Testing the Effectiveness, Convergence, and Stability of Search-Based Hyperparameter Optimizers

Students Fernando Berti Cruz Nogueira (abert036@uottawa.ca), Kelvin Mock (kmock073@uOttawa.ca)
Lecturer Dr. Shiva Nejati (snejati@uottawa.ca)

Abstract

Hyperparameter optimization is a critical but computationally expensive task for developing effective machine learning models. This report presents an empirical study comparing a Randomized Search (RS) baseline against two representative metaheuristics: a Genetic Algorithm (GA) and Particle Swarm Optimization (PSO). This selection is made to contrast two primary search strategies: global exploration (GA) and local exploitation (PSO). We evaluate the ability of these algorithms to optimize the hyperparameters of three distinct machine learning models (Decision Tree, k-Nearest Neighbors, and a Convolutional Neural Network) on the grayscale CIFAR-10 dataset Krizhevsky [2009]. To ensure a fair and balanced assessment we define a composite fitness function. We evaluate the optimizers across three quality attributes: effectiveness (solution quality), convergence (improvement over a fixed evaluation budget), and stability (consistency across runs). The empirical results will be validated using statistical tests to provide statistically grounded conclusions.

1 Introduction

The performance of machine learning models relies heavily on hyperparameters—configuration variables like learning rate and batch size that control the training process. Identifying the optimal configuration is a significant bottleneck in model development due to the high computational cost of evaluation. To address this complexity, we frame Hyperparameter Optimization (HPO) as a software verification problem. In this context, the model functions as the "software under test," where a suboptimal configuration is treated as a "defect" that causes the system to violate its performance specifications (e.g., high loss or instability). HPO therefore acts as an automated test driver, searching the configuration space to verify model performance against defined quality criteria.

1.1 Evaluation Criteria

We evaluate the optimizers across 3 quality attributes:

- **Effectiveness**: The quality of the final solution found (i.e., the best fitness score achieved).
- **Convergence**: The rate at which the algorithm improves its best-found solution over the course of the fixed evaluation budget.
- **Stability**: The consistency and reliability of the algorithm's performance across multiple independent runs (measured by variance).

1.2 Research Questions

This report seeks to answer the following research questions:

- **RQ1**: How do representative metaheuristic algorithms compare against a randomized search baseline in terms of effectiveness and convergence rates given a fixed evaluation budget?
- **RQ2**: What is the difference in performance stability between the selected metaheuristic algorithms and the randomized search baseline?

2 Problem Formulation

2.1 Representation and Objective Function

HPO is a black-box optimization problem. It happens **prior to** the actual training loop. The problem is represented by arrays of possible values of each parameter type in table 2. The objective function $f(\theta)$, which represents the model's performance for a given hyperparameter configuration θ, presents many challenges: it is computationally expensive to evaluate, it is non-differentiable, and the search space Θ is often complex and of mixed-types (continuous, discrete, and categorical). These properties make HPO suitable for search-based metaheuristic techniques.

2.2 Algorithm Selection

2.2.1 Baseline: Randomized Search

Random Search (RS) is the standard scientific baseline for HPO. Bergstra and Bengio [2012] demonstrated empirically that RS is more efficient than Grid Search for HPO. Therefore, any intelligent algorithm must demonstrate superiority over RS to be considered effective.

2.2.2 Evolutionary Genetic Algorithm

Inspired by Darwinian evolution, the Genetic Algorithm (GA) searches for optimal solutions using *selection*, *crossover*, and *mutation*. We implement a **Memetic Algorithm** variant, which includes a local search component to escape fitness plateaus. As described in Bibaeva [2018], a radius-based elitism is applied before crossover to refine the fittest individuals.

2.2.3 Particle Swarm Optimization

PSO models a swarm where individuals are influenced by both their personal best (`p_best`) and the global best (`g_best`) solutions. The velocity of each particle is updated using inertia weight (w) and acceleration coefficients (c_1, c_2), balancing exploration and exploitation.

Table 1: Optimizer Configuration Parameters

Algorithm	**Parameter**	**Value**	**Description**
Genetic Alg.	Population	30	Number of individuals per generation.
	Generations	10	Maximum total evolutionary iterations.
	Elitism	50%	Proportion of population preserved/selected.
	Radius	0.0	Memetic-local-search radius (0.15 when memetic is enabled; 0 for standard GA runs).
PSO	Particles	10	Size of the swarm.
	w (Inertia)	0.5	Inertia weight controlling velocity retention.
	c_1 (Cognitive)	1.5	Weight for personal best influence.
	c_2 (Social)	1.5	Weight for global best influence.

3 Experiment

3.1 Models and Dataset

3.1.1 Dataset

The CIFAR-10 dataset Krizhevsky [2009] is a dataset for object recognition. It consists of 32×32 colored images. There are 60000 images in total. Those images are in 10 balanced classes (so we do not need to worry about class imbalance issues).

Preprocessing We use a grayscale version of CIFAR-10. Those RGB images (32×32 pixels) are converted to grayscale using $Y = 0.2125R + 0.7154G + 0.0721B$ and normalized to $[0, 1]$.

3.1.2 Data Split

- Training Set: 45,000 images
- Validation Set: 5,000 images (for HPO fitness evaluation).
- Test Set: 10,000 images (held out for final model evaluation after HPO is complete).

3.1.3 Models

In our study, we aim to cover the following machine learning models:

Decision Tree (DT) DT is the simplest model in nature. It is tree-based which suggests making predictions based on binary predicates. Its architecture is not complex for training and it is also highly explainable to non-technical persons. It is also widely used in real-world production-grade systems like autonomous vehicles Raja Ben Abdessalem [2018]. Given its simplicity and popularity, we start our analysis on exploring what parameters minimally have to be tuned for the simplest model, and how it performs during tuning with metaheuristics. For simplicity, a prebuilt structure from Scikit-Learn [2025a] is used.

K-Nearest Neighbors (KNN) We ultilize *KNeighborsClassifier* from Scikit-Learn [2025b] which predicts based on the class of a nearest neighbor among existing data instances. From the perspective of explainability, a prior work Mock [2024] suggests that, depending on datasets, sometimes a KNN classifier could be linear-based, but in the case of an image dataset, KNN in our experiment are predicting from highly dimensional image arrays, whose predictions need to be generalized by a kernel-based explainer. The model's architecture itself is not complex but the dataset involved could be a bit heavier training task. We used it as an alternative type in our experiment.

Convolutional Neural Network (CNN) While the above models might exhibit a simpler architecture, CNN, on the other hand, is a common deep learning architecture in practical works, which helps learning image recognition tasks more efficiently. It is a fully-connected 3-layered neural network with batch normalization, which takes operations known as "convolutions". Each of which utilizes a subset of pixels, known as a "kernel", or a "filter", iteratively learn those patterns. Custom neural networks generally, in the real-world, involve far more hyper-parameters during their training, and thus, tuning them is computationally much heavier than those prebuilt surrogate models. However, bad hyper-parameters could also increase the resulting error rate of the model Bibaeva [2018]. Therefore, a suitable metaheuristic search here comes with an important role to help determine the best set of hyper-parameters with fewer resources than an exhaustive search. Figure 1 summarizes the CNN backbone used in our experiments.

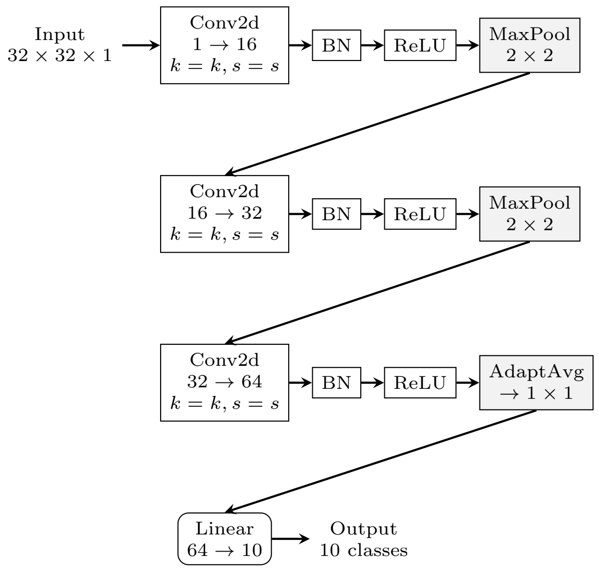

Figure 1: CNN Architecture. Note: k (kernel size) and s (stride) are optimized hyperparameters.

Significance of CNN From several existing works, an HPO problem for CNN models specifically is consistently treated as a combinatorial optimization problem, and metaheuristics are used to efficiently navigate the massive search space that grid search or manual tuning can hardly handle. A study Bibaeva [2018] applies Memetic Algorithms (MA) and Simulated Annealing (SA) to CNN architectures, using binary-encoded layers and fitness functions combining loss and accuracy, showing these metaheuristics outperform Random Search and Genetic Algorithms on common datasets like MNIST and CIFAR-10 by achieving near–state-of-the-art accuracy with far fewer evaluations. Another study Nematzadeh et al. [2022], on the other hand, highlights Genetic Algorithms (GA) and a swarm-based Grey Wolf Optimization (GWO) as effective tuning strategies, demonstrating that metaheuristics reliably improve deep model performance—including CNNs—over brute-force grid search across multiple biomedical datasets. Furthermore, a comprehensive CNN-optimization review Ibrahim et al. [2025] showcases the broad popularity of metaheuristics across CNN research, documenting a wide range of methods - PSO, Manta Ray Foraging Optimization, Reptile Search, Hybrid Wolf-Crow Optimization, and many others - successfully used to tune complicated CNN architectures, including common parameters namely learning rates and layer sizes across diverse computer-vision tasks such as melanoma detection and plant disease identification. As a summary, these works demonstrate that CNNs are the dominant deep-learning model receiving metaheuristic attention, and metaheuristic optimization has become a mainstream strategy for boosting CNN performance due to its flexibility, efficiency, and strong empirical results.

3.2 Hyperparameter Search Space

Table 2 defines the hyperparameter search space derived from the code implementation.

Table 2: Hyperparameter Search Space Definition

Model	Hyperparameter	Type	Range	Description
DT	criterion	Categorical	`['gini', 'entropy']`	Impurity measure.
	max_depth	Integer	`[3, 20]`	Max tree depth.
	min_samples_split	Integer	`[2, 20]`	Min samples to split node.
	min_samples_leaf	Integer	`[1, 10]`	Min samples at leaf.
KNN	n_neighbors	Integer	`[3, 30]`	Number of neighbors (k).
	weights	Categorical	`['uniform', 'distance']`	Weight function.
	metric	Categorical	`['minkowski', 'manhattan', 'euclidean', 'chebyshev']`	Distance metric.
CNN	learning_rate	Float (Log)	`[1e-5, 1e-2]`	Step size (log-scale).
	batch_size	Categorical	`[16, 32, 64, 128]`	Samples per gradient update.
	optimizer	Categorical	`['AdamW', 'SGD']`	Optimization algorithm.
	kernel_size	Integer	`[3, 5]`	Filter size.
	stride	Integer	`[1, 3]`	Convolution step size.
	weight_decay	Float	`[0.0, 0.01]`	L2 Regularization.

3.3 Composite Fitness

Based on searched results, our models are evaluated based on a multi-objective fitness function, which composes of a weighted sum of common metrics for machine learning models. Those weights define how important those metrics are within our evaluation. See table 3.

3.3.1 Justification

We identified an order of importance for each metric by its relative weight in the function.

Macro F1 Macro F1 is selected as the dominant term of the composite fitness because it enforces class-wise fairness during hyperparameter optimization. Unlike accuracy or micro-averaged metrics that can be inflated by majority-class performance, Macro F1 computes the unweighted mean of each class's F1 score, ensuring that every class contributes equally to the objective. This prevents the search process from converging to solutions that perform well only on frequent or easy classes while neglecting minority or harder classes. Since F1 already balances precision and recall through a harmonic mean, its macro-averaged form provides the strongest and most stable signal for steering the optimizer toward models that maintain consistent predictive quality across the entire label distribution.

Precision-Recall Thereafter, precision and recall ranks right after the macro F1 score in importance. While the precision informs us how many samples are labeled positive, it is more important to know, how many of them are truly correctly classified, and therefore, we assign a slightly above weight to the recall rather than the precision.

ROC-AUC The Receiver-Operating-Characteristic (ROC) Area-Under-the-Curve (AUC), is a measure concerning the fraction between true positive rates and false positive rates, by calculating the area bounded by a curve. It is relative to the score of a random classifier, 0.5. Of course, we aim to find whether a model, with respect to a particular set of hyper-parameters, could identify at least all truly positive instances correctly. At the same time, in our problem, we put less concern on whether those instances with positive labels (in the dataset) are misclassified. We keep its weight below macro F1 because AUC tends to be relative, which could sometimes be overly optimistic. We also need to note that, the CIFAR-10 dataset Krizhevsky [2009] shows a one-versus-rest problem, where each time we consider a target class over all others. Since the AUC score still indicates overall separability of classes, ensuring no class dominance from the comparison, it ranks slightly more important than the precision but less important than the macro F1 score.

Accuracy Accuracy is the simplest metric, which only measures the proportion of correctly classified samples (regardless of their labels) over the entire dataset. Thus, it might be too naive to inform us the insights that we could tell from other metrics.

Micro F1 As suggested by the formula of the macro F1 score above, micro F1 is just considering the precision-recall balance within 1 class. The dataset that we are in consideration consists of 10, relatively large number of classes, and so, this could even be more naive and redundant in the above macro F1's consideration.

Table 3: Composite Fitness – Weights Composition

Metric Category	**Metric**	**Weight**	**Notes**
Macro Metrics	F1 (Macro)	30%	Emphasizes balanced performance across all classes.
	Recall (Macro)	20%	Rewards models that capture minority classes.
	Precision (Macro)	15%	Penalizes false positives uniformly across classes.
Global Metrics	ROC-AUC	20%	Measures ranking quality across thresholds.
	Accuracy	10%	Overall correctness; kept low to avoid bias toward majority class.
Micro Metric	F1 (Micro)	5%	Aggregates contributions of all classes.

3.3.2 Evaluation and Analysis

3.3.3 Search Budget

Each HPO run is allocated a fixed budget of 50 fitness evaluations.

3.3.4 Stochasticity

To account for stochasticity, we perform $N = 10$ independent runs for each optimizer-model.

3.3.5 Evaluation Metrics

We will collect:

- **Effectiveness**: The distribution (mean, median, best, worst) of the final fitness score achieved across 10 runs.
- **Convergence**: The convergence trace (improvement over evaluations) for each run.
- **Stability**: The variance of the final fitness scores across the 10 runs.

3.3.6 Analysis

Performance differences are assessed qualitatively via convergence plots (search trajectory) and box plots (solution stability across runs). For quantitative comparison, we apply the Wilcoxon signed-rank test ($\alpha = 0.05$, non-parametric) to determine if observed differences are significant. The null hypothesis is that the median performance difference between any two optimizers is zero.

4 Results

This section presents the empirical findings from the 10 independent runs of each optimizer on the three models (DT, KNN, CNN).

4.1 RQ1: Effectiveness and Convergence

All optimizers improved solutions over 50 evaluations, but convergence efficiency varied by model (Figure 2). For DT and KNN, which have relatively small search spaces, GA, PSO, and RS converged rapidly to near-identical optimal configurations. Their final fitness scores were approximately 0.3384 and 0.4308, respectively (Figure 3).

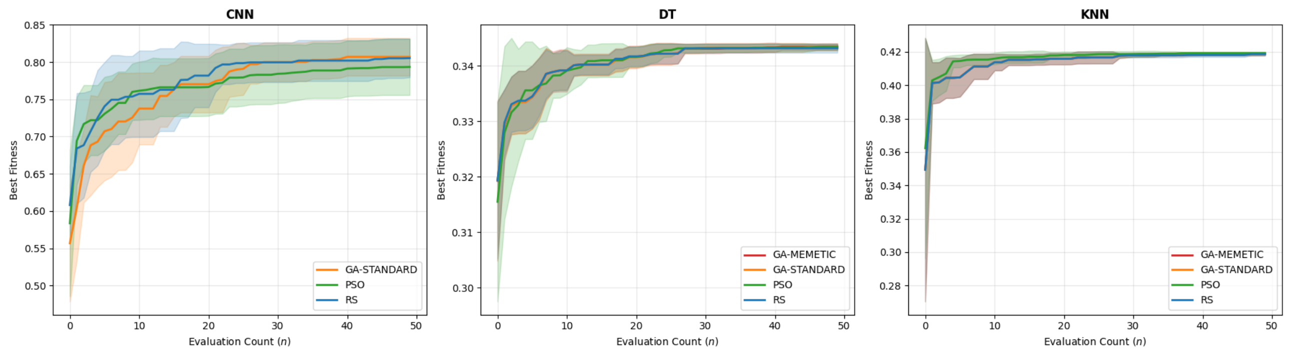

Figure 2: Best fitness convergence behavior of GA, PSO, and RS across 50 evaluations for all three models.

For the more complex CNN, GA showed steady improvement from a lower initial fitness, while RS occasionally found high-fitness configurations early (Figure 2). All optimizers converged to a similar final performance of approximately 0.77 (Figure 3).

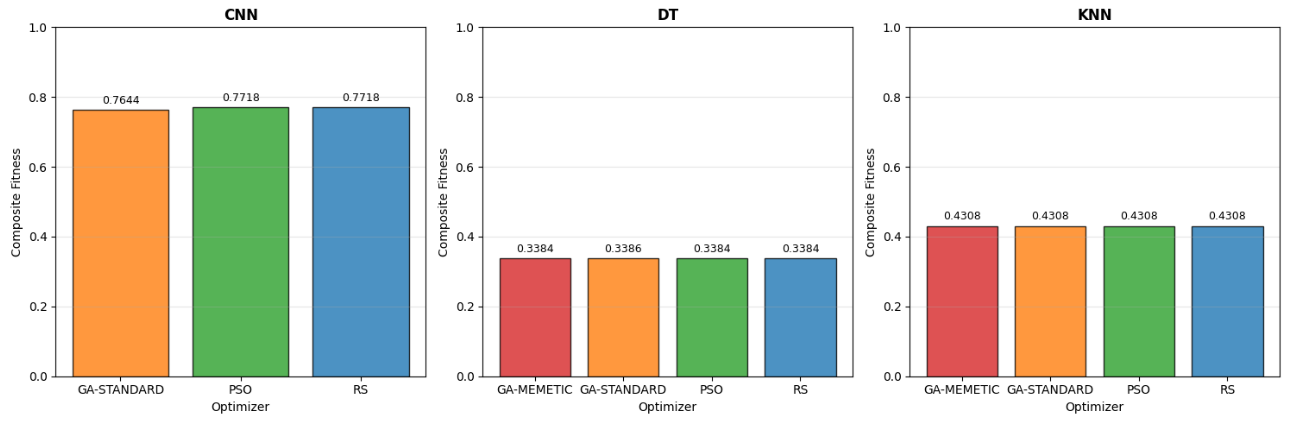

Figure 3: Final Test Performance (Composite Fitness) of the best solutions found by each optimizer.

4.2 RQ2: Stability

The stability of final solutions, shown by box plots of fitness across 10 runs (Figure 4), was high for all models. Variance was minimal for DT and KNN. For CNN, all optimizers produced similar interquartile ranges (approximately 0.79 to 0.83), with minor differences in outlier counts (GA: 1, PSO: 2, RS: 1).

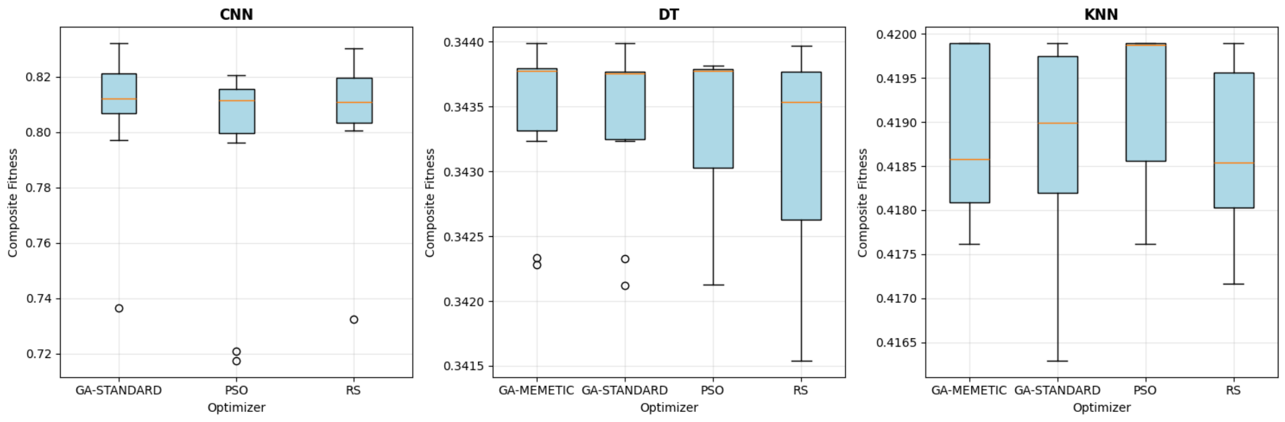

Figure 4: Box plots showing the distribution of final fitness scores over 10 independent runs.

4.3 Statistical Significance

Wilcoxon signed-rank tests showed no significant performance differences between any optimizer pairs across all models ($p > 0.05$; Table 4). The nearest to significance was PSO versus RS on KNN (p = 0.094). The memetic GA variant also showed no statistically significant improvement over the standard GA.

Table 4: Wilcoxon Signed-Rank Test Results (p-values)

Comparison	**CNN**	**DT**	**KNN**
GA-Standard vs PSO	0.322	1.000	0.426
GA-Standard vs RS	0.275	0.846	0.734
PSO vs RS	0.557	0.375	0.094
GA-Memetic vs GA-Standard	–	0.625	1.000
GA-Memetic vs PSO	–	0.770	0.344
GA-Memetic vs RS	–	0.432	0.688

5 Limitations

Insufficient Evolutionary Iterations. The fixed evaluation budget (50) relative to the population size (30) restricted the Genetic Algorithm to fewer than optimal generations. Consequently, 60% of the budget was consumed by initial random sampling, leaving insufficient evaluations for crossover, mutation, and selection to drive convergence.

Statistical Power. With 10 independent runs per configuration, the Wilcoxon signed-rank test had limited power to detect small but consistent performance differences between algorithms.

Scope Validity. Experiments were conducted only on grayscale CIFAR-10 with defined hyperparameter spaces. Results may not generalize to higher-dimensional spaces (e.g., RGB images or deeper architectures) where exploration-exploitation trade-offs could differ.

6 Conclusion

RQ1: Effectiveness and Convergence Rates against Baseline

This study evaluated metaheuristic optimizers (GA, PSO) against a Randomized Search baseline under a strict budget of 50 evaluations.

RQ2: Difference in Stability

No significant performance difference was found between searching methods across Decision Tree, KNN, or CNN models ($p > 0.05$), with all reaching similar fitness plateaus.

Further Observations

The result is explained by initialization overhead: with a population of 30, GA used 60% of its budget on initial sampling, leaving too few evaluations for evolutionary operators to yield improvement. In such micro-budget regimes (budget $< 2\times$ population), population-based methods behave similarly to Random Search.

Insufficient Budget Therefore, for hyperparameter optimization with very limited evaluations, the added complexity of metaheuristics like GA and PSO is not justified. Random Search proved equally effective baseline under these constraints.

Future Work

Further contribution should:

- Increase evaluation budgets to allow amortization of initialization costs.
- Explore adaptive or budget-aware variants of GA and PSO.
- Extend experiments to broader datasets and hyperparameter spaces.
- Use larger run counts to improve statistical power.

References

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

Victoria Bibaeva. Using metaheuristics for hyper-parameter optimization of convolutional neural networks. In *2018 IEEE INTERNATIONAL WORKSHOP ON MACHINE LEARNING FOR SIGNAL PROCESSING, SEPT. 17–20, 2018, AALBORG, DENMARK*, 2018.

Mohammed Q. Ibrahim, Nazar K. Hussein, David Guinovart, and Mohammed Qaraad. Optimizing convolutional neural networks: A comprehensive review of hyperparameter tuning through metaheuristic algorithms. *International Center for Numerical Methods in Engineering (CIMNE) 2025*, 2025. URL https://doi.org/10.1007/s11831-025-10292-x.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Kelvin Mock. Drug-consumption-machine-learning-analysis, 2024. URL https://github.com/kmock930/Drug-Consumption-Machine-Learning-analysis.

Sajjad Nematzadeh, Farzad Kiani, Mahsa Torkamanian-Afshar, and Nizamettin Aydin. Tuning hyperparameters of machine learning algorithms and deep neural networks using metaheuristics: A bioinformatics study on biomedical and biological cases. *Computational Biology and Chemistry Volume 97, April 2022, 107619*, 2022. URL https://doi.org/10.1016/j.compbiolchem.2021.107619.

Thomas Stifter Raja Ben Abdessalem, Shiva Nejati. Testing vision-based control systems using learnable evolutionary algorithms. In *ICSE '18: 40th International Conference on Software Engineering , May 27-June 3, 2018, Gothenburg, Sweden. ACM, New York, NY, USA, 11 pages.*, 2018. URL https://doi.org/10.1145/3180155.3180160.

Scikit-Learn. Decisiontreeclassifier, 2025a. URL https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.

Scikit-Learn. Kneighborsclassifier, 2025b. URL https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html.