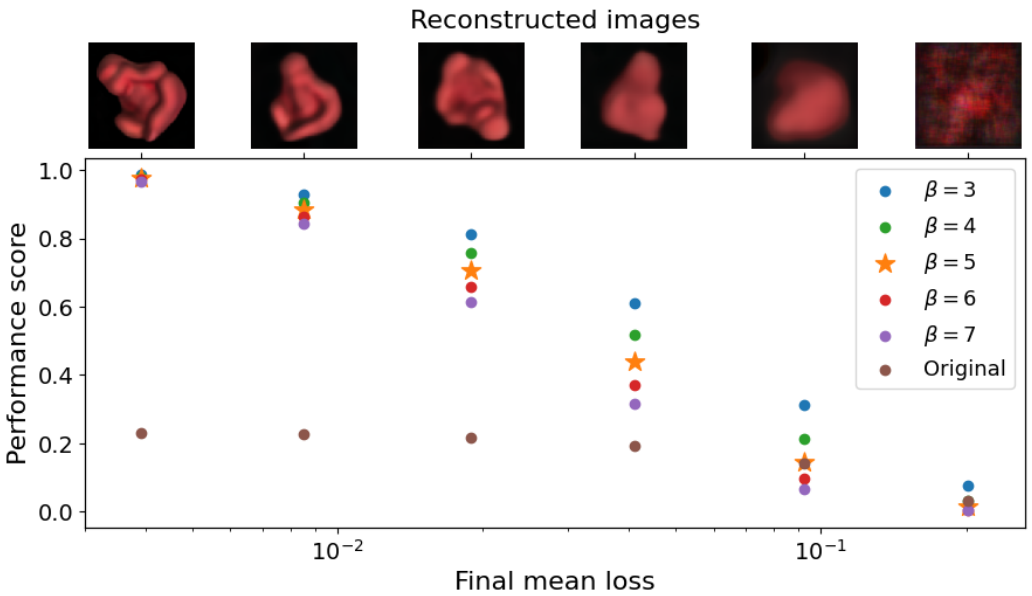


Figure 15. Reconstructed images, final mean loss, and performance score of some AAEs on the generated dataset.

The histograms in Figure 16 compare the results of three experimental setups: the experiment with the original performance function (blue), and the experiment with the proposed function with 400 (orange) and 800 iterations (green). Table 5 reports a statistical summary of these distributions. The leftmost plot shows that the difference in ADD recall is very minimal. Such behavior is expected given the inherent inconsistencies of this metric in the context of our task and the nature of our search strategy: We aimed to identify AAE models with low reconstruction loss, under the assumption that

these would hopefully correspond to high ADD recall values. However, the green distribution is less right-skewed, indicating that fewer models with low ADD recall are identified in the extended experiment using the proposed function. The middle panel illustrates predicted final mean loss values, with the experiments using the proposed function producing more optimistic predictions, especially during the additional iterations.

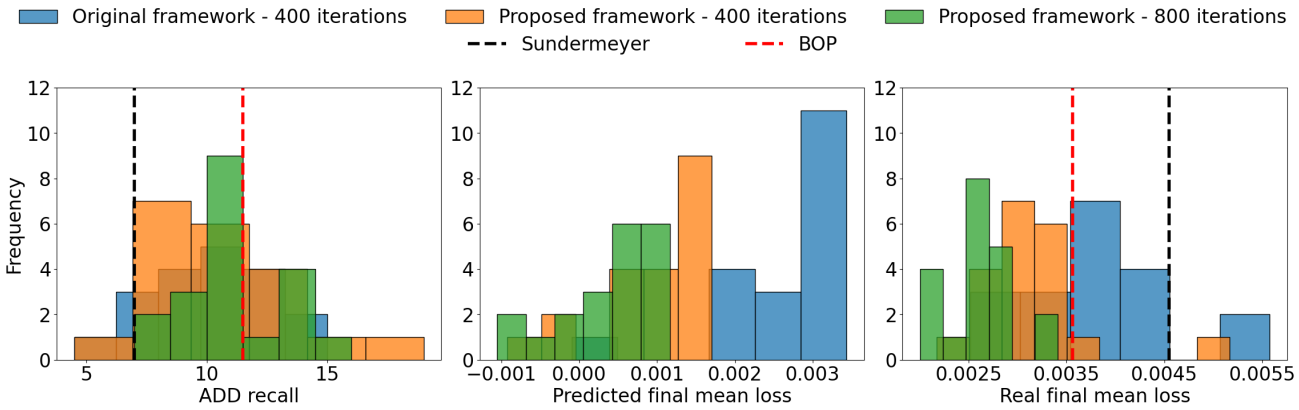


Figure 16. Distribution of ADD recall, predicted and real final mean loss across the two experiments.

Table 5. Statistics of ADD recall, predicted and real final mean loss across the two experiments.

		Original (400)	**Proposed (400)**	**Proposed (800)**
Predicted final mean loss	Mean	0.00262	0.00099	**0.00039**
	Median	0.00298	0.00119	**0.00065**
	Best	-0.00010	-0.00092	**-0.00106**
Real final mean loss	Mean	0.00647	0.00312	**0.00263**
	Median	0.00388	0.00299	**0.00265**
	Best	0.00252	0.00217	**0.00200**
ADD recall	Mean	9.68	10.20	**10.95**
	Median	10.00	9.75	**10.75**
	Min	4.50	4.50	**7.00**
	Max	15.00	**19.00**	16.00

The rightmost panel, showing real final mean loss, confirms this trend: more accurate networks were discovered using the proposed function, especially in the extended phase. A strong correlation emerges between predicted and real final mean loss across all experiments. The relative ranking of networks is preserved in both distributions, underscoring the reliability and effectiveness of the PP in guiding the exploration meaningfully across the search space.

3.4.2. Validation of the performance proxy

To prove the efficiency of the use of reconstruction loss as a proxy for pose estimation accuracy, we analyze the correlation between the two metrics for both the initial random dataset and the optimized models found by our NAS. Figure 17 presents the scatter plot of Final ADD recall versus Final Mean Loss. The blue points represent the initial random dataset, showing high variance in ADD recall in the high-loss region. However, the orange points, representing the top 20 models identified by the NAS, form a distinct cluster in the top-left region. This confirms that while the metrics are not perfectly correlated globally, minimizing reconstruction loss is a robust strategy for isolating the architectural subspace where both the lowest loss and the highest ADD recall values are achieved.

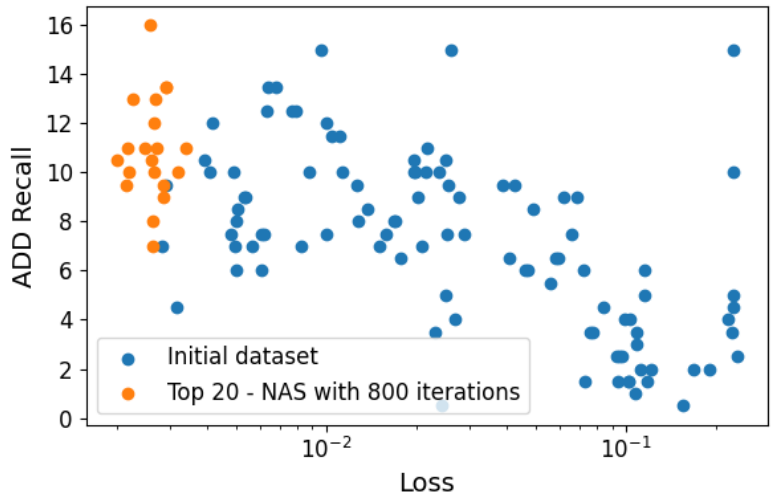


Figure 17. Scatter plot of final ADD recall vs. final mean loss.

3.5. Extension to other objects

We aim to assess whether the results are generalizable and applicable to other objects. Instead of performing a separate NAS for the AAE models of each object in the LineMOD dataset, we evaluate whether the optimal configurations derived from the NAS process tailored to the AAE for *ape* can be effectively applied to other objects to achieve accurate 6D pose estimations. This approach offers the potential to save significant amounts of time and computational resources.

In the following, we denote the best AAE configuration found during the 800 iterations with the proposed function as *NAS-2*, the network with the lowest real final mean loss as *MinLoss-2*, and the network with the highest ADD recall as *MaxADD-2*.

Referring to the comparison shown in Tables 6 and 7, the method appears very promising: For every object, at least one of the considered AAEs outperforms benchmark networks. As it is generally recommended to use a bootstrap ratio of 16 for darker objects, we also report results obtained when AAE models for the *phone* employ this hyperparameter instead of the usual value of 4. Interestingly, even with a bootstrap ratio set to 4, the AAEs obtained through NAS remain competitive in terms of both ADD recall and final mean loss.

Table 6. ADD recall for some objects of LineMOD obtained with different AAEs. (*) Bootstrap ratio 16.

	$A_{Sundermeyer}$	A_{BOP}	$A_{NAS\text{-}2}$	$A_{MinLoss\text{-}2}$	$A_{MaxADD\text{-}2}$
Ape	7.00	11.50	9.50	10.50	**16.00**
Camera	37.00	37.00	23.00	39.50	**44.50**
Duck	13.50	21.00	**22.00**	19.00	15.00
Phone	49.50	53.00	53.00	51.50	**57.50**
Phone (*)	51.00	50.00	54.00	**58.00**	43.00
Lamp	**67.00**	65.50	**67.00**	62.00	64.00
Mean	37.50	39.66	38.08	**40.08**	40.00

Table 7. Final mean loss values for some objects of LineMOD obtained with different AAEs. (*) Bootstrap ratio 16.

	$L_{Sundermeyer}$	L_{BOP}	$L_{NAS\text{-}2}$	$L_{MinLoss\text{-}2}$	$L_{MaxADD\text{-}2}$
Ape	0.00455	0.00356	0.00215	**0.00200**	0.00259
Camera	0.00851	0.00792	0.00496	**0.00403**	0.00549
Duck	0.01053	0.00921	0.00802	**0.00578**	0.00725
Phone	0.00781	0.00735	0.00444	**0.00420**	0.00531
Phone (*)	0.02441	0.02382	**0.01445**	0.01455	0.01723
Lamp	0.02377	0.02258	**0.01343**	0.01356	0.01642
Mean	0.01326	0.01241	0.00791	**0.00735**	0.00905

3.6. Efficiency and scalability

In terms of fully trained networks, the generation of the dataset requires training 100 AAEs up to 40000 epochs. Training up to 800 sampled networks for 6000 epochs is equivalent to training 120 networks to full convergence. Additionally, the hyperparameter optimization for SVR and RF can be considered equivalent to training one additional network. To drive the 20 best-performing networks to numerical convergence, starting from the checkpoint, they are trained for the remaining 34000 epochs. This is equivalent to training 17 networks for 40000 epochs. This approach proves to be significantly resource-efficient and time-saving, as the total training count amounts to 238 fully trained networks. Moreover, extending the results to other objects requires training 3 networks for each object. By training a total of only 256, benchmark results are outperformed both for the AAE model tailored to the *ape* object (for which the NAS process was designed) and for other AAE models designed to estimate the pose of other objects in the same dataset.

4. Conclusions

DL has revolutionized many fields, leading to rapid progress and improvements. However, the higher performance of these methods comes with increasing model complexity and a greater need for computational resources. One of the main challenges today is the large number of hyperparameters that significantly influence the efficiency of a DL model. As a result, managing

such complex models tailored for real-world tasks is not straightforward. This paper contributes to this problem by investigating 6D pose estimation using AAEs and the application of NAS for hyperparameter optimization. Starting from the original SVR-RF-based method [10], in this work we demonstrate the great potential of a probabilistic strategy to efficiently search the hyperparameter space applied to complex real-world deep-learning frameworks. One key finding of our study is that the choice of performance metric plays a critical role in the effectiveness of the space exploration and hyperparameter optimization. To address this, we introduce a novel modification to the original SVR-RF-based framework that improves the efficiency of the method, further accelerating convergence without compromising accuracy. Specifically, we focus solely on the information provided by the loss function. Since the AAE is inherently trained to minimize reconstruction loss, this approach is both meaningful and effective, as it provides more consistent and reliable features for guiding the NAS process, ultimately leading to improved 6D performance scores.

Experimental results extensively validate the proposed methodology. Firstly, they highlight that the proposed method leads to a faster and more effective convergence. While the SVR-RF-based original framework identifies only the optimal learning rate around epoch 350, the novel performance function allows for identifying the same optimal hyperparameter around epoch 100 and the best optimizer around epoch 190. In finding the first optimal hyperparameter, the proposed methodology is 71.42% faster than the original framework. Secondly, numerical experiments show that the optimal configurations found with the NAS process outperform benchmark AAE models from the literature on the LineMOD dataset, achieving competitive pose estimation accuracy and improved reconstruction quality. In terms of ADD recall, the proposed methodology achieves a 6.25% improvement over the original SVR-RF-based framework, a 39.13% improvement over BOP, and a 128.57% improvement compared to Sundermeyer. In terms of final mean loss, the proposed methodology achieves a 20.63% improvement over the original framework, a 43.82% improvement over BOP, and a 56.04% improvement compared to Sundermeyer.

It is worth noting that the optimal configurations obtained through NAS are associated with AAE models tailored to estimate the pose of a single, specific object. However, these configurations are extended to AAE models designed for other objects within the same dataset, proving to be more effective and accurate than the benchmarks. In terms of ADD recall, a mean improvement of 15.16% is achieved over BOP and 40.28% over Sundermeyer across all the tested objects. For the final mean loss, a mean improvement of 42.15% over BOP and 47.39% over Sundermeyer is observed. This scalability and generalizability are crucial, as optimizing hyperparameters for each new model would be labor-intensive and time-consuming. Instead, improvements achieved for an AAE model designed for one object successfully translate to others, demonstrating the broader applicability of the optimized configurations.

Our NAS strategy proves to be highly efficient: only 256 networks are trained out of 82944 possible configurations – just 0.31% of the total search space (see Section 3.6). For the *ape* object alone, the full process – sampling, predictor training, and final fine-tuning – requires the equivalent of just 238 fully trained networks. Despite this limited budget, our method outperforms benchmark results.

Crucially, the optimized architecture generalizes effectively to other objects in the LineMOD dataset. For each additional object, only 3 networks are trained, enabling practically zero-shot adaptation. This highlights the scalability and reusability of our approach across multiple LineMOD objects with minimal additional cost.

This work provides a substantial contribution to the field of 6D pose estimation by addressing the challenge of hyperparameter optimization in AAEs through a scalable and resource-efficient methodology. Beyond the immediate results, our approach lays the groundwork for future extensions. One promising direction is to dynamically update the performance predictor during the NAS process, leveraging progressively identified high-performing configurations to refine the predictor itself. This could lead to even more accurate and efficient search strategies. Nonetheless, the framework and results presented in this work already constitute a solid contribution to the field, offering a practical and scalable solution for hyperparameter optimization.

Use of Generative-AI tools declaration

The authors declare they have not used AI tools in the creation of this article.

Acknowledgments

M. Lombardi's work is partly funded by the Italian MUR through the PRIN 2022 Project *"Numerical Optimization with Adaptive Accuracy and Applications to Machine Learning"*, (project code: 2022N3ZNAX, CUP: E53D23007700006), under the National Recovery and Resilience Plan (PNRR), Italy, Mission 04, Component 2, Investment 1.1, funded by the European Commission - NextGeneration EU programme.

G. Franchini, E. Govi, and M. Lombardi's work is partly funded by "Gruppo Nazionale per il Calcolo Scientifico" (GNCS INdAM).

Conflict of interest

The authors declare no conflicts of interest.

References

1. T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, Optuna: A next-generation hyperparameter optimization framework, *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, 2623–2631. https://doi.org/10.1145/3292500.3330701
2. B. Baker, O. Gupta, R. Raskar, N. Naik, Accelerating neural architecture search using performance prediction, *arXiv*, 2017. https://doi.org/10.48550/arXiv.1705.10823
3. J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, *J. Mach. Learn. Res.*, **13** (2012), 281–305.
4. M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, et al., End to end learning for self-driving cars, *arXiv*, 2016. https://doi.org/10.48550/arXiv.1604.07316
5. T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, et al., Language models are few-shot learners, *arXiv*, 2020. https://doi.org/10.48550/arXiv.2005.14165
6. A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, et al., An image is worth 16x16 words: transformers for image recognition at scale, *arXiv*, 2020. https://doi.org/10.48550/arXiv.2010.11929

7. T. Elsken, J. H. Metzen, F. Hutter, Neural architecture search: a survey, *J. Mach. Learn. Res.*, **20** (2019), 1997–2017.

8. M. Everingham, L. Van Gool, C. K. Williams, J. Winn, A. Zisserman, The pascal visual object classes (VOC) challenge, *Int. J. Comput. Vis.*, **88** (2010), 303–338. https://doi.org/10.1007/s11263-009-0275-4

9. G. Franchini, GreenNAS: A green approach to the hyperparameters tuning in deep learning, *Mathematics*, **12** (2024), 850. https://doi.org/10.3390/math12060850

10. G. Franchini, V. Ruggiero, F. Porta, L. Zanni, Neural architecture search via standard machine learning methodologies, *Math. Eng.*, **5** (2023), 1–21. https://doi.org/10.3934/mine.2023012

11. E. Govi, D. Sapienza, S. Toscani, I. Cotti, G. Franchini, M. Bertogna, Addressing challenges in industrial pick and place: a deep learning-based 6 degrees-of-freedom pose estimation solution, *Comput. Ind.*, **161** (2024), 104130. https://doi.org/10.1016/j.compind.2024.104130

12. Y. Guo, Y. Chen, Y. Zheng, P. Zhao, J. Chen, J. Huang, et al., Breaking the curse of space explosion: towards efficient NAS with curriculum search, *Proceedings of the 37th International Conference on Machine Learning*, 2020, 3822–3831.

13. K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, 770–778. https://doi.org/10.1109/CVPR.2016.90

14. S. Hinterstoisser, C. Cagniart, S. Ilic, P. Sturm, N. Navab, P. Fua, et al., Gradient response maps for real-time detection of textureless objects, *IEEE transactions on pattern analysis and machine intelligence*, **34** (2011), 876–888. https://doi.org/10.1109/TPAMI.2011.206

15. S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, et al., Technical demonstration on model based training, detection and pose estimation of texture-less 3D objects in heavily cluttered scenes, In: A. Fusiello, V. Murino, R. Cucchiara, *Computer Vision – ECCV 2012. Workshops and Demonstrations*, Lecture Notes in Computer Science, Springer, **7585** (2012), 593–896. https://doi.org/10.1007/978-3-642-33885-4_60

16. T. Hodaň, D. Barath, J. Matas, EPOS: Estimating 6D pose of objects with symmetries, *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, 11700–11709. https://doi.org/10.1109/cvpr42600.2020.01172

17. T. Hodaň, J. Matas, Š. Obdržálek, On evaluation of 6D object pose estimation, In: G. Hua, H. Jégou, *Computer Vision – ECCV 2016 Workshops. ECCV 2016*, Lecture Notes in Computer Science, Springer, **9915** (2016), 606–619. https://doi.org/10.1007/978-3-319-49409-8_52

18. T. Hodaň, F. Michel, E. Brachmann, W. Kehl, A. GlentBuch, D. Kraft, et al., Bop: Benchmark for 6D object pose estimation, In: V. Ferrari, M. Hebert, C. Sminchisescu, Y. Weiss, *Computer Vision – ECCV 2018*, Lecture Notes in Computer Science, Springer, **11214** (2018), 19–35. https://doi.org/10.1007/978-3-030-01249-6_2

19. Y. Hu, N. Belkhir, J. Angulo, A. Yao, G. Franchi, Learning deep morphological networks with neural architecture search, *Pattern Recogn.*, **131** (2022), 108893. https://doi.org/10.1016/j.patcog.2022.108893

20. Y. Hu, X. Wang, L. Li, Q. Gu, Improving one-shot NAS with shrinking-and-expanding supernet, *Pattern Recogn.*, **118** (2021), 108025. https://doi.org/10.1016/j.patcog.2021.108025

21. D. P. Huttenlocher, G. A. Klanderman, W. J. Rucklidge, Comparing images using the hausdorff distance, *IEEE Trans. Pattern Anal. Mach. Intell.*, **15** (1993), 850–863. https://doi.org/10.1109/34.232073

22. Y. Jaafra, J. L. Laurent, A. Deruyver, M. S. Naceur, Reinforcement learning for neural architecture search: a review, *Image Vision Comput.*, **89** (2019), 57–66. https://doi.org/10.1016/j.imavis.2019.06.005

23. Y. Jiang, C. Hu, T. Xiao, C. Zhang, J. Zhu, Improved differentiable architecture search for language modeling and named entity recognition, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019, 3585–3590, https://doi.org/10.18653/v1/d19-1367

24. K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, E. Xing, Neural architecture search with Bayesian optimisation and optimal transport, *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018, 2020–2029.

25. W. Kehl, F. Manhardt, F. Tombari, S. Ilic, N. Navab, SSD-6D: Making rgb-based 3D detection and 6d pose estimation great again, *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, 1530–1538. https://doi.org/10.1109/ICCV.2017.169

26. A. Kendall, M. Grimes, R. Cipolla, PoseNet: A convolutional network for real-time 6-dof camera relocalization, *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, 2938–2946. https://doi.org/10.1109/ICCV.2015.336

27. A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, *Commun. ACM*, **60** (2017), 84–90. https://doi.org/10.1145/3065386

28. Y. Labbé, J. Carpentier, M. Aubry, J. Sivic, CosyPose: Consistent multi-view multi-object 6D pose estimation, In: A. Vedaldi, H. Bischof, T. Brox, JM. Frahm, *Computer Vision – ECCV 2020. ECCV 2020*, Lecture Notes in Computer Science, Springer, **12362** (2020), 574–591. https://doi.org/10.1007/978-3-030-58520-4_34

29. J. Lin, L. Liu, D. Lu, K. Jia, SAM-6D: Segment anything model meets zero-shot 6D object pose estimation, *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, 27906–27916. https://doi.org/10.1109/CVPR52733.2024.02636

30. C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L. J. Li, et al., Progressive neural architecture search, In: V. Ferrari, M. Hebert, C. Sminchisescu, Y. Weiss, *Computer Vision – ECCV 2018*, **11205** (2018), 19–35. https://doi.org/10.1007/978-3-030-01246-5_2

31. H. Liu, K. Simonyan, Y. Yang, DARTS: Differentiable architecture search, *arXiv*, 2018. https://doi.org/10.48550/arXiv.1806.09055

32. J. G. López, A. Agudo, F. Moreno-Noguer, E-DNAS: Differentiable neural architecture search for embedded systems, *2020 25th International Conference on Pattern Recognition (ICPR)*, 2021, 4704–4711. https://doi.org/10.1109/ICPR48806.2021.9412130

33. R. Luo, X. Tan, R. Wang, T. Qin, E. Chen, T. Y. Liu, Accuracy prediction with non-neural model for neural architecture search, *arXiv*, 2020. https://doi.org/10.48550/arXiv.2007.04785

34. R. Luo, F. Tian, T. Qin, E. Chen, T. Y. Liu, Neural architecture optimization, *arXiv*, 2018. https://doi.org/10.48550/arXiv.1808.07233

35. R. Pasunuru, M. Bansal, Continual and multi-task architecture search, *arXiv*, 2019. https://doi.org/10.48550/arXiv.1906.05226

36. S. Peng, Y. Liu, Q. Huang, X. Zhou, H. Bao, PVNet: Pixel-wise voting network for 6DoF pose estimation, *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, 4556–4565. https://doi.org/10.1109/CVPR.2019.00469

37. H. Pham, M. Guan, B. Zoph, Q. Le, J. Dean, Efficient neural architecture search via parameters sharing, *Proceedings of the 35th International Conference on Machine Learning*, 2018.

38. M. Poyser, T. P. Breckon, Neural architecture search: a contemporary literature review for computer vision applications, *Pattern Recogn.*, **147** (2024), 110052. https://doi.org/10.1016/j.patcog.2023.110052

39. A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, et al., Learning transferable visual models from natural language supervision, *Proceedings of the 38th International Conference on Machine Learning*, 2021.

40. D. Sapienza, E. Govi, S. Aldhaheri, M. Bertogna, E. Roura, È. Pairet, et al., Model-based underwater 6D pose estimation from RGB, *IEEE Robot. Autom. Lett.*, **8** (2023), 7535–7542. https://doi.org/10.1109/LRA.2023.3320028

41. C. Scribano, D. Sapienza, G. Franchini, M. Verucchi, M. Bertogna, All you can embed: Natural language based vehicle retrieval with spatio-temporal transformers, *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2021, 4248–4257. https://doi.org/10.1109/CVPRW53098.2021.00481

42. K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, *arXiv*, 2014. https://doi.org/10.48550/arXiv.1409.1556

43. E. Strubell, A. Ganesh, A. McCallum, Energy and policy considerations for modern deep learning research, *Proceedings of the AAAI Conference on Artificial Intelligence*, **34** (2020), 13693–13696. https://doi.org/10.1609/aaai.v34i09.7123

44. Y. Su, M. Saleh, T. Fetzer, J. Rambach, N. Navab, B. Busam, et al., ZebraPose: Coarse to fine surface encoding for 6dof object pose estimation, *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, 6728–6738 https://doi.org/10.1109/CVPR52688.2022.00662

45. M. Sundermeyer, Z. C. Marton, M. Durner, R. Triebel, Augmented autoencoders: implicit 3D orientation learning for 6D object detection, *Int. J. Comput. Vis.*, **128** (2020), 714–729. https://doi.org/10.1007/s11263-019-01243-8

46. B. Tekin, S. N. Sinha, P. Fua, Real-time seamless single shot 6D object pose prediction, *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, 292–301. https://doi.org/10.1109/CVPR.2018.00038

47. N. C. Thompson, K. Greenewald, K. Lee, G. F. Manso, The computational limits of deep learning, *arXiv*, 2020. https://doi.org/10.48550/arXiv.2007.05558

48. J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, S. Birchfield, Deep object pose estimation for semantic robotic grasping of household objects, *arXiv*, 2018. https://doi.org/10.48550/arXiv.1809.10790

49. H. Wang, S. Sridhar, J. Huang, J. Valentin, S. Song, L. J. Guibas, Normalized object coordinate space for category-level 6D object pose and size estimation, *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, 2637–2646. https://doi.org/10.1109/CVPR.2019.00275

50. G. Wang, F. Manhardt, F. Tombari, X. Ji, GDR-Net: Geometry-guided direct regression network for monocular 6D object pose estimation, *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, 16606–16616. https://doi.org/10.1109/CVPR46437.2021.01634

51. B. Wen, W. Yang, J. Kautz, S. Birchfield, FoundationPose: Unified 6D pose estimation and tracking of novel objects, *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, 17868–17879. https://doi.org/10.1109/CVPR52733.2024.01692

52. W. Wen, H. Liu, Y. Chen, H. Li, G. Bender, P. J. Kindermans, Neural predictor for neural architecture search, In: A. Vedaldi, H. Bischof, T. Brox, JM. Frahm, *Computer Vision – ECCV 2020*, Lecture Notes in Computer Science, Springer, **12374** (2020), 660–676. https://doi.org/10.1007/978-3-030-58526-6_39

53. C. White, W. Neiswanger, Y. Savani, BANANAS: Bayesian optimization with neural architecture for neural architecture search, *Proc. AAAI Conf. Artif. Intell.*, **35** (2021), 10293–10301. https://doi.org/10.1609/aaai.v35i12.17233

54. Y. Xiang, T. Schmidt, V. Narayanan, D. Fox, PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes, *arXiv*, 2017. https://doi.org/10.48550/arXiv.1711.00199

55. S. Xie, H. Zheng, C. Liu, L. Lin, SNAS: Stochastic neural architecture search, *arXiv*, 2018. https://doi.org/10.48550/arXiv.1812.09926

AIMS Press