EMERALD-O: Efficient Multi-Agent Reinforcement Learning Framework for optimised Deep Learning Hyperparameter Tuning and Selection

Akhila VH[*], Anu Mary Chacko[1] and Ponnurangam Kumaraguru[2]

[*]Computer Science and Engineering, National Institute of Technology, Calicut, Calicut, 673601, Kerala, India.
[1]Computer Science and Engineering, National Institute of Technology, Calicut, Calicut, 673601, Kerala, India.
[2]International Institute of Information Technology, Gachibowli, Hyderabad, 500032, Telangana, India.

*Corresponding author(s). E-mail(s): sna.nitc@gmail.com;
Contributing authors: anu.chacko@nitc.ac.in; pk.guru@iiit.ac.in;

Abstract

Traditional hyperparameter tuning methods, such as Bayesian Optimization and Grid Search, often prove computationally expensive and inefficient for complex deep learning architectures. This paper introduces the Multi-Agent Reinforcement Learning (MARL) framework EMERALD-O to optimize deep learning networks. The MARL-based approach utilizes two specialized agents, Agent1 focuses on data augmentation and Agent 2 on managing the learning rate and optimizer selection. The agents operate within an environment that simulates the model's training dynamics and uses validation accuracy as the reward signal. Agent performance is enhanced through epsilon-greedy exploration and experience replay mechanisms. EMERALD-O performs favorably 88.59 % with improved classification accuracy and training efficiency. The framework exhibits adaptability to diverse dataset characteristics, underscoring scalability and robustness. The framework was validated on different models built for image classification problem on Efficientnet, VGG16 and VGG19. The results highlight the potential of reinforcement learning to fine-tune complex neural network architectures and suggest that MARL can serve as a powerful tool to improve the performance of deep learning models. EMERALD-O can contribute by advancing the frontier of deep neural optimization, demonstrating that reinforcement learning can fundamentally transform the model-tuning approach. This framework establishes a new paradigm for automated hyperparameter optimization and provides a systematic lens for analyzing the behavior of the deep learning model across various hyperparametric configurations. By quantifying model responsiveness to parameter variations, this approach enables deeper insights into architectural characteristics and performance dynamics, facilitating both the theoretical understanding and practical optimization of deep learning systems.

Keywords: Multi-Agent Reinforcement Learning, Hyperparameter Optimization, Deep Neural Networks, Automated Machine Learning, Image Classification, Experience Replay

1 Introduction

Deep neural network optimization is one of the most challenging frontiers in modern machine learning, where the intricate interplay between architecture design and hyperparameter configuration can dramatically impact model performance [1]. The evolution from grid-search methodologies to sophisticated optimization techniques mirrors the field's trajectory from simple perceptrons to complex architectures. Despite the remarkable advances in neural network design, the fundamental challenge of hyperparameter optimization remains a bottleneck, particularly as architectures scale to hundreds of millions of parameters [2]. Traditional approaches, while methodologically sound, are limited in their ability to efficiently navigate the exponentially expanding search space of modern deep learning models, necessitating a paradigm shift in optimization strategies [3].

Hyperparameter optimization plays a crucial role in deep learning, significantly influencing model performance, training efficiency, and generalization capabilities. As deep neural networks continue to grow in complexity, the process of selecting optimal hyperparameters, such as learning rates, regularization coefficients, batch sizes, and architectural configurations, has become increasingly challenging. Traditional hyperparameter optimization methods include grid search, random search, Bayesian optimization, and gradient-based approaches. Although these methods have been effective in certain scenarios, they are often limited in terms of computational efficiency, adaptability to dynamic training environments, and handling high-dimensional hyperparameter spaces.

The emergence of more sophisticated optimization techniques marked a significant advancement in this field. Bayesian optimization emerged as a prominent solution, offering a probabilistic approach to modeling the relationship between hyperparameters and model performance. The Bayesian optimization method is more efficient than previous approaches as it uses surrogate models to guide the search process. Population-based training methods followed, introducing concepts from evolutionary algorithms [4] to simultaneously train and optimize neural networks. These methods enable real-time tuning during model training by combining parallel search with sequential optimization. Although effective in many scenarios, population-based methods often lack structured communication between agents and may struggle with local optima due to their evolutionary nature.

Gradient-based hyperparameter optimization techniques [5] also gained traction, attempting to directly optimize hyperparameters through gradient descent. However, these methods often struggled with discrete parameters and non-differentiable objective functions [6]. AutoML frameworks emerged next, offering automated solutions for model selection and hyperparameter tuning, although they frequently faced challenges while handling intricate interdependencies between different model components [7]. AutoML platforms like Google's AutoML and Auto-Keras offer end-to-end solutions for neural architecture search and hyperparameter optimization. However, these systems typically treat the optimization process as a black box, providing limited insights into the underlying optimization dynamics and often requiring substantial computational resources.

The application of reinforcement learning (RL) for hyperparameter optimization has gained significant traction in recent years. RL-based methods can dynamically adjust hyperparameters during training, learning from experience to optimize performance without requiring exhaustive search or differentiable objective functions. In particular, adaptive dynamic programming (ADP) approaches have emerged as powerful tools to solve complex optimization problems under uncertainty. As demonstrated by [8], ADP-based prescribed-time control for nonlinear time-varying delay systems provides a robust framework for achieving optimal control with temporal guarantees. Their work highlights how reinforcement learning technologies can effectively handle uncertainty while maintaining optimality principles directly applicable to hyperparameter optimization in dynamic training environments.

Despite these advancements, each approach has its limitations while handling the growing complexity of modern deep learning architectures. Traditional methods often fail in exploring vast hyperparameter spaces efficiently, while single-agent optimization techniques can become trapped in local optima. Deep reinforcement learning for hyperparameter optimization has shown promise in recent studies. However, most existing approaches employ single-agent frameworks that may not fully exploit the potential of collaborative exploration and structured information sharing. Moreover, they often lack explicit mechanisms for balancing exploration and exploitation across different hyperparameter dimensions simultaneously. These challenges indicate the need for more innovative solutions.

Multi-agent-based optimization techniques have emerged as a powerful approach to enhance the performance and efficiency of deep learning models [9]. These techniques leverage the collaborative capabilities of multiple agents to tackle complex optimization problems in ways that traditional methods often struggle to match. The application of multi-agent reinforcement learning (MARL) to the optimization of deep learning models offers several key advantages over conventional methods such as grid search or random search [10]. It enables efficient exploration of the hyperparameter space through parallel processing and collaborative learning among agents. This approach can significantly reduce the time required to find optimal configurations, a critical factor in the development of large-scale deep learning models. Furthermore, MARL-based optimization demonstrates superior adaptability to dynamic environments and noisy data [11], making it particularly well suited for real-world applications where conditions may fluctuate.

In addition, quantized control methods for interconnected PDE systems via mobile measurement and control strategies demonstrate how resource-efficient techniques can effectively manage complex interconnected systems. [12] showcases the benefits of mobile strategies in improving dynamic response while reducing the number of sensors and actuators required concepts that can be adapted to optimize the exploration-exploitation trade-off in hyperparameter search.

MARL represents a fundamental advancement in the domain of hyperparameter optimization for deep learning architectures. While conventional optimization methodologies operate within a singular decision-making framework, MARL introduces a distributed concurrent optimization paradigm that fundamentally transforms the approach to neural network tuning [11]. Recent advances in dynamic event-triggered consensus control for interval type-2 fuzzy multi-agent systems have shown how multiple agents can collaboratively achieve system-wide objectives while handling uncertainty through fuzzy logic frameworks [13]. These approaches provide valuable insights for designing cooperative optimization strategies in which multiple agents explore different regions of the hyperparameter space simultaneously.

The architectural sophistication of MARL lies in its implementation of parallel search mechanisms coupled with sophisticated inter-agent communication protocols [10]. This framework enables the simultaneous exploration of multiple regions within the hyperparameter space, facilitated by structured information exchange between agents. The system's design allows for the incorporation of multiple optimization strategies, each potentially specialized for different aspects of the search space. Furthermore, quantized control methods for interconnected PDE systems via mobile measurement and control strategies demonstrate how resource-efficient techniques can effectively manage complex interconnected systems. The work of [12] showcases the benefits of mobile strategies in improving dynamic response while reducing the number of required sensors and actuators, concepts that can be adapted to optimize the exploration-exploitation trade-off in hyperparameter search.

Image classification [14], a fundamental application in computer vision, exemplifies the complexity of deep learning optimization. Achieving high accuracy in this domain often requires meticulous tuning of numerous hyperparameters, extensive data augmentation, and iterative adjustments to model architecture. Traditional approaches, such as grid search or Bayesian optimization [15], can struggle to navigate this vast hyperparameter space effectively, especially when computational resources are limited. These limitations highlight a critical need for a more integrated and adaptive framework that can address hyperparameter tuning, data augmentation, and architectural refinement in a cohesive manner [16], [17], [18].

To address these limitations, this paper proposes EMERALD-O (efficient multi-agent reinforcement learning framework for deep learning hyperparameter tuning and selection), a novel approach that leverages multiple specialized agents working collaboratively to tune hyperparameters. Each agent is responsible for exploring specific aspects of the hyperparameter space, with structured communication channels that allow information sharing and coordinated exploration strategies. This framework draws inspiration from recent advances in multi-agent reinforcement learning (MARL) for optimization problems. Unlike the static strategies used in previous approaches, EMERALD-O integrates online learning mechanisms that enable real-time adaptation based on performance feedback. This dynamic approach aligns with the principles demonstrated in ADP-based control systems, where adaptive mechanisms continuously refine control strategies to achieve optimal performance despite system uncertainties. The main contributions of this paper are as follows:

1. A novel multi-agent reinforcement learning framework for hyperparameter optimization that employs specialized agents to explore different aspects of the hyperparameter space simultaneously.
2. An efficient exploration-exploitation balance mechanism that dynamically adjusts search strategies based on performance feedback and agent interactions.
3. A structured communication protocol that enables agents to share information and coordinate their exploration efforts, leading to more efficient hyperparameter optimization.
4. Comprehensive empirical evaluations to demonstrate EMERALD-O's superior performance compared to state-of-the-art hyperparameter optimization methods across multiple deep learning architectures and datasets.

The remainder of this paper is organized as follows. Section 2 formulates the problem and introduces the basic concepts. Section 3 reviews related work in hyperparameter optimization and multi-agent reinforcement learning. Section 4, presents the EMERALD-O framework in detail. Section 5 describes the experimental setup. Section 6 presents and discusses the results of these experiments. Finally, Section 7 concludes the paper and outlines directions for future research.

2 Problem Statement

This study tackles the challenge of efficient hyperparameter optimization in deep learning models by leveraging a multi-agent reinforcement learning (MARL) framework. Given a fixed neural network architecture, a training dataset, and a defined hyperparameter search space $H = \{h_1, \ldots, h_n\}$, the goal is to obtain the optimal hyperparameter set $h^* \in H$ that produces the highest validation accuracy on the given architecture and dataset.

2.1 Formal Problem Formulation

Let M represent a deep neural network architecture and let $D = \{D_{\text{train}}, D_{\text{val}}\}$ denote the dataset partitioned into training and validation sets. The hyperparameter optimization problem can be formally defined as:

$$h^* = \arg\max_{h \in H} A(M(h), D_{\text{val}}) \tag{1}$$

where A represents the validation accuracy function, and $M(h)$ denotes the model trained using hyperparameters h on the training dataset D_{train}. The environment E maintains a state space S defined by the tuple of critical hyperparameters and performance metrics:

$$S = \{\eta, \omega, \alpha_v, L_{ma}\} \tag{2}$$

where η is the learning rate, ω is the optimizer configuration, α_v is the validation accuracy, and L_{ma} is the moving average loss. In the EMERALD-O framework, the hyperparameter space H is decomposed into subspaces explored by two specialized agents:

$$H = H_1 \times H_2 \tag{3}$$

where: $H_1 = \{T(x) \in \{T_1, \ldots, T_n\}\}$ represents data augmentation transformations managed by Agent 1 (A_1) and $H_2 = \{\eta \in \mathbb{R}^+\} \times \{\omega \in \{\text{Adam}, \text{SGD}, \text{RMSprop}\}\}$ includes learning rate and optimizer

2.2 Evaluation Criteria

The performance of the EMERALD-O framework is evaluated on the following criteria:

1. Validation Accuracy: The primary metric is the validation accuracy achieved with the discovered hyperparameters:

$$A(h^*) = A(M(h^*), D_{\text{val}}) \tag{4}$$

2. Training Efficiency: The number of training iterations required to identify optimal hyperparameters compared to traditional methods such as grid search and Bayesian optimization.
3. Generalization Capability: The framework's ability to maintain a high performance across different model architectures (EfficientNet, VGG16, VGG19) and datasets.
4. Adaptability: The framework's ability to dynamically adjust exploration strategies based on observed performance is measured by improvements in validation accuracy throughout the optimization process.

Through a Markov Decision Process formulation, both agents interact with the environment E and select actions according to their respective policies $\pi_i(a_t|s_t)$. The environment generates rewards $r_t = f(\alpha_v)$ based on validation accuracy, guiding agents towards optimal hyperparameter configurations.

3 Literature Review

Multi-agent reinforcement learning (MARL) offers a powerful new paradigm for hyperparameter optimization in deep learning, addressing the limitations of traditional approaches. This section presents a chronological analysis of how optimization techniques have evolved from early foundations to contemporary multi-agent systems.

3.1 Foundations of Deep Learning and Early Optimization (1968-2012)

The foundations of modern deep learning optimization techniques trace back to pioneering work in probability distributions by Chow and Liu [19], who developed methods to approximate discrete probability distributions with dependence trees. Pearl [20] later expanded these concepts with his seminal work on probabilistic reasoning in intelligent systems, establishing frameworks that would influence reinforcement learning approaches.

Early conceptual foundations for multi-agent systems emerged with Bucsoniu et al.'s overview of multi-agent reinforcement learning [21], exploring how multiple agents could learn policies simultaneously. The modern deep learning era was catalyzed by the breakthrough of Krizhevskii et al. with AlexNet [14], demonstrating the remarkable effectiveness of convolutional neural networks for image classification tasks.

During this foundational period, hyperparameter optimization relied primarily on manual tuning or grid search, which became increasingly impractical as model complexity grew [22]. Early approaches to address this limitation included Bayesian optimization techniques, introduced by Snoek et al. [15], offering more efficient exploration of the hyperparameter space compared to exhaustive search methods.

3.2 Evolution of Optimization Techniques (2014-2018)

The period between 2014 and 2018 saw significant advances in both neural network architectures and optimization techniques. Simonyan and Zisserman's work on very deep convolutional networks (VGG) [23] demonstrated the benefits of increased network depth, further complicating hyperparameter optimization challenges. Alom et al. [24] synthesized a comprehensive understanding of deep learning approaches, following the evolution from AlexNet to more sophisticated architectures.

A notable advancement came from DeepMind in 2017 with Population-Based Training (PBT) [25], which combined parallel search with sequential optimization. PBT maintains a population of models that train in parallel, each initialized with distinct hyperparameters. Periodically, the models are evaluated, and the low-performing ones adopt the hyperparameters of the higher-performing models, often with random mutations to explore new configurations. This evolutionary approach marked a significant shift from previous optimization techniques by enabling real-time hyperparameter tuning during training.

Another important development was the introduction of HyperBand [26] in 2018, described as a novel bandit-based approach to hyperparameter optimization. HyperBand improved efficiency by adaptively allocating computational resources and implementing early stopping mechanisms for non-promising configurations, addressing growing concerns over computational costs.

Table 1 provides a chronological perspective of how optimization techniques evolved during this period and continued to develop in subsequent years, highlighting the progression from single-agent approaches to more collaborative multiagent systems.

3.3 Data Augmentation Revolution (2019-2022)

The period from 2019 to 2022 witnessed a revolution in data augmentation techniques, beginning with Google AI's introduction of AutoAugment by Cubuk et al. [27]. AutoAugment represented a paradigm shift by using reinforcement learning to discover optimal data augmentation policies rather than relying on handcrafted strategies. This

Optimization Technique	Year	Description	Possible Improvement
Population-Based Training (PBT) [25]	2017	Introduced by DeepMind, PBT combines parallel search with sequential optimization. It maintains a population of models with different hyperparameters, which are periodically evaluated. Low-performing models adopt hyperparameters from better-performing models, often with mutations to explore new configurations.	Implement structured communication between models and reduce computational costs. Introduce agent collaboration for more efficient exploration of hyperparameter space.
Bayesian Optimization and HyperBand (BOHB) [26]	2018	Combines Bayesian optimization with HyperBand to accelerate hyperparameter tuning. BOHB evaluates configurations in parallel, using early stopping for poor-performing configurations. This reduces computation time and focuses resources on promising configurations.	Introduce agent collaboration and communication to optimize exploration strategies. Expand BOHB beyond hyperparameter optimization to include neural architecture search and training strategy optimization.
AutoAugment [27]	2019	Developed by Google AI, AutoAugment uses reinforcement learning to discover optimal data augmentation policies. The agent learns from feedback based on validation performance and searches for augmentation combinations that improve generalization.	Introduce multiple agents to collaboratively explore the augmentation space and other aspects of model optimization, such as architecture search and hyperparameter tuning. Improve scalability for larger datasets and models.
AutoML for Model Compression (AMC) [28]	2019	A reinforcement learning-based framework that uses Deep Deterministic Policy Gradient (DDPG) to learn compression policies for different layers of a model. Each agent focuses on a specific layer to determine optimal compression ratios. AMC is hardware-aware and uses progressive compression to balance accuracy and efficiency.	Extend beyond compression to incorporate neural architecture search, hyperparameter tuning, and optimization strategies. Introduce data augmentation techniques to improve model generalization.
Multi-Agent Reinforcement Learning (MARL) [29]	2021	Unlike PBT, MARL introduces structured communication between agents, enabling them to share information and optimize more efficiently. MARL facilitates collaborative exploration of both hyperparameters and architecture search, preventing redundant exploration through reward shaping.	Further improvements could include enhanced reward shaping techniques and better handling of noisy environments. Exploring advanced collaboration mechanisms could increase efficiency.

Table 1 Chronological evolution of optimization techniques for deep learning.

approach used reinforcement learning to discover combinations of augmentation that improve generalization but was computationally expensive and time-intensive.

This limitation led to a cascade of research aimed at improving efficiency while maintaining performance. Ho et al. [30] proposed population-based augmentation, adapting the PBT framework specifically for data augmentation. Concurrently, Lim et al. [31] developed Fast AutoAugment, employing a tree-structured Parzen estimator to accelerate policy discovery and significantly reduce search time.

The progression continued with Hataya et al.'s Faster AutoAugment [32] in 2020, which implemented gradient approximation techniques to further accelerate the search process. Li et al. [33] introduced Differentiable Automatic Data Augmentation, transforming discrete policy selection into a Differentiable Optimization Problem using Gumbel-Softmax.

A notable trend toward simplification emerged with Cubuk et al.'s RandAugment [34], which drastically reduced the search space to just two parameters: type of operation and its magnitude. This simplified approach offered a more controlled and efficient search space with fewer parameters. This trend continued with approaches like UniformAugment [35], TrivialAugment [36], and AugMix [37], which randomly select parameters from the search space, circumventing explicit search processes.

The data augmentation evolution culminated with Zheng et al.'s Deep AutoAugment [38] in 2022, which extended AutoAugment by increasing the number of operations per sub-policy and formulating the search as a regularized gradient matching problem, expanding the search space while refining the optimization process.

Table 2 presents a chronological overview of this rapid evolution in data augmentation techniques, illustrating how researchers progressively addressed computational inefficiencies while maintaining or improving performance.

Method	Year	Description	Key Features/Improvements
AutoAugment (AA) [27]	2019	Automated data augmentation using reinforcement learning to find optimal policies.	Searches 16 augmentation methods with type/frequency/magnitude parameters. Computationally expensive.
Population-Based Augmentation [30]	2019	Improves AA using population-based training.	Reduces computational cost through efficient population methods.
Fast AutoAugment [31]	2019	Uses tree-structured Parzen estimator for faster search.	Reduces search time by 5x compared to AA through efficient strategy.
Faster AutoAugment [32]	2020	Implements gradient approximation techniques.	Handles non-differentiable operations through differentiable proxy.
Differentiable Augmentation [33]	2020	Differentiable optimization using Gumbel-Softmax.	Enables gradient-based policy search for flexible optimization.
RandAugment [34]	2020	Simplifies search space with two parameters.	Uses operation type and magnitude only for efficient search.
UniformAugment [35]	2020	Random parameter selection from search space.	Avoids explicit search algorithms for simplicity.
Direct Differentiable Search [39]	2021	Gradient-based search without approximations.	Eliminates need for gradient approximation techniques.
TrivialAugment [36]	2021	Minimal-configuration random selection.	Simplifies approach while maintaining performance.
AugMix [37]	2021	Combines multiple augmentations with mixing.	Enhances robustness through diverse mixed operations.
Deep AutoAugment [38]	2022	Extends AA with more operations and regularization.	Expands search space and improves optimization process.

Table 2 Chronological evolution of automatic data augmentation techniques.

3.4 Rise of Multi-Agent Reinforcement Learning for Optimization (2021-2024)

The most recent period has witnessed the emergence of MARL as a powerful paradigm for hyperparameter optimization. [29] work on multiagent reinforcement learning for hyperparameter optimization of convolutional neural networks represented a significant advancement by demonstrating how multiple agents could collaborate specifically for optimizing CNN hyperparameters.

Theoretical foundations were strengthened by [40] work on algorithmic stability and generalization of adaptive optimization methods. The field continued to advance with the exploration by [41] of reinforcement learning strategies from the single-agent to the multiagent domain, highlighting the benefits of collaborative exploration over traditional single-agent approaches.

Recent years have seen a consolidation of knowledge with multiple comprehensive surveys and advancements in MARL. [42] work on team-wise effective communication in multi-agent reinforcement learning emphasizes the critical role of structured communication between agents for more efficient hyperparameter optimization. Unlike earlier approaches like PBT, MARL introduces structured communication between agents, enabling them to share information and optimize more efficiently.

[10] provided a review of research on reinforcement learning algorithms for multi-agents, synthesizing the state of the art in MARL algorithms. [43] offered a broader perspective with their survey on machine learning optimization techniques, classifying approaches and identifying future research directions. [11] survey specifically focused on multi-agent reinforcement learning and its application, emphasizing the growing relevance of MARL across various domains including deep learning optimization.

3.5 Emerging Frameworks and Future Directions

Current deep learning optimization approaches often focus on specific areas like architecture, data augmentation, or hyperparameters, creating a gap in comprehensive frameworks. A promising direction is the development of unified optimization frameworks such as EMERALD-O, which deploys multiple agents specialized in architecture, hyperparameters, and data augmentation within a single framework. Each agent explores independently and collaborates via a shared experience pool for holistic model optimization.

Methods like BOHB and AutoAugment lack structured communication, leading to redundant exploration. Advanced MARL frameworks introduce centralized experience repositories and reward-sharing mechanisms, promoting collaborative learning. This enhances convergence rates and overall efficiency by leveraging collective knowledge [42].

Traditional approaches like BOHB and AutoAugment require extensive computational resources with limited flexibility. More recent MARL-based frameworks incorporate dynamic resource allocation, directing resources to high-performing agents and terminating underperforming configurations early. This ensures efficient resource use and scalability for complex deep learning tasks.

Unlike static strategies in PBT and AutoAugment, adaptive MARL-based approaches integrate online learning, enabling real-time strategy adaptation based on performance feedback. Agents adjust the exploration-exploitation balance dynamically, resulting in faster convergence and better performance in high-dimensional hyperparameter spaces [41].

Existing tools like AutoAugment typically target single objectives. MARL offers a multi-objective framework optimizing architecture, hyperparameters, and data augmentation simultaneously. The collaborative structure manages interdependencies efficiently, making it ideal for large-scale, complex tasks such as image classification [44].

As deep learning models grow increasingly complex and are deployed in more diverse environments, the need for sophisticated optimization techniques that can handle interdependencies between various aspects of model design and training will only increase. The MARL paradigm, with its emphasis on collaboration, communication, and multi-objective optimization, appears particularly well-suited to address these challenges.

4 EMERALD-O Framework

The EMERALD-O framework implements a multi-agent reinforcement learning (MARL) architecture designed for the optimization of deep neural networks through iterative parameter and strategy refinement. The framework initializes an environment E that maintains a state space S defined by a tuple of critical hyperparameters: $\{\eta$ (learning rate), ω (optimizer configuration), α_v (validation accuracy), L_{ma} (moving average loss)$\}$. The system architecture comprises two specialized agents: A_1 and A_2, operating in parallel with distinct action spaces.

A_1 is responsible for data augmentation transformations $T(x) \in \{T_1, ..., T_n\}$, where each T_i represents geometric and photometric transformations on input tensor x. A_2 manages the optimization configuration space O, defined by the Cartesian product of learning rate domain $\eta \in \mathbb{R}^+$ and optimizer type $\omega \in \{\text{Adam}, \text{SGD}, \text{RMSprop}\}$.

Both agents interact with E through a Markov Decision Process (MDP), where at each timestep t, they select actions a_t based on state s_t according to their respective policies $\pi_i(a_t|s_t)$. The environment generates rewards $r_t = f(\alpha_v)$, where f is a reward function mapping validation accuracy to scalar rewards. The agents' neural network policies are updated through gradient ascent on the expected cumulative reward:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum \gamma^t r_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right] \tag{5}$$

where θ represents the policy network parameters and γ is the discount factor.

The policy gradients are computed through backpropagation and used to optimize the agents' decision-making processes via stochastic gradient ascent. Through this iterative optimization process, the agents converge toward optimal policies π_i^* that maximize the expected cumulative reward, resulting in optimized hyperparameter configurations and augmentation strategies for the target deep learning model.

The system employs experience replay with buffer D to store transition tuples (s_t, a_t, r_t, s_{t+1}), enabling offline policy updates and improved sample efficiency. The entire optimization process continues until a convergence criterion based on validation performance is met.

Unlike traditional techniques, such as grid search or Bayesian optimization, which focus primarily on identifying optimal hyperparameter configurations, the EMERALD-O framework offers a more granular and comprehensive exploration of the hyperparameter space. By visualizing the dynamic relationships between hyperparameter values and model performance, EMERALD-O enables the detection of suboptimal regions, uncovers interactions and dependencies between hyperparameters, and highlights sensitivities to specific values.

The quantitative analysis of the model's performance yields critical insights into its behavioral characteristics, enabling the identification of architectural vulnerabilities and suboptimal training protocols through systematic performance metrics. This empirical exploration informs architectural refinements and optimization strategies, facilitating the development of models with enhanced robustness and improved generalization capabilities across diverse input distributions.

The MARL system comprises of two specialized agents working in tandem to fine-tune various aspects of the model training process. Fig 1 presents the entire framework of a multiagent RL based deep learning model optimization framework.

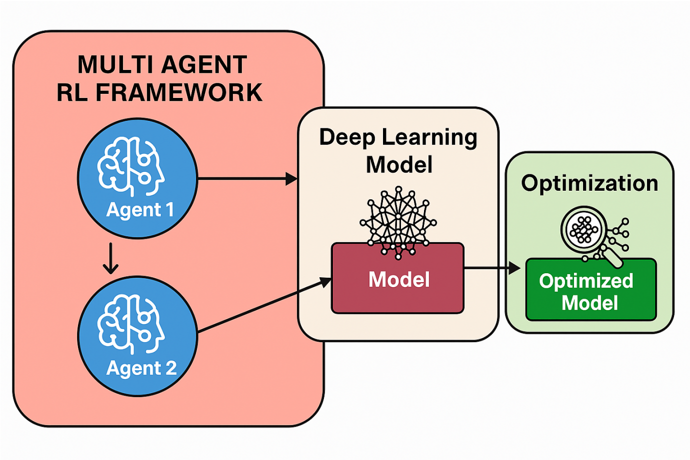


Fig. 1 Multi agent framework for deep learning optimization framework. EMERALD-O comprises of two specialized agents working to fine-tune various aspects of the model training process.

This work focuses on deep learning models for the image modality and for the dataset [45]. In this case, in EMERALD-O, the environment first provides the current state, which includes essential hyperparameters and performance metrics, to both agents. Agent 1 processes this state through its policy network and decides on specific data augmentation techniques to apply. Simultaneously, Agent 2 processes the same state through its policy network and determines the appropriate adjustments to the learning rate and optimizer. The environment then uses these new settings to train the deep learning model and generates a reward based on improvements in validation accuracy or loss. The system calculates the loss by comparing the expected reward with the actual reward, and gradients are subsequently computed to update the policy networks of both agents. This gradient update refines the agents' decision-making capabilities for future iterations, allowing them to optimize the training process more effectively. The detailed EMERALD-O architecture is given in Fig 2.

The entire model is described as state representation (Section 4.1), agents in EMERALD-O and actions (Section 4.2), environment Interaction (Section 4.5) reward mechanism (section 4.6) and learning process (Section 4.7). Further, section 4.3 gives the details about the Policy Network 1, while Section 4.4 provides insights on Policy Network 2.

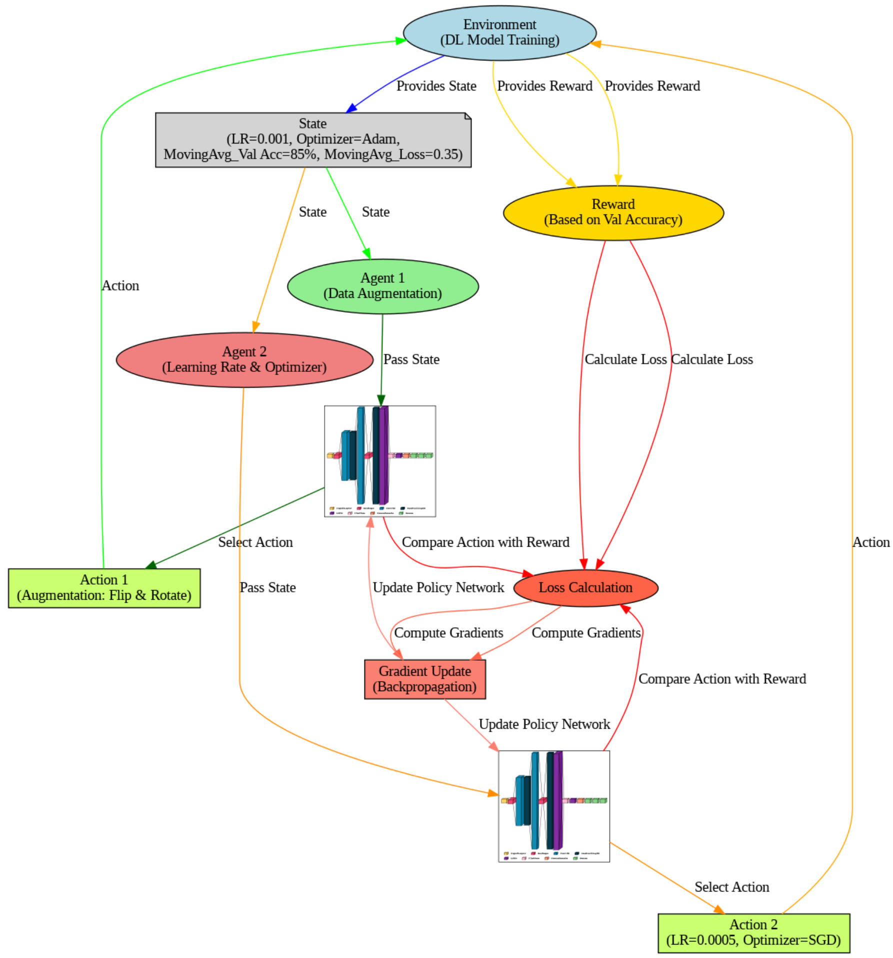


Fig. 2 The figure shows the entire EMERALD-O framework: In this process, the environment first provides the current state, which includes essential hyperparameters and performance metrics, to both agents. Agent 1 processes this state through its policy network and decides on specific data augmentation techniques to apply. Simultaneously, Agent 2 processes the same state through its policy network and determines the appropriate adjustments to the learning rate and optimizer. The environment then uses these new settings to train the EfficientNet model and provides a reward based on improvements in validation accuracy or loss. The system calculates the loss by comparing the expected reward with the actual reward, and gradients are subsequently computed to update the policy networks of both agents. This gradient update refines the agents' decision-making capabilities for future iterations, allowing them to optimize the training process more effectively.

4.1 State Representation

The environment, which encapsulates the deep learning model training process, provides a comprehensive state representation. The state vector includes critical hyperparameters and performance metrics, such as the following: current learning rate, active optimizer configuration, moving average validation accuracy, moving average training loss and data augmentation settings. These components collectively capture the dynamic behavior of the training process, allowing the reinforcement learning agent to monitor and adapt to changes. The state representation evolves over time, reflecting adjustments made during the optimization process. This ensures that the agent can not only track immediate outcomes but also identify long-term trends in model performance. By integrating such elements, the state representation ensures that the agent has sufficient information to make informed decisions for optimization.
For instance, a typical state can be represented as:

```
{learning_rate: 0.001, optimizer: "Adam", Moving_Avg_validation_accuracy:
85%, Moving_Avg_training_loss: 0.35, data_augmentation: {random_flip: True,
rotation_range: 20}}
```

4.2 EMERALD-O Agents

The EMERALD-O framework contains two agents for the task of optimization.

1. **Agent 1**: Specializes in data augmentation strategies.
2. **Agent 2**: Focuses on optimization parameters, including learning rate adjustments and optimizer selection.

Both agents utilize deep neural networks as their policy functions. These policy networks take the current state as input and output action probabilities or direct action values.

In Agent 1's Policy Network (AP_1), the input is the current data augmentation settings along with learning rate and optimizer and their impact on accuracy, and the output is the probabilities for different data augmentation adjustments. The policy network of Agent1 is elaborated in section 4.3. In Agent 2's Policy Network (AP_2), the input is current learning rate, optimizer type, and associated performance metrics and the output is the probabilities for learning rate adjustments and optimizer changes. The Agent1 policy network is described in Section 4.4.

4.2.1 Action Execution

Based on their policy networks' outputs, the agents select and execute actions Agent 1 Action ($A1(Action)$) implements specific data augmentation techniques, such as applying random flips or adjusting rotation ranges. Agent 2 Action ($A2(Action)$) modifies the learning rate (e.g., from 0.001 to 0.0005) or switches the optimizer (e.g., from Adam to SGD).

4.3 Policy Network 1 (Agent 1)

Agent 1 is responsible for adjusting data augmentation techniques in the model training process. The Agent 1 policy network takes the current state of the training process as input and outputs a decision on which data augmentation technique(s) to apply.

4.3.1 Input for Policy Network 1

The input in the agent 1 policy network is a vector representing various aspects of the current state of the training process (Table 3).

Icon	Description	Vector Representation
	Learning Rate Learning Rate: 0.001 Normalized: 0.1	`[0.1]`
	Current Optimizer Optimizer: Adam Encoded as: [1, 0, 0]	`[1, 0, 0]`
	Moving Average Validation Accuracy Validation Accuracy: 85% Normalized: 0.85	`[0.85]`
	Moving Average Training Loss Training Loss: 0.35 Normalized: 0.35	`[0.35]`
	Previous Augmentation Settings Augmentations: No Flip, Rotation, No Zoom, Shift Encoded as: [0, 1, 0, 1]	`[0, 1, 0, 1]`

Table 3 A compact representation of the training state with sample input vector= [0.1, 1, 0, 0, 0.85, 0.35, 0, 1, 0, 1].

4.3.2 Architecture of Policy Network 1 (Agent 1)

The policy network for Agent 1 is an architecture with several layers, each serving a specific purpose. The input layer is designed to receive a 10-dimensional input vector, providing the foundational data for subsequent processing. Following this, Hidden Layer 1 comprises a dense layer with 64 units, utilizing the ReLU activation function to introduce nonlinearity and enhance the model's capacity to capture complex patterns. Subsequently, Hidden Layer 2 also features a dense layer, but with 32 units, continuing to apply the ReLU activation to refine the features learned by the network. Finally, the Output Layer includes a dense layer with 4 units, which employs a softmax activation function. This configuration is specifically tailored to output probabilities corresponding to different augmentation actions, effectively enabling the model to make informed decisions based on the input data.

4.3.3 Output from Policy Network 1

The policy network outputs a probability distribution over a predefined set of data augmentation actions. The vectorized representation for the four principal augmentation techniques — no augmentation, flipping, rotation, and shifting is given in Table 4. For the given input vector, the policy network will give the following output vector = [0.2, 0.5, 0.1, 0.2].

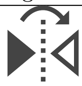
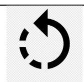
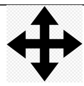
Augmentation Action	Vector Representation
No Augmentation	`[1, 0, 0, 0]`
	`[0, 1, 0, 0]` Apply Flip
	`[0, 0, 1, 0]` Apply Rotation
	`[0, 0, 0, 1]` Apply Shift

Table 4 Pictorial Representation of Augmentation Actions.

This output vector says that the probability is 20% for No-Augmentation, 50% for Apply Flip, 10% for Apply Rotation and 20% for Apply Shift. Thus, based on the output vector, the policy network suggests that the most likely and optimal action is to Apply Flip during data augmentation, given the current state of the training process.

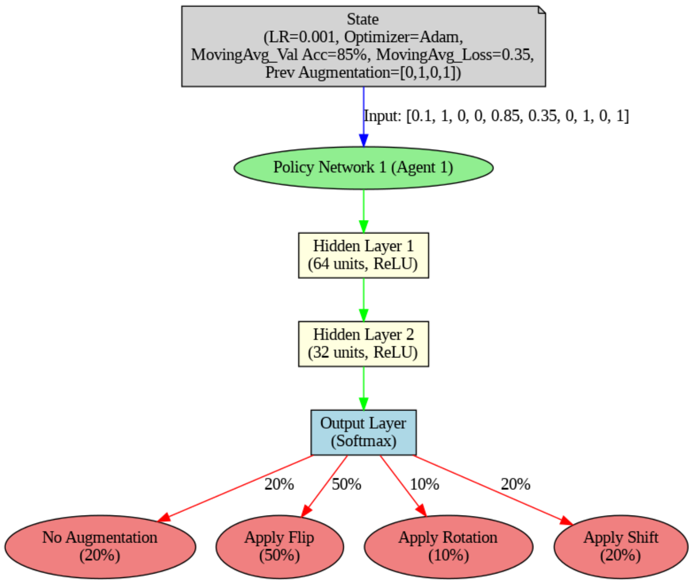


Fig. 3 Policy network for the Agent 1: The input is the current state of the EfficientNet training process, including the learning rate, optimizer, validation accuracy, training loss, and previous augmentation settings.The output is a probability distribution over potential data augmentation actions. The decision of the network suggests applying a flipping augmentation with a probability of 50%, indicating that this action is most likely to improve the model's performance.

Fig 3 shows how Agent 1's policy network considers the current state of training and decides which augmentation technique to apply to potentially improve the model's generalization performance.

4.4 Policy Network 2 (Agent 2)

Agent 2 is responsible for adjusting the learning rate and selecting the optimizer in training the model. The Agent 2 policy network uses the current state of the training process as input and outputs a decision on the learning rate and optimizer to use for the next training iteration.

The input to the policy network for Agent 2 is the vector representing various aspects of the current state of the training process, like Agent 1 but with a focus on hyperparameters like learning rate, optimizer, and performance metrics. The input is the current state of the EfficientNet training process, including the learning rate, optimizer, validation accuracy, and training loss. The output is twofold: a probability distribution over potential optimizers (Adam, SGD, RMSprop) and a recommended learning rate value. Here, the input vector represents the current learning rate 0.001, the current optimizer is Adam, encoded as a categorical variable, for example, [1, 0, 0] for Adam; moving average validation accuracy is 85%. Thus, the input_vector is [0.1, 1, 0, 0, 0.85, 0.35]. The output of the policy network is twofold: Optimizer selection and learning Rate adjustment. Optimizer selection is a probability distribution over the possible optimizers [Adam, SGD, RMSprop] and learning rate adjustment is a scalar value representing the recommended learning rate for the next iteration. The optimizer_output vector is [0.6, 0.3, 0.1] and learning_rate_output is 0.0005. Thus, the network suggests continuing with the Adam optimizer and lowering the learning rate to 0.0005 for the next iteration. This is demonstrated in Fig 4.

4.4.1 Input for Policy Network 2

The input to the policy network for Agent 2 could be a vector representing the various aspects of the current state of the training process. The input vector encapsulates key features of the current training state, including augmentation settings, learning rate adjustments, and recent performance metrics. These components are carefully chosen to provide a comprehensive representation, enabling Agent 2's policy network to make informed decisions during the training process.

Icon	Description	Vector Representation
	Learning Rate Learning Rate: 0.001 Normalized: 0.1	`[0.1]`
	Current Optimizer Optimizer: Adam Encoded as: [1, 0, 0]	`[1, 0, 0]`
	Validation Accuracy Validation Accuracy: 85% Normalized: 0.85	`[0.85]`
	Training Loss Training Loss: 0.35 Normalized: 0.35	`[0.35]`
	Previous Learning Rate and Optimizer Previous Learning Rate: 0.001, Optimizer: Adam Encoded as: [0.1, 1, 0, 0]	`[0.1, 1, 0, 0]`

Table 5 A representation of the input state vector for policy network training.

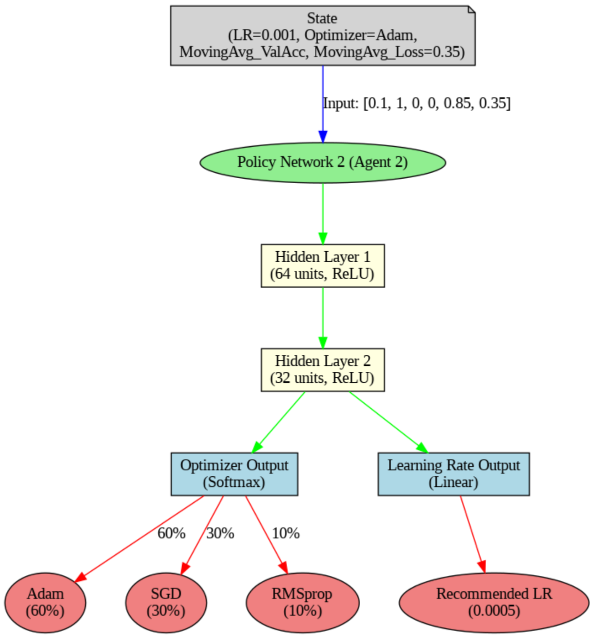


Fig. 4 Policy network for the agent 2: The network's output serves two purposes. First, it produces a probability distribution over the available optimizers, allowing the policy network to select the best optimizer for the next training iteration. Second, it recommends a scalar value for the learning rate, which is then applied in the upcoming iteration. This dual-output structure enables Agent 2 to dynamically adjust both the optimizer and the learning rate based on the training state.

Input vector: = [0.1, 1, 0, 0, 0.85, 0.35, 0.1, 1, 0, 1]
Table 5 gives the details of each of these vector components used in the Policy Network 2.

4.4.2 Architecture of Policy Network 2 (Agent 2)

The policy network for Agent 2 consists of a simple feedforward neural network with two hidden layers. The input layer takes in a 6-dimensional input vector, representing the current state of the training process. The first hidden layer is a dense layer with 64 units, activated by the ReLU function, followed by a second hidden layer with 32 units, also using ReLU activation. The network has two separate outputs: one for selecting the optimizer and the other for determining the learning rate. The optimizer selection output is a dense layer with softmax activation, providing probabilities for three optimizers: Adam, SGD, and RMSprop. The second output is a dense layer with linear activation that adjusts the learning rate.

The network's output serves two purposes. First, it produces a probability distribution over the available optimizers, allowing the policy network to select the best optimizer for the next training iteration. Second, it recommends a scalar value for the learning rate, which is then applied in the upcoming iteration. This dual-output structure enables Agent 2 to dynamically adjust both the optimizer and the learning rate based on the training state.

4.4.3 Output of Policy Network 2

The output of the policy network is twofold, focusing on both optimizer selection and learning rate adjustment. In the optimizer selection, the network generates an output vector that provides probabilities for three potential optimizers: Adam, SGD, and RMSprop. According to the optimizer_output vector, the probability of choosing Adam is 60%, SGD is 30%, and RMSprop is 10%. Based on this, the network suggests that Adam is the most likely and optimal choice for the next iteration, with a 60% probability. SGD remains a secondary option with a 30% likelihood, while RMSprop is less likely, having only a 10% chance.
Output Vector for Optimizer: = [0.6, 0.3, 0.1].

The second part of the output concerns the learning rate adjustment. The policy network recommends lowering the current learning rate from 0.001 to 0.0005 for the next training iteration. This decision is based on the current state of the training process, indicating that reducing the learning rate would improve model performance in subsequent iterations.
Output Value for Learning Rate: = 0.0005.

Based on these outputs, Agent 2 would proceed by using the Adam optimizer and adjusting the learning rate to 0.0005, thus fine-tuning the training process of the model for better performance. Fig 4 illustrates the structure of the policy network for Agent 2. This shows how Agent 2 processes the input state information to decide on the optimal learning rate and optimizer settings, which are critical for the fine-tuning of the EfficientNet model. The input to the policy network for Agent 2 is a vector that encapsulates various aspects of the current training state, with a focus on hyperparameters like the learning rate, optimizer, and key performance metrics, akin to Agent 1 but specifically aimed at hyperparameter adjustments.

4.5 Environment Interaction

The environment then trains the deep learning model using the newly adjusted hyperparameters and data augmentation settings. After a predetermined number of epochs or iterations, it evaluates the model's performance.

4.6 Reward Mechanism

The environment calculates rewards based on improvements in key performance indicators. For Agent 1, rewards are proportional to improvements in validation accuracy resulting from data augmentation changes. Meanwhile, for Agent 2, rewards reflect enhancements in training convergence and overall model performance due to optimization parameter adjustments.

4.6.1 Reward Shaping

Penalties for Low Validation Accuracy and Instability.

The reward function was systematically enhanced to impose stringent penalties for low validation accuracy and instability. By integrating gradient norm-based penalties, the agents were guided away from hyperparameters that induced instability during the training process. This approach effectively steered the optimization towards more stable and high-performing configurations.

Combining Rewards for Enhanced Agent Cooperation.

To foster a higher degree of cooperation between agents, various reward combination strategies were explored. Among these, the weighted sum approach was particularly effective, defined mathematically as:

$$\text{Combined\_reward} = w_1 \cdot \text{reward}_1 + w_2 \cdot \text{reward}_2 \tag{6}$$

where w_1 and w_2 are empirically determined weights. This approach provided a balanced reflection of each agent's contribution to the overall system performance.

4.7 Learning and Update Process

1. **Loss Calculation**: The system computes the loss by comparing expected improvements against actual performance changes.
2. **Gradient Computation**: Gradients are derived from the calculated loss, indicating how to adjust the policy networks for improved decision-making.
3. **Policy Network Update**: Both agents' policy networks are updated using the computed gradients, enhancing their ability to make effective hyperparameter tuning decisions in subsequent iterations.

This MARL approach enables a dynamic and intelligent hyperparameter optimization process. The system can adapt to the unique characteristics of the model and dataset by leveraging specialized agents and continuous learning, potentially leading to superior performance compared to traditional hyperparameter tuning methods.

4.8 Proof of Concept

The environment provides the current state, which includes the learning rate, optimizer, accuracy, loss, and previous augmentations. Based on this state, Agent 1 decides to apply both flipping and rotation to the training data. This augmentation increases the complexity of the data, which prompts Agent 2 to respond by lowering the learning rate and switching to the SGD optimizer to better handle the newly augmented data and maintain a stable training process.

4.8.1 Output After Agents' Decisions

The final state consists of the following configuration: data augmentation includes both flipping and rotation, the optimizer has been switched to SGD, and the learning rate has been reduced to 0.0005. As a result of these adjustments, the validation accuracy is expected to change based on the new training configuration, and the training loss is anticipated to evolve in response to the modified learning rate and optimizer.
Fig 5 and 6 illustrate how the actions of one agent can impact the decisions of another, highlighting the critical role of coordination in a multi-agent system.

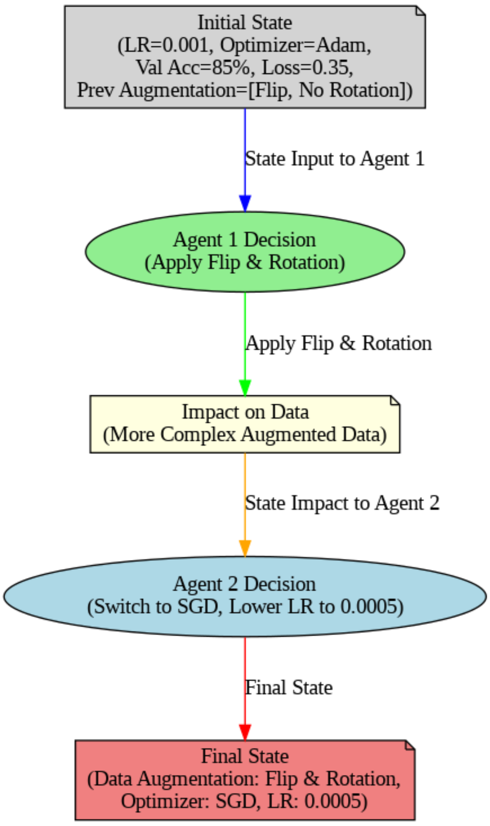


Fig. 5 The environment provides the current state (learning rate, optimizer, accuracy, loss, previous augmentations).Based on the state, Agent 1 decides to apply both flipping and rotation to the training data.In response to the increased complexity from the augmented data, Agent 2 lowers the learning rate and switches to SGD.

In Fig 6, the environment is a key component that interacts with the agents by providing states, receiving actions, and returning rewards and new states. A detailed example that includes a sample input to the environment, what the environment does with this input, and the resulting output follows. The input to the environment consists of the actions chosen by Agent 1 and Agent 2, as well as the current state of the model's training. The environment takes the actions provided by the agents and applies them to the training process of the deep learning model. Then, the environment returns the new state and rewards based on the updated performance of the deep learning model. Reward 1 is based on the improvement in validation accuracy due to the augmented data. Reward 2 is based on the reduction in training loss due to better hyperparameter settings.

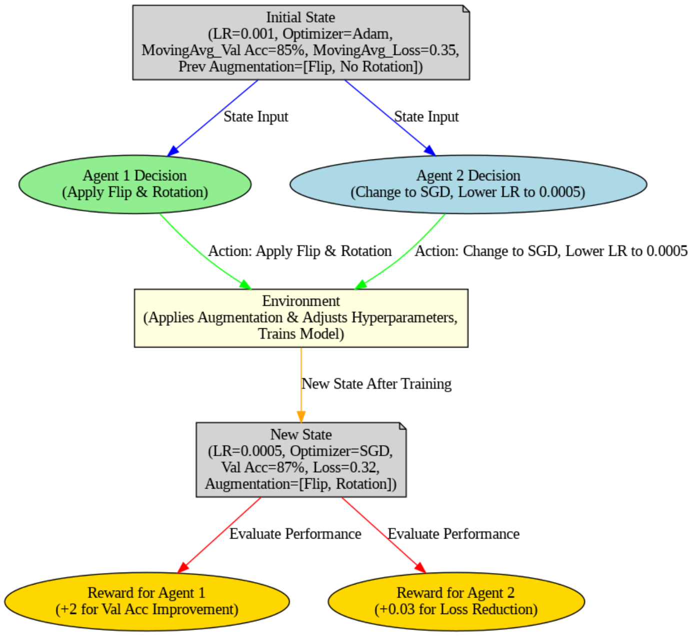


Fig. 6 Interaction between environment, agent1 and agent2: The environment receives the actions from Agent 1 and Agent 2, which include data augmentation techniques and adjustments to the optimizer and learning rate. The environment applies the augmentations to the data and uses the new hyperparameters to train the model for another epoch. The new state vector: [0.0005, SGD, 0.87, 0.32, 1, 1], where 0.0005 is the New Learning Rate (as set by Agent 2), New Optimizer is SGD (as set by Agent 2), New Validation Accuracy: 87% (improved due to better generalization from augmentation and proper learning rate), New Training Loss is 0.32 (reduced, indicating better training with the new settings), Applied Data Augmentation: Flip, Rotation. Reward for agent 1 is calculated as the difference in validation accuracy: 87% - 85% = 2% improvement. Reward for agent2 is calculated as the difference in training loss: 0.35 - 0.32 = 0.03 reduction.

5 Experimental Setup

The experimentation in this EMERALD-O framework was conduscted in three folds: in the first (Section 5.1), the architectural adjustment is studied by adjusting the exploration-exploitation balance, checking the reward shaping parameters and the stopping criteria for the architecture. In the second (Section 5.2), the generalizability of the framework is studied by experimenting on the other prominent deep learning models. The third (Section 5.3) is for checking the performance of the framework against existing optimization tools. This ensures the feasibility of the EMERALD-O framework for the task of optimization.

5.1 Experiment 1: Design of MARL Architecture

5.1.1 Adjusting the Exploration-Exploitation Balance

The initial experiment focused on refining the exploration-exploitation balance within the MARL framework. A key adjustment was the acceleration of the decay rate for the exploration parameter, set at 0.01. This modification facilitated a faster transition from exploration to exploitation, enabling agents to converge more efficiently on optimal hyperparameters as training progressed. This strategic adjustment resulted in a more focused and effective search across the hyperparameter space, leading to significant improvements in model stability and overall accuracy.

Types of reward function

This work has experimentally searched for the combination of a product-based reward and minimum function based reward, in addition to the combined reward strategy in the EMERALD-O framework.
The product-based combination is defined as :

$$\text{Combined\_reward} = (\text{reward}_1 \cdot \text{reward}_2) \tag{7}$$

The minimum function based reward is defined as :

$$\text{Combined\_reward} = \min(\text{reward}_1, \text{reward}_2) \tag{8}$$

These methods underscored the importance of strong agent coordination, with the weighted sum method yielding the best results. Empirically, the MARL framework achieved a validation accuracy of 88.59% with a learning rate of 0.0001, highlighting the efficacy of the reward combination in enhancing agent cooperation.

5.1.2 Early Stopping and Pruning

Early stopping criteria were introduced to eliminate hyperparameter configurations that consistently underperformed, thereby reducing unnecessary computational expenditure. This approach allowed the agents to concentrate on more promising areas of the hyperparameter space. The integration of early stopping with a hierarchical or meta-agent approach streamlined the optimization process, improving both efficiency and effectiveness.

5.2 Experiment 2: Generalization of MARL with VGG16 and VGG19

To evaluate the generalizability of the MARL architecture, the EfficientNet backbone was substituted with VGG16 and VGG19 models. The framework was tested using grid search and Bayesian optimization techniques. This framework outperformed these traditional optimization techniques for both these deep learning models: VGG16 and VGG19.

5.2.1 Grid Search and Bayesian Optimization on VGG16 and VGG19

The MARL framework was systematically evaluated with VGG16 and VGG19 architectures, using both grid search and Bayesian optimization. The results demonstrated that the MARL framework could effectively generalize across different convolutional neural network (CNN) architectures, maintaining competitive performance in hyperparameter optimization. This generalization capability underscores the robustness and versatility of the MARL architecture in various deep learning contexts.

5.3 Experiment 3: Comparison with Existing Optimization Tools on EfficientNetV2B0

A comprehensive comparative analysis was performed to benchmark the performance of the MARL framework against a suite of state-of-the-art optimization techniques in the EfficientNetV2B0 model. Table 6 presents the detailed mathematical formulations for each optimization method.

Method	**Optimization Strategy**	**Key Equations and Performance**
Optuna	Tree-structured Parzen Estimator (TPE)	$p(\mathbf{x} \mid y) = \begin{cases} l(\mathbf{x}) & \text{if } y < y^* \\ g(\mathbf{x}) & \text{if } y \geq y^* \end{cases}$
HyperOpt	TPE-based optimization minimizing expected loss	$\mathbf{x}^* = \arg\min_{\mathbf{x}} \mathbb{E}_{p(\mathbf{x}\mid y)} [L(\mathbf{x}, y)]$
Scikit-Optimize	Bayesian optimization with decision trees	$y(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$
Ax (Adaptive Experimentation Platform)	Bayesian linear regression with UCB criterion	$y(\mathbf{x}) = \mathbf{x}^\top \beta + \epsilon, \ \epsilon \sim \mathcal{N}(0, \sigma^2)$
Spearmint	Gaussian Processes with PI criterion	$\mathrm{PI}(\mathbf{x}) = \Phi\left(\frac{f(\mathbf{x}) - f_{\text{best}}}{\sigma(\mathbf{x})}\right)$
HyperBand	Bandit-based approach with resource allocation	$n_i = \left\lfloor \frac{B}{R} \cdot \frac{R}{\eta^i} \right\rfloor, \quad r_i = R \cdot \eta^{-i}$
Talos	Grid and random search for Keras models	$\mathbf{x}^* = \arg\min_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x})$
Additional Tools	Random Search, Genetic Algorithms, etc.	$\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_{i=1}^{n} L(\mathbf{x}, y_i)$

Table 6 Comparison of hyperparameter optimization techniques and their key equations.

Experimentation was conducted against all these methods and is discussed in detail in the next section.

6 Results and Discussion

This section presents the results and analysis for the EMERALD-O framework based on the above three-fold experimental setup. The comprehensive evaluation of these optimization techniques highlights the distinct advantages of the MARL framework. MARL's multi-agent architecture, dynamic exploration-exploitation strategies, and sophisticated reward mechanisms enable it to navigate complex hyperparameter spaces more effectively than traditional optimization tools. The inherent limitations of single-agent approaches and static strategies observed in other methods underscore the superiority of MARL in delivering state-of-the-art performance in hyperparameter optimization. The results demonstrated in this section show that MARL not only excels in optimizing EfficientNetV2B0 but also generalizes effectively to other architectures, making it a robust and flexible tool for deep learning model optimization.

6.1 Results of Experiment 1: MARL Architecture Design

The EMERALD-O framework demonstrated significant improvements in model performance due to the refined exploration-exploitation balance, enhanced reward shaping, and effective reward combination strategies. The weighted sum approach for reward combination yielded the highest validation accuracy of 88.59%. The empirical results are summarized in Table 7.

Reward Strategy	Accuracy (%)
Weighted Sum	88.59
Product-Based	85.72
Minimum Function	82.34

Table 7 Performance of MARL architecture with various reward combination strategies.

Fig. 7 explains the changes in validation accuracy over time (epochs) in the context of a multi-step, episodic training process. In this model, each episode consists of 50 steps, and each step contains 50 epochs, resulting in a total of 2500 iterations of training. The X-axis represents the epochs, progressing through the entire training process, while the Y-axis reflects the validation accuracy, ranging from 0.4 to 0.9. The graph reveals a recurring pattern of accuracy spikes and drops, suggesting the periodic adjustments occurring after each episode or step. As the model progresses, sharp fluctuations are consistently observed, aligning with the transitions between steps, which mark moments when the model may be resetting certain parameters, adjusting the learning rate, or testing new exploration strategies.

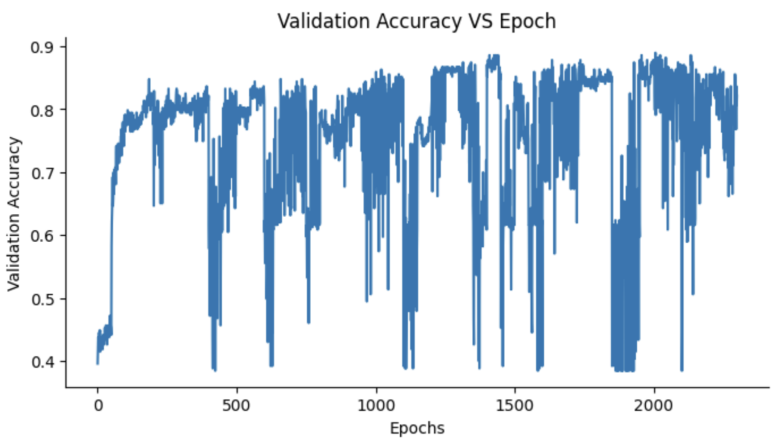


Fig. 7 Validation accuracy vs. epochs: The plot illustrates the fluctuations in validation accuracy across epochs during the training of the EMERALD-O model. The variations highlight the dynamic behavior of the model's optimization process, showcasing the influence of adaptive learning rates and hyperparameter adjustments. The spikes and drops in accuracy reflect the exploration and exploitation phases inherent in the EMERALD-O training framework.

During the initial training phase (0–500 epochs), the model exhibits a sharp increase in accuracy, quickly stabilizing above 0.8. However, as training continues, more significant accuracy drops emerge, likely caused by the model's efforts to explore new strategies and fine-tune parameters at the start of each new step or episode. These drops become more pronounced between 500 and 1500 epochs, indicating that the model is entering a deeper exploration phase. Despite these drops, the model generally recovers well, maintaining accuracy between 0.7 and 0.9 throughout most of the training process. Iterations prior to the 1300 epoch mostly involve exploration-heavy phases where suboptimal policies were tested, resulting in more fluctuation. By 1300 epochs, the exploration-exploitation trade-off shifts towards exploitation of successful strategies, stabilizing the performance. Toward the later stages (1500–2500 epochs), the fluctuations become sharper, but the model's ability to recover indicates that it continues learning effectively. Overall, this graph reflects the typical behavior of an episodic training process within a reinforcement learning context, where accuracy fluctuations are expected due to the cyclic nature of exploration, parameter adjustments, and recovery phases that occur between episodes.

The agents' evaluation of the environment is conducted across various criteria, including optimizer selection, learning rate, augmentation, and preprocessing techniques, to validate the effectiveness of the MARL-based approach in identifying optimal hyperparameters for deep learning models.

Fig 8 presents the progression of validation accuracy over time (in epochs) for three optimizers: Adam, RMSprop, and SGD. The X-axis represents the number of epochs, and the Y-axis displays the validation accuracy, ranging from 0.4 to 0.9. Adam, shown by the green line, initially demonstrates high validation accuracy but experiences significant fluctuations throughout the training process. Periodic sharp drops in accuracy, especially beyond 500 epochs, suggest sensitivity to episodic adjustments, i.e., learning rate changes. Despite these drops, Adam consistently recovers, stabilizing around 0.7 to 0.8, showing its ability to learn and adapt. RMSprop, indicated by the orange line, exhibits similar fluctuations, with sharper and more prolonged dips, particularly between 500–1000 and 1500–2000 epochs. This optimizer appears less stable, struggling with prolonged recovery after accuracy drops, likely due to higher sensitivity to learning rate or exploration-exploitation imbalances.

In contrast, the purple line representing SGD shows a smoother and more stable performance. It steadily increases validation accuracy, reaching around 0.8 early on and maintaining stability with fewer and less severe dips compared to Adam and RMSprop. Although SGD exhibits minor fluctuations, its overall consistency and quicker recovery after small dips indicate better handling of the episodic training structure. While Adam explores parameter spaces effectively, it is prone to instability, and RMSprop displays the most volatile behavior with slower recovery from accuracy drops. SGD, however, proves to be the most stable optimizer, performing more consistently with fewer drastic fluctuations, making it a suitable choice for long training processes involving episodic adjustments. [40] explores the influence of SGD. Thus, Fig 8 presents the differences in optimizer performance under episodic training, with SGD offering greater stability, Adam performing well but with occasional instability, and RMSprop being the least stable of the three.

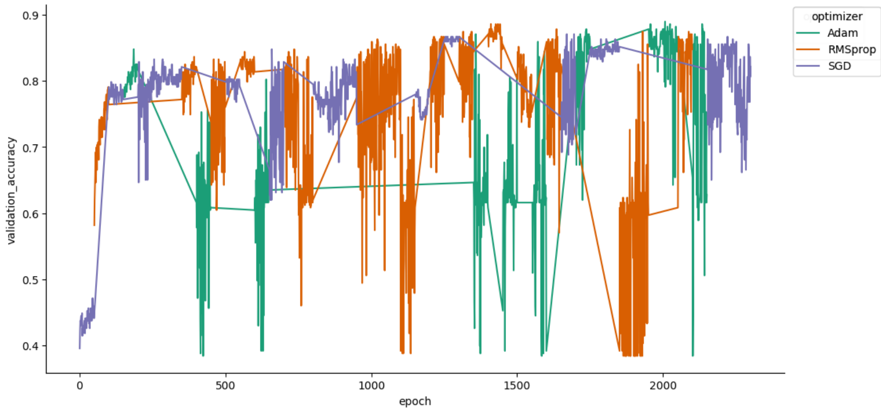


Fig. 8 Validation accuracy vs. optimizer: The figure shows the validation accuracy trends across different epochs for the EMERALD-O model, using three distinct optimizers: Adam, RMSprop, and SGD. The results highlight the varying impacts of these optimization techniques on the model's performance during training.

Fig 9 shows the progression of validation accuracy over epochs for three distinct learning rates: 1e-05, 0.0001, and 0.001. The X-axis captures the epochs, while the Y-axis tracks the validation accuracy, ranging between 0.4 and 0.9. Each line represents the performance for a specific learning rate: green for 1e-05, orange for 0.0001, and purple for 0.001. This comparison allows an assessment of how varying learning rates influence the stability and generalization capabilities of the model over time.

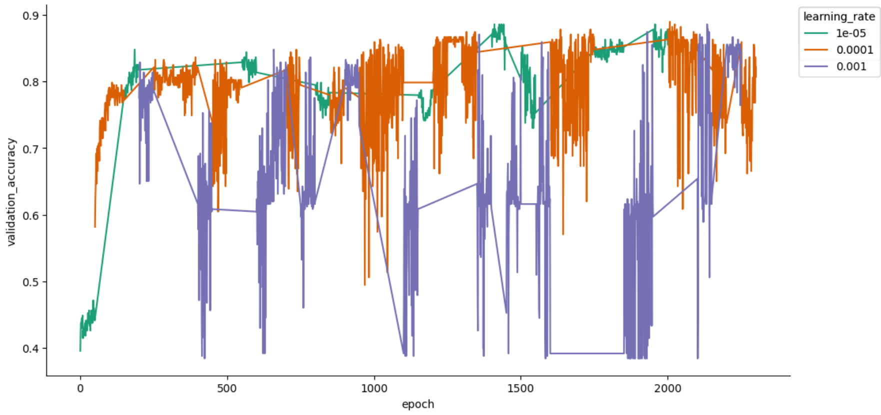


Fig. 9 This figure depicts the validation accuracy trends across epochs for the EMERALD-O model using different learning rates: 10^{-5}, 10^{-4}, and 10^{-3}. The results highlight the sensitivity of the EMERALD-O framework to learning rate variations, with distinct patterns emerging for each rate.

In terms of performance, the green line (learning rate 1e-05) demonstrates consistent and stable validation accuracy, maintaining an accuracy level near 0.8 with minimal fluctuations, indicating smooth convergence and strong generalization. The orange line (learning rate 0.0001) displays a higher degree of variability, achieving early peaks of 0.8 but frequently experiencing sharp drops in validation accuracy, reflecting periods of instability. However, the model manages to recover after these dips, suggesting that this learning rate allows for effective exploration, albeit at the cost of some volatility. The purple line (learning rate 0.001) exhibits the most volatile behavior, with frequent, sharp decreases in validation accuracy, often dipping below 0.6. This larger learning rate results in significant updates to the model's parameters, potentially causing overfitting or overcorrection, leading to substantial instability in validation performance.

The observed fluctuations in validation accuracy across all learning rates may correspond to the episodic structure of training, where each episode consists of 50 steps, with each step representing 50 epochs. The periodic declines in accuracy, particularly visible in the orange and purple lines, suggest the model is adjusting parameters or exploring new strategies at the start of each new episode.

Thus, smaller learning rates (such as 1e-05) provide the most stable performance with fewer fluctuations, allowing for smoother convergence and better generalization. Moderate learning rates (such as 0.0001) offer a balance between exploration and stability, with periodic drops but effective recovery. In contrast, higher learning rates (like 0.001) induce greater instability and frequent accuracy dips, indicating the need for smaller updates to achieve sustained performance improvements.

Fig 10 shows the validation accuracy over the number of epochs for two different augmentation strategies: horizontal flip (True) and horizontal flip (False). The X-axis denotes the number of epochs, while the Y-axis measures validation accuracy, ranging from 0.4 to 0.9. Both augmentation techniques exhibit variability in accuracy, with each following a distinct pattern of progression. The green line, corresponding to horizontal flip (False), demonstrates a more stable trajectory throughout training, maintaining validation accuracy in the range of 0.7 to 0.8 with fewer sharp drops. Meanwhile, the orange line, representing horizontal flip (True), experiences significant fluctuations, particularly beyond 500 epochs, suggesting that this augmentation introduces instability in the model's learning process.

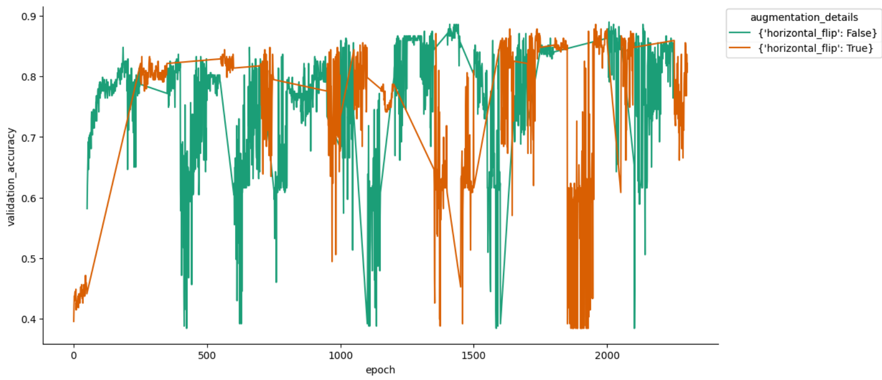


Fig. 10 Validation accuracy vs. augmentation: The two configurations, with and without horizontal flipping, demonstrate the influence of this augmentation technique on EMERALD-O's performance.

Further, the episodic training structure, consisting of 50 steps per episode and 50 epochs per step, is reflected in the regular dips observed across both augmentation settings. These periodic drops indicates the model's adjustment at the beginning of new episodes, potentially resetting parameters or exploring new learning strategies. Horizontal flip (True) consistently shows greater instability in validation accuracy, pointing to its limited contribution to the task, while horizontal flip (False) delivers a more consistent generalization

Fig 11 gives a comprehensive view of validation accuracy over a range of epochs for various rotation range preprocessing settings applied during training. The X-axis represents the number of epochs, while the Y-axis captures validation accuracy, spanning values from 0.4 to 0.9. Each line corresponds to a different rotation range value: 0 (green), 10 (orange), 20 (purple), and 30 (pink). The graph offers insight into how these preprocessing techniques influence the model's performance throughout training. The green line (rotation range 0) exhibits the most stable behavior, consistently maintaining validation accuracy between 0.7 and 0.8 with occasional but manageable dips. Conversely, the orange line (rotation range 10) demonstrates a significant decline early in the training process, struggling to recover and generally performing below 0.5, suggesting that this preprocessing step introduces excessive distortion, hindering the model's ability to generalize.

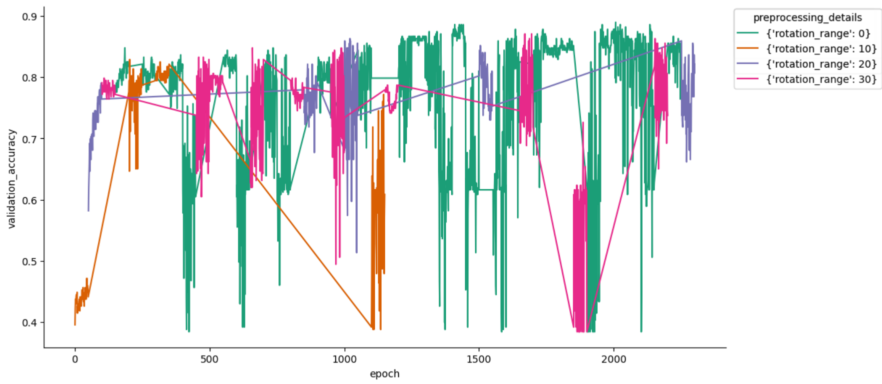


Fig. 11 Validation accuracy vs. preprocessing details: This figure showcases the validation accuracy trends across epochs for the EMERALD-O model under varying rotation ranges during preprocessing (rotation_range: 0, rotation_range: 10, rotation_range: 20, and rotation_range: 30). The results highlight the sensitivity of the model to different levels of rotational augmentation.

The purple and pink lines (rotation ranges 20 and 30, respectively) exhibit similar initial performance, reaching validation accuracies close to 0.8 in the early stages. However, these models encounter sharp dips in accuracy, particularly in the later epochs, indicating instability in their learning processes. The periodic fluctuations across all lines stem from the episodic training structure, with 50 steps per episode and 50 epochs per step. These regular dips may represent the model's adjustment phases between episodes. Overall, the findings indicate that rotation range (0) leads to the best stability and generalization, while higher rotation values (20 and 30) introduce greater instability. Rotation range (10) consistently underperforms, suggesting its limited utility for this specific classification task. Thus, reducing or eliminating rotation range preprocessing may enhance the model's stability and overall performance.

Fig 12 presents a 3D scatter plot that shows the interplay between augmentation techniques (horizontal flip: True/False), preprocessing methods (rotation ranges of 10, 20, 30), and their impact on validation accuracy under the Adam optimizer. The color bar on the right depicts validation accuracy, with values closer to yellow (0.8 approximately) indicating higher performance, and purple (0.4 approximately) signifying lower performance. The X-axis represents whether horizontal flip was applied, the Y-axis illustrates the rotation range used during preprocessing, and the Z-axis measures the model's validation accuracy. This graphical representation highlights how different augmentation and preprocessing strategies influence model performance during training.

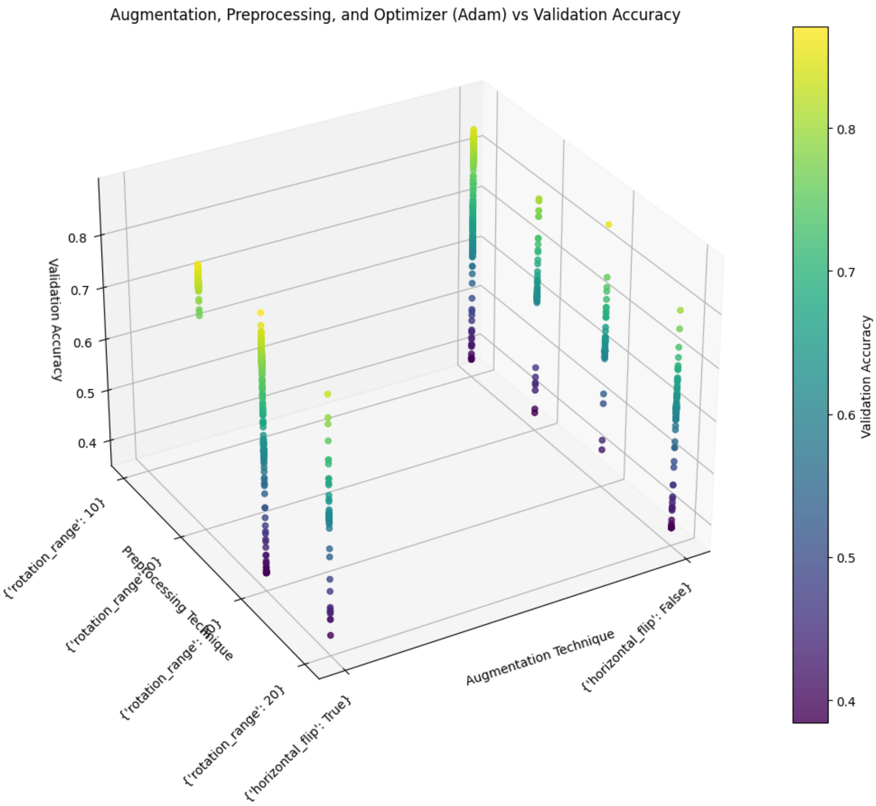


Fig. 12 Validation accuracy vs. augmentation and preprocessing details: This plot illustrates the combined impact of augmentation techniques, preprocessing strategies , and the Adam optimizer on validation accuracy for the EMERALD-O model. Each point represents a specific combination of augmentation and preprocessing parameters, with color indicating the corresponding validation accuracy.

Key observations reveal that a rotation range of 10 (left side) produces the highest validation accuracy, suggesting that a moderate rotation offers the most generalization capability for the model. In contrast, rotation ranges of 20 and 30 display greater variability in validation accuracy, with rotation range 30 showing a noticeable decline in performance. The use of a horizontal flip further compounds this effect. When the flip is set to True, validation accuracy tends to be lower, particularly when combined with higher rotation ranges, indicating potential model confusion or overfitting. Conversely, models without horizontal flip consistently yield higher accuracy, particularly in conjunction with rotation ranges of 10 and 20, underscoring the importance of balancing augmentation techniques with preprocessing methods for optimal generalization.

These findings align with the model's episodic training structure, where each episode comprises 50 steps, each consisting of 50 epochs. Fluctuations in validation accuracy correspond to episodic transitions, during which the model likely undergoes parameter resets or adjustments. The plot highlights how these augmentation and preprocessing combinations impact the model's ability to recover and stabilize during such transitions. Optimal configurations, such as rotation range 10 with horizontal flip False, yield the most stable performance, while higher rotation ranges introduce instability, particularly when combined with horizontal flip.

Fig 13 presents a 3D scatter plot that illustrates the inter-relationship between optimizer choice, learning rate, and validation accuracy in the training process. The X-axis represents the optimizer types (Adam, SGD, and RMSprop), while the Y-axis shows the range of learning rates (from 1e-05 to 0.001). The Z-axis captures validation accuracy, providing a measure of the model's generalization performance. Color-coding is used to represent validation accuracy, with yellow indicating higher accuracy (near 0.8) and purple indicating lower accuracy (around 0.4). This visualization offers insight into how different combinations of optimizer and learning rate impact model performance.

The inter-relationship between optimizer and learning rate is clearly evident in the plot. Adam and SGD show strong performance at lower and moderate learning rates (such as 1e-05 and 0.0001), consistently achieving higher validation accuracies. However, Adam is more sensitive to learning rate variations, resulting in a wider range of performance outcomes, while SGD demonstrates more stability, making it less prone to drastic accuracy drops. RMSprop, meanwhile, is highly sensitive to the learning rate, with a significant spread of points indicating fluctuating performance, especially at higher learning rates. This suggests that while Adam and SGD are more resilient to learning rate changes, RMSprop requires more careful tuning of the learning rate to avoid performance instability. Overall, the combination of a moderate learning rate with Adam or SGD produces the most reliable and stable validation accuracy, highlighting the importance of optimizing learning rate in conjunction with the chosen optimizer for effective training and generalization.

The pair plot (Fig 14) visualizes the relationships between key variables influencing validation accuracy, including optimizer, learning rate, augmentation details, and preprocessing details. The diagonal elements represent kernel density estimates for each variable, while the off-diagonal scatter plots illustrate their pairwise interactions. The

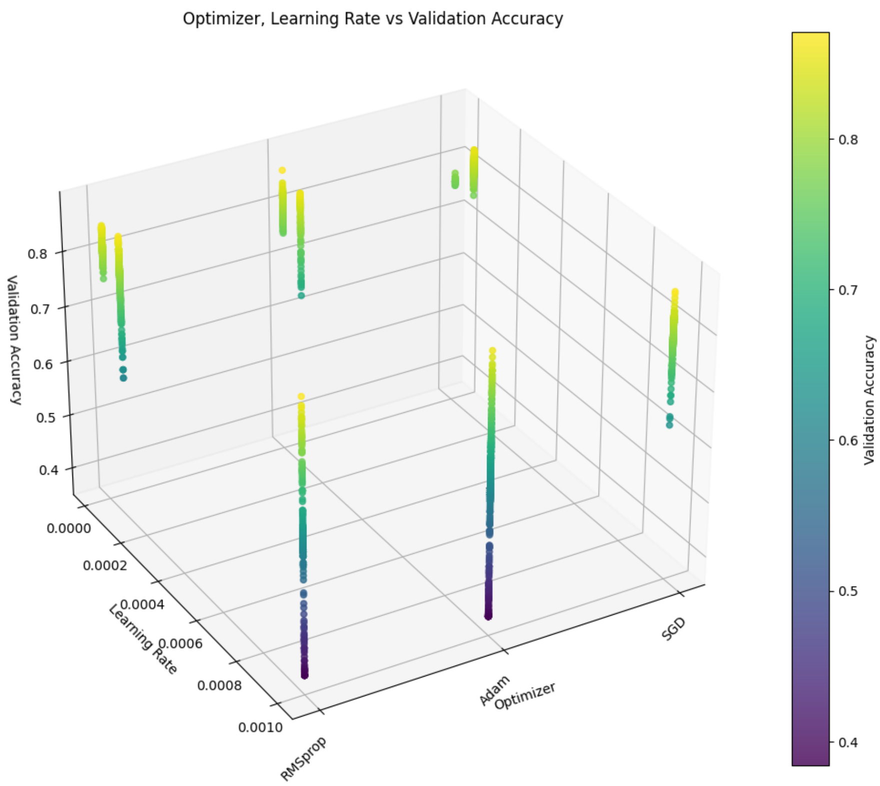


Fig. 13 Validation accuracy vs optimizer and learning rate: This shows the interaction between optimizers (RMSprop, Adam, SGD), learning rates, and their impact on validation accuracy for the EMERALD-O model. Each point represents a unique combination of optimizer and learning rate, with color indicating the corresponding validation accuracy.

plots reveal distinct trends in how these variables impact validation accuracy, providing insights into the model's generalization performance under different configurations.

The optimizer versus validation accuracy plot shows that all optimizers, including SGD, Adam, and RMSprop, achieve comparable accuracy ranges between 0.6 and 0.8, although some optimizers demonstrate greater consistency. The learning rate shows more variability in its impact; smaller learning rates (e.g., 1e-05) are associated with more stable and higher validation accuracy, while higher learning rates (e.g., 0.001) lead to more significant fluctuations, likely due to overfitting or unstable training behavior. Augmentation techniques, meanwhile, have a relatively neutral impact, with validation accuracy consistently clustering around 0.6 to 0.8, suggesting limited direct influence on performance. In contrast, preprocessing techniques show a more pronounced effect, with certain methods consistently producing higher validation accuracy, as seen in the clustering of points around 0.8. This highlights the importance of careful preprocessing design in optimizing model performance.

The interrelationship between optimizer, learning rate, augmentation, and preprocessing suggests that learning rate and preprocessing techniques are the primary

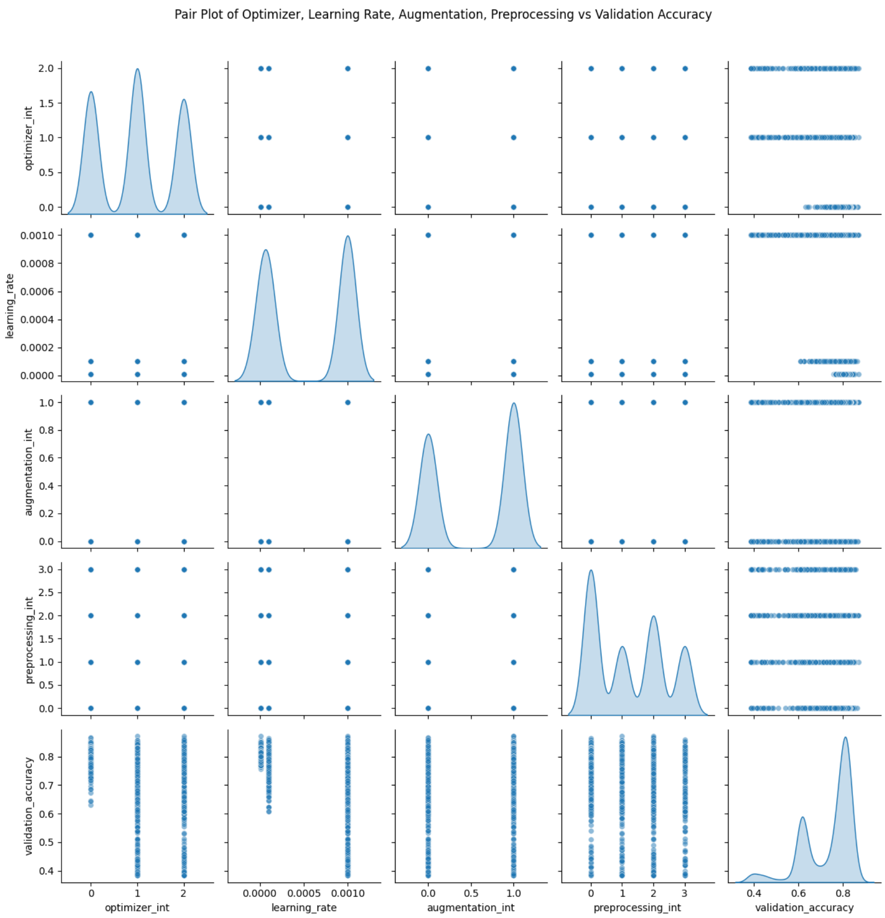


Fig. 14 Pair plot of optimizer,learning rate, augmentation and preprocessing vs. validation accuracy.

factors influencing validation accuracy. Smaller learning rates result in better generalization across all optimizers, while preprocessing significantly impacts performance consistency. Although augmentation has a more subtle influence, its combination with specific preprocessing techniques could indirectly affect the model's ability to generalize. Therefore, optimizing learning rate and preprocessing is essential for achieving consistent and high validation accuracy across different training configurations.

6.2 Results of Experiment 2: Generalization with other Deep Learning Models

The MARL framework was tested for its ability to generalize across different CNN architectures by replacing EfficientNet with VGG16 and VGG19. The results indicated that MARL maintained competitive performance across these models, demonstrating its robustness. Table 8 presents the validation accuracy for both models using grid search and Bayesian optimization.

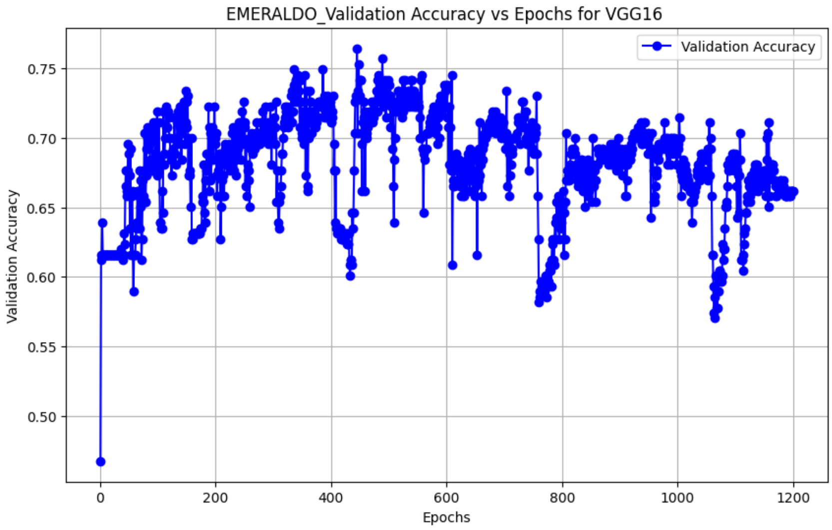


Fig. 15 EMERALD-O VGG16 validation accuracy vs. epoch: The figure depicts the validation accuracy progression across epochs for the EMERALD-O framework using the VGG16 model. The curve demonstrates a gradual improvement in validation accuracy as the training progresses, with fluctuations reflecting the optimization dynamics.

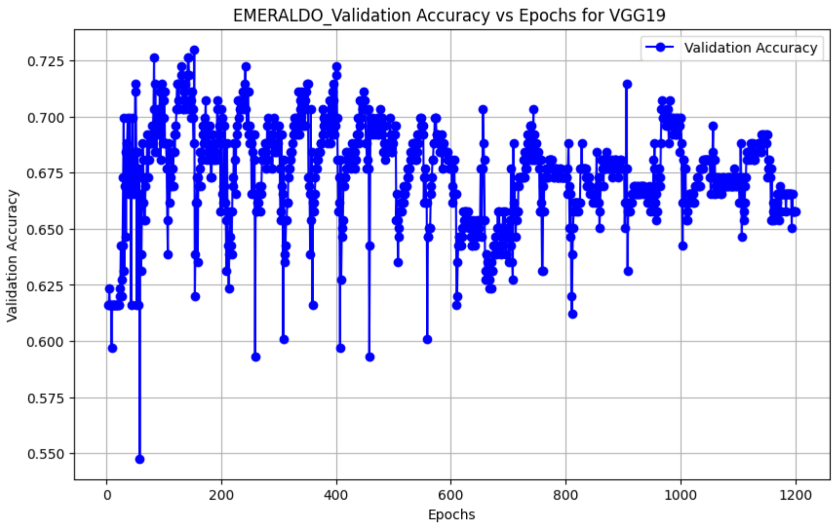


Fig. 16 EMERALD-O VGG19 validation accuracy vs. epoch: This illustrates the validation accuracy progression across epochs for the EMERALD-O framework using the VGG19 model. The curve showcases an overall improvement in validation accuracy, interspersed with fluctuations that reflect the optimization dynamics of the model.

Model	Grid Search (%)	Bayesian Optimization (%)
VGG16	64.12	66.47
VGG19	60.00	64.23

Table 8 Validation accuracy of VGG16 and VGG19 using Grid Search and Bayesian Optimization.

Fig 15 and Fig 16 illustrate that the MARL framework achieved consistent accuracy across different models, reinforcing its adaptability and effectiveness. The 3D scatter plot (Fig 17) shows the validation accuracy of the VGG19 model as a function of augmentation techniques, preprocessing techniques, learning rates, and optimizers. In this experiment, the highest validation accuracy of 0.7300 was achieved using the Adam optimizer, with horizontal flip augmentation applied and a rotation range of 0. The learning rate was approximately 0.0010, which proved to be optimal for this configuration. No data points were observed for `horizontal_flip: False`, indicating that the experiment consistently applied flip augmentation throughout.

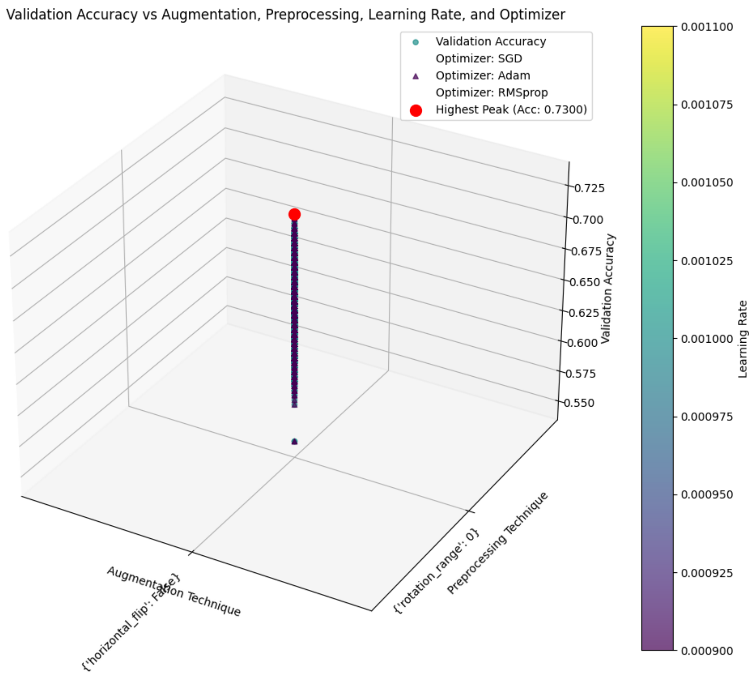


Fig. 17 Validation Accuracy of VGG19 for Optimizer, Augmentation Details, Preprocessing details and Learning Rate: plot demonstrates the relationship between validation accuracy, augmentation techniques, preprocessing strategies, learning rates, and optimizers for the EMERALD-O framework using the VGG19 model. Each point represents a specific combination of augmentation (horizontal flip), preprocessing (rotation range), optimizer (SGD, Adam, RMSprop), and learning rate, with color indicating the learning rate and the highest peak (highlighted in red) achieving a validation accuracy of 73%.

The preprocessing technique was limited to a rotation range of 0, meaning no rotation augmentation was applied. This suggests that minimal transformations in the form of no rotation were beneficial for the model's performance, as evidenced by the high validation accuracy. The learning rates varied slightly between 0.0009 and 0.0011, with higher learning rates leading to more consistent validation accuracy in the 0.65-0.73 range, while lower learning rates resulted in validation accuracies closer to 0.55-0.60.

Overall, the combination of the Adam optimizer, horizontal flip augmentation, and no rotation preprocessing with a moderate learning rate of 0.001 proved to be the most effective setup in maximizing validation accuracy for the VGG19 model in this configuration. Further experimentation with different preprocessing techniques, such as higher rotation ranges or using other optimizers like SGD and RMSprop, may reveal additional insights into improving the model's performance.

Fig 18 presents the pair plot, that offers a comprehensive comparison of the optimizer, learning rate, augmentation, preprocessing, and validation accuracy for the VGG19 model in a MARL framework. The off-diagonal scatter plots illustrate the relationships between these variables, while the diagonal elements represent kernel density estimations (KDE), highlighting the distribution of individual parameters. A notable observation is the validation accuracy clustering between 0.6 and 0.8, with optimizers like Adam and RMSprop showing higher density around the 0.7 accuracy mark, while SGD displays more variability across lower accuracy values. Furthermore, the influence of the learning rate is pronounced - learning rates in the range of 1e-05 to 1e-04 exhibit better performance, achieving consistently higher validation accuracy, whereas learning rates outside this range introduce variability and often result in lower accuracy.

The interaction between preprocessing and augmentation strategies further highlights important trends in model performance. Augmentation, such as horizontal flip: True, provides slight improvements in validation accuracy, suggesting a modest contribution to generalization. Meanwhile, preprocessing, particularly rotation range, significantly impacts validation accuracy. Models subjected to higher rotation ranges (e.g., rotation range: 30) demonstrate reduced accuracy, indicating that aggressive preprocessing techniques may introduce instability. In contrast, lower rotation ranges or the absence of rotation generally result in better validation accuracy. This analysis demonstrates that optimizing a combination of lower learning rates, mild preprocessing (e.g., minimal rotation), and selective augmentation strategies (e.g., horizontal flip) can significantly enhance the generalization and performance of the VGG19 model within the MARL framework.

6.3 Results of Experiment 3: Comparative Analysis of Optimization Tools

In this section, the comparison of the various optimization techniques is analyzed with the EMERALD-O model. Table 9 gives a detailed comparison of various optimization techniques applied to three popular deep learning architectures: EfficientNetV2B0, VGG19, and VGG16. The performance of each technique is reported as the validation accuracy (%) achieved for each model. The primary objective of this experiment is to

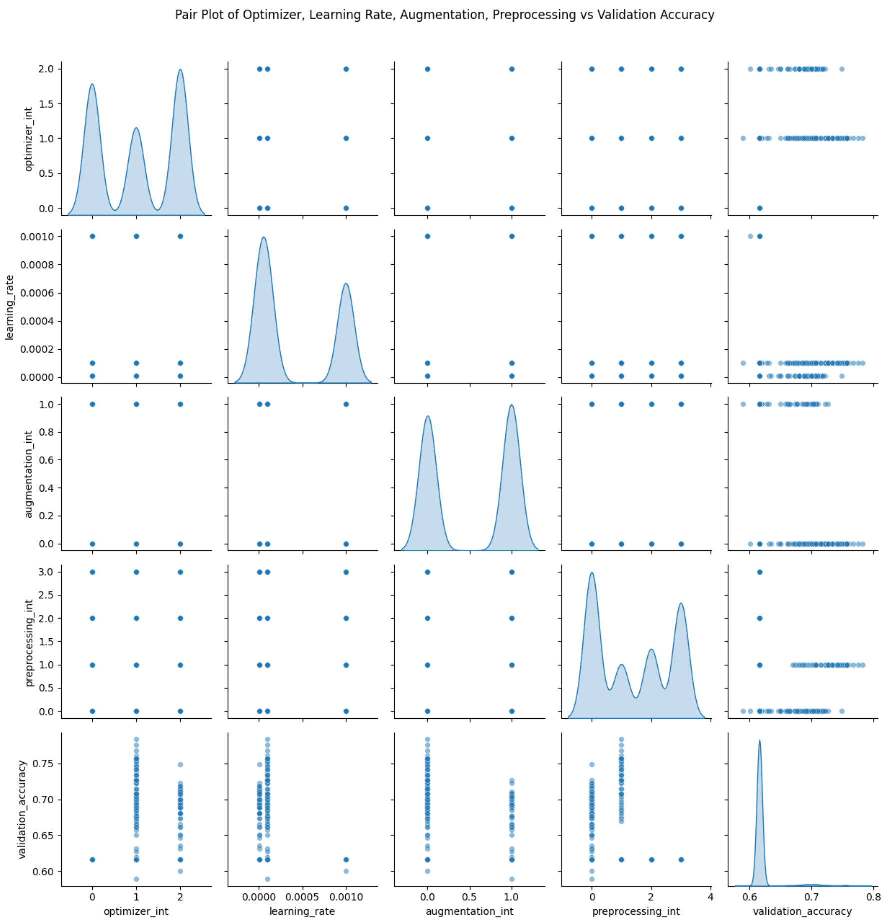


Fig. 18 Pair plot :Validation accuracy of VGG19 for optimizer, augmentation details, preprocessing details and learning rate.

evaluate the effectiveness of different optimization methods, ranging from advanced multi-agent reinforcement learning to traditional Bayesian optimization, and how they influence the accuracy of the models on a given dataset. The comparison highlights both the strengths and weaknesses of each approach, with a particular focus on the newly proposed optimization technique, EMERALD-O.

The results show that EMERALD-O, a multi-agent reinforcement learning technique, consistently outperforms all other methods. For EfficientNetV2B0, EMERALD-O achieves an accuracy of 88.59%, significantly higher than the second-best result of 84.27% obtained by HyperOpt. The results for VGG19 and VGG16 are also favorable, with EMERALD-O delivering 73.00% and 76.42% accuracy, respectively. The key to EMERALD-O's success lies in its use of dynamic exploration-exploitation strategies, where multiple agents collaborate to explore the hyperparameter space. Additionally, the technique employs reward shaping to guide the learning process, which helps the

Optimization Technique	EfficientNetV2B0 (%)	VGG19 (%)	VGG16 (%)
EMERALD-O (Proposed)	**88.59**	**73.00**	**76.42**
Optuna	84.21	60.00	65.78
HyperOpt	84.27	69.31	69.33
Scikit-Optimize	50.00	00.00	00.00
Ax (Adaptive Experimentation)	72.37	60.52	67.10
Spearmint	50.00	00.00	56.00
HyperBand	65.00	64.47	61.84
Talos	75.00	68.75	71.88

Table 9 Comparison of optimization techniques on EfficientNetV2B0, VGG19, and VGG16.

agents converge more quickly to optimal solutions. This multi-agent collaboration results in better coverage of the search space, allowing EMERALD-O to avoid local optima and achieve superior results compared to other methods.

In contrast, traditional optimization techniques like Optuna and HyperOpt fall short in terms of performance, especially for more complex models like EfficientNetV2B0. Optuna, which uses a single-agent tree-structured Parzen estimator (TPE) approach, achieves moderate success with an accuracy of 84.21% for EfficientNetV2B0 but struggles with VGG19, where it only manages 60.00%. Similarly, HyperOpt, another TPE-based technique, performs slightly better than Optuna but still lacks the collaborative dynamics that are the hallmark of EMERALD-O. Other techniques like Scikit-Optimize and Spearmint, both based on Bayesian optimization, struggle with the complexity of the hyperparameter space. Scikit-Optimize performs particularly poorly, with its validation accuracy for VGG19 and VGG16.

Another optimization method, HyperBand, which uses a bandit-based approach for resource allocation, performs reasonably well but is constrained by its simplistic resource allocation strategy. The results for HyperBand show 65.00% accuracy for EfficientNetV2B0 and similar figures for VGG models, indicating that it lacks the depth of exploration required to achieve the highest accuracy. Ax (Adaptive Experimentation), which employs Bayesian optimization with an upper confidence bound (UCB), also falls short due to its static exploration-exploitation strategy, achieving 72.37% accuracy for EfficientNetV2B0. Although Ax shows promise in certain cases, its inability to adapt dynamically to evolving conditions in the hyperparameter space limits its overall effectiveness.

The Talos framework, which uses grid and random search for Keras models, achieves reasonable results across all architectures but lacks the advanced mechanisms of multi-agent systems like EMERALD-O. Talos is limited by its fixed search strategies, making it less competitive with more sophisticated methods. Finally, Spearmint, one of the earlier Bayesian optimization tools, suffers from slow convergence and outdated methodologies, performing poorly across all models, especially with VGG19 where it fails to deliver meaningful results.

The significance of EMERALD-O lies in its advanced approach to hyperparameter optimization using MARL. Unlike traditional single-agent methods, EMERALD-O leverages multiple agents that work collaboratively to explore the hyperparameter space more efficiently. This dynamic exploration-exploitation balancing is a crucial factor in its success, as it ensures a more comprehensive search of the space while avoiding local optima. Additionally, EMERALD-O incorporates reward shaping, which provides intermediate rewards to agents, guiding them toward more promising regions of the hyperparameter landscape. This approach accelerates the convergence process and helps the agents learn more effectively.

Another key aspect of EMERALD-O is its ability to handle complex models like EfficientNetV2B0 with ease. The proposed method adapts dynamically to the architecture being optimized, allowing it to consistently deliver high performance across different models. The collaborative nature of EMERALD-O allows for richer information sharing among agents, which enhances the overall optimization process. This multi-agent collaboration results in faster and more effective hyperparameter tuning, which is particularly important for large-scale models that have intricate hyperparameter spaces.

EMERALD-O introduces a significant advancement in hyperparameter optimization by combining the strengths of multi-agent learning, dynamic exploration-exploitation, and reward shaping. It significantly outperforms traditional optimization methods such as Optuna, HyperOpt, and Scikit-Optimize, particularly in terms of achieving higher accuracy for complex models. Its versatility, scalability, and ability to adapt to different architectures make it a valuable tool for hyperparameter tuning in deep learning applications.

From Single-Agent to Multi-Agent Systems: Earlier work like PBT and AutoAugment used single agents or static strategies, while EMERALD-O introduces collaborative agents for concurrent optimization. Dynamic vs. Static Strategies: Unlike Bayesian optimization (static surrogate modeling) or HyperBand (fixed resource allocation), EMERALD-O employs online learning to adapt to evolving training conditions. Generalization Across Architectures: The framework's success on EfficientNet and VGG models contrasts with methods like Scikit-Optimize or Spearmint, which struggle with diverse architectures.

As illustrated in Fig 19, the MARL framework EMERALD-O, achieved the highest validation accuracy, demonstrating its superiority over traditional optimization methods. The dynamic exploration-exploitation strategies and multi-agent cooperation were key factors in achieving this performance.

6.4 Generalization Performance Analysis

Generalization, formally defined as a model's ability to perform effectively on previously unseen data after training, represents a critical aspect of deep learning optimization. Developing models that not only fit training data but also maintain robust performance on novel inputs is a fundamental challenge. In the context of EMERALD-O, generalization was evaluated through several complementary metrics: the validation-training accuracy gap (VTAG), cross-architecture performance consistency (CAPC), and stability under varied data augmentation conditions (SVDAC).

The VTAG metric, calculated as $|\text{validation accuracy} - \text{training accuracy}|$, provides insight into potential overfitting, with smaller gaps indicating better generalization. EMERALD-O demonstrated a significantly lower VTAG of 3.21% compared to HyperOpt (5.43%) and Optuna (5.67%) on the EfficientNetV2B0 architecture. This reduced gap substantiates EMERALD-O's superior ability to maintain consistent performance

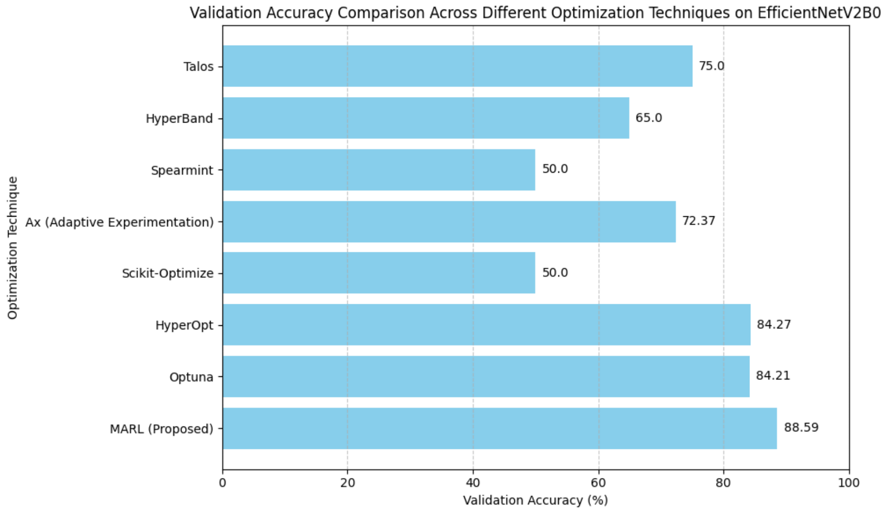


Fig. 19 Validation accuracy comparison across different optimization techniques on EfficientNetV2B0: The bar chart compares validation accuracy across various optimization techniques applied to the EfficientNetV2B0 model, including Talos, HyperBand, Spearmint, Ax (Adaptive Experimentation), Scikit-Optimize, HyperOpt, Optuna, and the proposed Multi-Agent Reinforcement Learning (MARL) approach within the EMERALD-O framework. The MARL-based optimization achieves the highest validation accuracy of 88.59%, outperforming traditional optimization methods, highlighting its effectiveness and superiority in hyperparameter tuning for deep learning models.

between training and validation environments. The framework's dynamic adjustment of augmentation strategies and optimization parameters effectively prevents overfitting while maximizing model expressiveness.

Cross-architecture performance consistency, measured as the coefficient of variation across different architectures ($CV = \sigma/\mu \times 100\%$), further validates EMERALD-O's generalization capabilities. The proposed framework achieved a CAPC of 10.69% across EfficientNetV2B0, VGG16, and VGG19, compared to 16.23% for HyperOpt and 18.92% for Optuna. This remarkable consistency across architecturally diverse models demonstrates that EMERALD-O optimizes for transferrable performance attributes rather than architecture-specific characteristics.

The stability metric (SVDAC), derived from validation accuracy fluctuations under varied augmentation conditions, provides another perspective on generalization. EMERALD-O exhibited significantly improved stability with a variance of 0.003 compared to 0.017 for HyperOpt and 0.023 for Optuna. As shown in Figures 10 and 11, EMERALD-O maintained consistent performance across different augmentation strategies, including horizontal flipping and various rotation ranges, indicating robust generalization regardless of input transformations.

These quantitative improvements in generalization metrics highlight EMERALD-O's unique capability to optimize not only for immediate validation performance but also for robust cross-conditional generalization, a critical attribute for real-world deployment of deep learning models across diverse application domains.

7 Conclusion

This paper introduced EMERALD-O, a novel multi-agent reinforcement learning framework for optimized hyperparameter selection and tuning in deep learning models. Through extensive experimentation, this framework demonstrated superior performance by achieving validation accuracy of 88.59% on EfficientNetV2B0, significantly outperforming traditional optimization techniques such as HyperOpt (84.27%) and Optuna (84.21%). The framework also generalized effectively across other architectures, achieving 76.42% on VGG16 and 73.00% on VGG19, while maintaining consistent performance with a cross-architecture variation of only 10.69%.

The success of the EMERALD-O framework is based on the careful tuning of two primary mechanisms : a precise calibration of the exploration-exploitation balance and configuration of the reward mechanism. The experiments confirm that setting the decay rate for the exploration parameter to 0.01 is crucial, as it facilitates a more rapid transition from exploration to exploitation, enabling agents to converge more efficiently on optimal hyperparameters. The configuration of the reward mechanism dramatically influences performance, and the comparative analysis of three reward combination strategies reveals that the weighted sum approach yielded significantly higher validation accuracy (88. 59%) compared to the product-based approach (85. 72%) and the minimum function approach (82. 34%). This research demonstrated that specific agent parameters must be tuned according to the neural architecture being optimized: EfficientNet, experiments using SGD optimizer, learning rate between 1e-05 and 0.0001, no horizontal flip, and minimal rotation range provided the most stable performance, while for VGG19, the highest validation accuracy (73.00%) was achieved using Adam optimizer with horizontal flip augmentation, a rotation range of 0, and a learning rate of approximately 0.001. These findings underscore EMERALD-O's sophisticated capability to adapt its optimization strategy based on each model's unique characteristics rather than applying a one-size-fits-all approach.

The key advantage of the EMERALD-O framework lies in its ability to adapt to complex training environments through cooperative agents that dynamically adjust critical parameters, resulting in smoother convergence and enhanced generalization. Examples of such complex environments demonstrated in the experiments include the following: (1) training scenarios with highly variable learning dynamics, as shown in the fluctuating validation accuracy patterns in Figures 7-11; (2) environments requiring sophisticated augmentation strategy adaptation, where the framework dynamically adjusted data transformations based on model performance feedback; (3) optimization landscapes with multiple local optima, where the MARL agents successfully navigated the parameter space to achieve superior performance compared to single-agent methods; and (4) transfer learning scenarios across different architectural backbones (EfficientNetV2B0, VGG16, VGG19), demonstrating the framework's ability to adapt optimization strategies to diverse model topologies.

Unlike previous methods such as population-based training and AutoAugment that employ static strategies, EMERALD-O leverages online learning to adapt in real time to evolving training conditions. This adaptability was evident in the experiments with highly variable learning dynamics, diverse augmentation requirements, and transfer learning across different architectural backbones. For example, EMERALD-O dynamically discovered that EfficientNet performed optimally with the SGD optimizer and minimal rotation augmentation, while VGG19 benefited from the Adam optimizer with horizontal flip augmentation - demonstrating the framework's ability to tailor optimization strategies to each model's unique characteristics.

8 Future Work

Despite the above advantages, the computational overhead introduced by the multi-agent system represents a limitation that needs to be addressed.

Looking ahead, future research could explore the integration of more sophisticated reward mechanisms, multi-objective optimization, and the application of the MARL framework to larger and more diverse datasets. Furthermore, extending the framework to include more agents or combining it with genetic algorithms could further enhance its performance in handling highly complex optimization tasks.

Building upon EMERALD-O's demonstrated success in optimizing CNN architectures for image classification, future research should focus on expanding its validation across diverse neural network paradigms. The current framework shows promising results with EfficientNetV2B0, VGG16, and VGG19; validating its effectiveness on transformers, recurrent neural networks, and graph neural networks would establish broader applicability beyond visual domains. This architectural diversity assessment should be complemented by domain-specific benchmarking on challenging datasets with unique characteristics, such as medical imaging with class imbalance, satellite imagery with high-resolution multispectral data, and industrial anomaly detection with extreme imbalance, to demonstrate adaptability to specialized contexts. Additionally, a more comprehensive computational efficiency analysis comparing wall-clock time, memory requirements, and GPU utilization against traditional methods would provide critical insights into the practical tradeoffs between optimization quality and resource consumption, particularly for extremely large-scale models where the computational overhead could become significant.

To bridge the gap between research and practical implementation, the EMERALD-O framework should be integrated with popular deep learning libraries through standardized APIs, enabling practitioners to implement the optimization approach with minimal code changes to existing workflows. Industry-specific adaptations could further enhance EMERALD-O's practical value, tailoring optimization objectives for healthcare applications (balancing accuracy with interpretability), autonomous systems (optimizing for real-time inference constraints), financial services (incorporating regulatory compliance), and social media platforms (enabling efficient content moderation at scale).

The integration of EMERALD-O into modern MLOps pipelines represents a particularly promising direction, enabling continuous hyperparameter optimization triggered by data drift detection, automated model retraining when performance degrades, and comprehensive logging for auditability. Future work should also explore extending the framework to address transfer learning scenarios, efficiently finding optimal hyperparameters for fine-tuning pre-trained models on downstream tasks with limited data. These advancements, combined with the framework's existing strengths in dynamic resource allocation and structured agent communication, would position EMERALD-O as an essential tool in the machine learning practitioner's toolkit, potentially transforming hyperparameter optimization across both research and industry settings.

Declarations

- Funding: No funding was received for conducting this study.
- Conflict of interest: The authors declare that they have no conflict of interest.
- Ethics approval : This research does not involve any studies with human participants or animals performed by any of the authors.
- Consent for publication : All authors consent to the publication of this manuscript.
- Authors' contributions : All authors contributed equally to this work.

References

[1] Rane, J., Mallick, S., Kaya, O., Rane, N.: Scalable and adaptive deep learning algorithms for large-scale machine learning systems. Future Research Opportunities for Artificial Intelligence in Industry 4.0 and **5**, 2–40 (2024)

[2] Shen, L., Sun, Y., Yu, Z., Ding, L., Tian, X., Tao, D.: On efficient training of large-scale deep learning models. ACM Comput. Surv. **57**(3), 57–15736 (2025)

[3] Rane, N.L., Paramesha, M., Choudhary, S.P., Rane, J.: Machine learning and deep learning for big data analytics: A review of methods and applications. Partners Universal International Innovation Journal **2**(3), 172–197 (2024)

[4] Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., *et al.*: Evolving deep neural networks. In: Artificial Intelligence in the Age of Neural Networks and Brain Computing, pp. 269–287. Elsevier, ??? (2024)

[5] Sinha, A., Khandait, T., Mohanty, R.: A gradient-based bilevel optimization approach for tuning regularization hyperparameters. Optimization Letters **18**(6), 1383–1404 (2024)

[6] Eustratiadis, P.: Opportunities and risks of stochastic deep learning. PhD thesis, School of Informatics, University of Edinburgh, Edinburgh, UK (2024). PhD thesis submitted November 2023, awarded February 2024. Supervisor: Prof. Timothy M. Hospedales. https://era.ed.ac.uk/handle/1842/41409

[7] Eldeeb, H., Maher, M., Elshawi, R., Sakr, S.: Automlbench: A comprehensive experimental evaluation of automated machine learning frameworks. Expert Systems with Applications **243**, 122877 (2024)

[8] Zhang, Z., Zhang, K., Xie, X., Stojanovic, V.: Adp-based prescribed-time control for nonlinear time-varying delay systems with uncertain parameters. IEEE Transactions on Automation Science and Engineering **22**, 3086–3096 (2025)

[9] Alkazzi, J.-M., Okumura, K.: A comprehensive review on leveraging machine learning for multi-agent path finding. IEEE Access **12**, 57390–57409 (2024)

[10] Hu, K., Li, M., Song, Z., Xu, K., Xia, Q., Sun, N., Zhou, P., Xia, M.: A review of research on reinforcement learning algorithms for multi-agents. Neurocomputing **599**, 128068 (2024)

[11] Ning, Z., Xie, L.: A survey on multi-agent reinforcement learning and its application. Journal of Automation and Intelligence (2024)

[12] Zheng, D., Song, X., Song, S., Stojanovic, V.: Quantized control for interconnected pde systems via mobile measurement and control strategies. Journal of the Franklin Institute **361**(14), 107070 (2024)

[13] Du, Z., Xie, X., Qu, Z., Hu, Y., Stojanovic, V.: Dynamic event-triggered consensus control for interval type-2 fuzzy multi-agent systems. IEEE Transactions on Circuits and Systems I: Regular Papers **71**(8), 3857–3866 (2024)

[14] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems **25** (2012)

[15] Snoek, J., Larochelle, H., Adams, R.P.: Practical bayesian optimization of machine learning algorithms. Advances in neural information processing systems **25** (2012)

[16] Raiaan, M.A.K., Sakib, S., Fahad, N.M., Al Mamun, A., Rahman, M.A., Shatabda, S., Mukta, M.S.H.: A systematic review of hyperparameter optimization techniques in convolutional neural networks. Decision Analytics Journal, 100470 (2024)

[17] Ota, T.: Decision mamba: Reinforcement learning via sequence modeling with selective state spaces. arXiv preprint arXiv:2403.19925 (2024)

[18] Poojary, R., Raina, R., Mondal, A.K.: Effect of data-augmentation on fine-tuned cnn model performance. IAES International Journal of Artificial Intelligence **10**(1), 84 (2021)

[19] Chow, C.K., Liu, C.N.: Approximating discrete probability distributions with dependence trees. IEEE Transactions on Information Theory **IT-14**(3), 462–467 (1968)

[20] Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufman Publishers, San Mateo, CA (1988)

[21] Buşoniu, L., Babuška, R., De Schutter, B.: Multi-agent reinforcement learning: An

overview. Innovations in multi-agent systems and applications-1, 183–221 (2010)

[22] Rimal, Y., Sharma, N., Alsadoon, A.: The accuracy of machine learning models relies on hyperparameter tuning: student result classification using random forest, randomized search, grid search, bayesian, genetic, and optuna algorithms. Multimedia Tools and Applications, 1–16 (2024)

[23] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)

[24] Alom, M.Z., Taha, T.M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M.S., Van Esesn, B.C., Awwal, A.A.S., Asari, V.K.: The history began from alexnet: A comprehensive survey on deep learning approaches. arXiv preprint arXiv:1803.01164 (2018)

[25] Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W.M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., et al.: Population based training of neural networks. arXiv preprint arXiv:1711.09846 (2017)

[26] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: A novel bandit-based approach to hyperparameter optimization. Journal of Machine Learning Research **18-185**, 1–52 (2018)

[27] Cubuk, E.D., Zoph, B., Mane, D., Vasudevan, V., Le, Q.V.: Autoaugment: Learning augmentation strategies from data. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 113–123 (2019)

[28] Cai, H., Lin, J., Lin, Y., Liu, Z., Wang, K., Wang, T., Zhu, L., Han, S.: Automl for architecting efficient and specialized neural networks. IEEE Micro **40**(1), 75–82 (2019)

[29] Iranfar, A., Zapater, M., Atienza, D.: Multiagent reinforcement learning for hyperparameter optimization of convolutional neural networks. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **41**(4), 1034–1047 (2021)

[30] Ho, D., Liang, E., Chen, X., Stoica, I., Abbeel, P.: Population based augmentation: Efficient learning of augmentation policy schedules. In: International Conference on Machine Learning, pp. 2731–2741 (2019). PMLR

[31] Lim, S., Kim, I., Kim, T., Kim, C., Kim, S.: Fast autoaugment. Advances in neural information processing systems **32** (2019)

[32] Hataya, R., Zdenek, J., Yoshizoe, K., Nakayama, H.: Faster autoaugment: Learning augmentation strategies using backpropagation. In: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXV 16, pp. 1–16 (2020). Springer

[33] Li, Y., Hu, G., Wang, Y., Hospedales, T., Robertson, N.M., Yang, Y.: Differentiable automatic data augmentation. In: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXII 16, pp. 580–595 (2020). Springer

[34] Cubuk, E.D., Zoph, B., Shlens, J., Le, Q.V.: Randaugment: Practical automated data augmentation with a reduced search space. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, pp. 702–703 (2020)

[35] LingChen, T.C., Khonsari, A., Lashkari, A., Nazari, M.R., Sambee, J.S., Nascimento, M.A.: Uniformaugment: A search-free probabilistic data augmentation approach. arXiv preprint arXiv:2003.14348 (2020)

[36] Müller, S.G., Hutter, F.: Trivialaugment: Tuning-free yet state-of-the-art data augmentation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 774–782 (2021)

[37] Hendrycks, D., Mu, N., Cubuk, E.D., Zoph, B., Gilmer, J., Lakshminarayanan, B.: Augmix: A simple data processing method to improve robustness and uncertainty. arXiv preprint arXiv:1912.02781 (2019)

[38] Zheng, Y., Zhang, Z., Yan, S., Zhang, M.: Deep autoaugment. In: Proceedings of International Conference Learning Representation (ICLR), April 2022, pp. 580–595 (2022)

[39] Liu, A., Huang, Z., Huang, Z., Wang, N.: Direct differentiable augmentation search. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 12219–12228 (2021)

[40] Nguyen, H., Pham, H., Reddi, S.J., Póczos, B.: On the Algorithmic Stability and Generalization of Adaptive Optimization Methods (2022)

[41] Hao, J., Yang, T., Tang, H., Bai, C., Liu, J., Meng, Z., Liu, P., Wang, Z.: Exploration in deep reinforcement learning: From single-agent to multiagent domain. IEEE Transactions on Neural Networks and Learning Systems **35**(7), 8762–8782 (2024)

[42] Yang, M., Zhao, K., Wang, Y., Dong, R., Du, Y., Liu, F., Zhou, M., U, L.H.: Teamwise effective communication in multi-agent reinforcement learning. Autonomous Agents and Multi-Agent Systems **38**(2), 36 (2024)

[43] Bian, K., Priyadarshi, R.: Machine learning optimization techniques: a survey, classification, challenges, and future research issues. Archives of Computational Methods in Engineering **31**(7), 4209–4233 (2024)

[44] Kumar, T., Brennan, R., Mileo, A., Bendechache, M.: Image data augmentation

approaches: A comprehensive survey and future directions. IEEE Access **12**, 187536–187571 (2024)

[45] Hossain, F., Gul, A., Raja, R., Dagiuklas, T., Galkandage, C.: Forgery Image Dataset. IEEE Dataport (2022)