Agent Based Model for AUTODL Optimisation

Aroua Hedhili[1,2][a] and Imen Khelfa[1,2][b]
[1]*National School of Computer Sciences, Manouba University, Manouba 2010, Tunisia*
[2]*Research Lab: LAboratory of Research in Artificial Intelligence LARIA, ENSI, University of Manouba, Tunisia*

Keywords: Auto Deep Learning, Multi-Objective Optimization, Collective Intelligence, Agent Model.

Abstract: Auto Deep Learning (AUTODL) has witnessed remarkable growth and advancement in recent years, simplifying neural network model selection, hyperparameter tuning, and model evaluation, thereby increasing accessibility for users with limited deep learning expertise. Nevertheless, certain performance limitations persist, notably in the realm of computational resource utilization. In response, we introduce an agent-based AUTODL methodology that leverages multi-objective optimization principles and collective intelligence to create high-performing artificial neural networks. Our experimental results confirm the effectiveness of this approach across various criteria, including accuracy, computational inference time, and resource consumption.

1 INTRODUCTION

In recent years, researchers have explored the fascinating concept of **Auto**mated **D**eep **L**earning (AutoDL). AutoDL focuses on automating the process of deep learning model design and optimization. It aims to develop techniques and algorithms that can automatically discover the best neural network architectures, hyperparameters, and optimization strategies for a given task, without requiring manual intervention or expert knowledge (Feurer et al., 2015). This concept gained attention around the mid-2010s, but its roots can be traced back to earlier work in the field of machine learning. Since then, numerous research papers and techniques have been proposed such as (Ren et al., 2021), (Elsken et al., 2019), and (Jin et al., 2019). Despite the growing interest among researchers in Auto Deep Learning and the advancements in research within the field, it is still in its early stages of development, it also requires high computational demand(Ahmadianfar et al., 2015). Further, theoretical guidance and experimental analysis are necessary to fully explore its potential. Numerous research works have focused on using multi-objective optimization algorithms with AutoDL. Additionally, some studies have explored the concept of collective intelligence to mitigate the computational costs associated with the search and optimization processes. In light of these advancements, we explore an alternative angle to solve the problem. We propose an agent-based system that exploits the collaborative contribution of agents within an evolutionary multi-objective optimization algorithm. The primary objective of our research is to strike a balance between achieving high accuracy rates, minimizing inference time, and reducing memory footprint in neural network architectures.

First and foremost, we need to define the components of our Neural Architecture Search (NAS) framework. NAS is the process of automating the design of neural architectures for a given task. In fact, interesting AutoDL survey (Elsken et al., 2019) consider that a NAS framework is primarily composed of three key elements: **search space**, **search strategy**, and **performance estimation strategy**. While the creation and selection of deep learning models are inherently multi-objective optimization problems where trade-offs between accuracy, complexity, and inference speed are desired. In our case, we propose the following composition for NAS framework:

- **Search Space (S):** Define a search space S, which represents all possible neural network architectures.

  $$S = \{\text{Architecture}_1, \dots, \text{Architecture}_N\} \qquad (1)$$

  Each architecture in this space is defined by its elements (e.g., convolutional layers, recurrent layers, pooling, skip connections) and its hyperparameters (e.g., kernel size, number of filters, activation functions).

[a] https://orcid.org/0000-0002-6918-0797
[b] https://orcid.org/0009-0007-2055-873X

Hedhili, A. and Khelfa, I.
Agent Based Model for AUTODL Optimisation.
DOI: 10.5220/0012371700003636
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 16th International Conference on Agents and Artificial Intelligence (ICAART 2024) - Volume 3*, pages 568-575
ISBN: 978-989-758-680-4; ISSN: 2184-433X
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

- **Performance Metric (P):** Define a performance metric P, which quantifies the quality of a neural network architecture on the task of interest. This can be accuracy, validation loss, or any other relevant metric.

$$P(\text{Architecture}_i) \in \mathbb{R} \tag{2}$$

- **Optimization Objective (O):** Define an optimization objective O, which specifies what we want to achieve. For example, we might aim to maximize the performance metric while constraining the computational cost.

$$O : \text{Maximize } P(\text{Architecture}_i) \tag{3}$$

- **Search Strategy:** Employ a search algorithm A to explore the search space S and evaluate the performance of different architectures using the performance metric P.

$$\text{A} = \arg \max_{\text{Architecture}_i \in S} O\left(P(\text{Architecture}_i)\right) \tag{4}$$

- **Evaluation Strategy:** measures the performance of the generated network architectures in terms of accuracy, computational resource consumption and inference time.

The remaining sections of this paper are structured as follows: Section 2 covers related works that deal with multi-objective optimization approaches and collective intelligence techniques for NAS. Afterwards, in section 3, we introduce our contribution. Section 4 describes the experimental setup and results. Finally, we conclude the paper by summarizing our findings, discussing limitations, and suggesting future directions for our work.

2 RELATED WORKS

Prior works have investigated the utilization of multi-objective optimization algorithms in NAS to enhance its performance. These approaches aim to identify a set of high-performing neural network architectures that exhibit various trade-offs between accuracy, computational efficiency, and memory utilization. On the other hand, the concept of collective intelligence has been explored to alleviate the search and to optimize costs in NAS.

2.1 Multi-Objective Optimization with Collective Intelligence

Multi-objective optimization is a mathematical optimization technique that deals with finding the optimal solutions to problems with multiple, often conflicting, objectives. The task is to find a set of solutions known as the Pareto front (or Pareto set), representing the best compromise between these objectives.
A multi-objective optimization problem includes Objective Functions each representing a different aspect of the problem. These objective functions can be represented as follows (Arora, 2017):

$$f(x) = (f_1(x), f_2(x), \ldots, f_n(x)) \tag{5}$$

where $f(x)$ is a vector of objective function values, x is the decision variable vector, and n is the number of objectives.
Several works treat multi-objective optimization problems in AutoDL context. (Dong et al., 2018), (Elsken et al., 2018), (Lu et al., 2019), (Real et al., 2019) and in this paper we focus on multi-objective optimization problems with collective intelligence. In fact, NAS often tackles the challenge of optimizing multiple objectives concurrently, such as enhancing model accuracy, reducing model size, and improving inference speed. To address this, we believe that collective intelligence techniques can be used to handle multi-objective optimization problems effectively. Actually, collective intelligence refers to the shared intelligence and problem-solving capabilities that emerge from the collective efforts of a group of individuals (Bigham et al., 2015). It involves the aggregation of diverse knowledge, perspectives, and skills from group members to achieve better outcomes than what could be achieved by individuals working alone (Bigham et al., 2015). Next, we review the most important and recent works in this context.

Cetin and Gundogmus (Cetin and Gundogmus, 2019) drew inspiration from Daniel Kahneman's book "Thinking, Fast and Slow" (Kahneman, 2015) for their work. In the book, Kahneman introduces the metaphor of two cognitive systems, System 1 and System 2, representing fast and slow thinking, respectively. System 1 operates intuitively and automatically, while System 2 engages in focused and critical thinking. They represented these systems in two agents, each agent represent Evolutionary Genetic Algorithms (EGAs) with different mutation rates. The main problem with this solution is that the algorithm's efficiency and computational requirements may become a limitation as the dataset size and number of features increase. Moreover, the algorithm's hyper-parameter settings are provided for a toy dataset and applied to real datasets, but there is no systematic exploration of hyper-parameters for different types of real datasets.
The work (Zoph et al., 2018) introduced a search method based on reinforcement learning (RL). In

their approach, they employ controllers to generate architectural hyperparameters for neural networks. These controllers are implemented as recurrent neural networks, and they predict various parameters like filter height, filter width, stride height, stride width, etc. Each instance of the controller generates m different child architectures, which are trained concurrently. Afterward, the controller collects gradients based on the outcomes of this batch of m architectures when they converge and sends these gradients to the parameter server for weight updates across all controller replicas. Their limitations lie in the absence of metaheuristics in their reinforcement learning methods, as they rely on empirical predictions, resulting in slow performance or excessively lengthy processing times to achieve satisfactory results.
In addition, (Gupta and Raskar, 2018) propose an agent based method. They define multiple agents for a distributed deep learning training. This algorithm showcased promising results for optimizing the learning process. However, as the number of agents increases, so do computational resource requirements, and managing communication between agents becomes more complex.

2.2 Discussion

In the following table, we emphasize the advantages and limitations of the use of collective intelligence for treating multi-objective problems.

In the context of addressing multi-objective optimization, the use of collective intelligence principles, where agents work collaboratively, offers several advantages. First and foremost, collaboration among agents allows for the aggregation of diverse knowledge, perspectives, and skills, as stated by Bigham et al (Bigham et al., 2015)). This diversity can lead to more comprehensive problem-solving approaches and a broader exploration of the solution space, which is particularly valuable in multi-objective optimization scenarios where finding a diverse set of Pareto-optimal solutions is essential. Additionally, collaborative agents can leverage their individual strengths, such as different mutation rates or optimization strategies, to enhance the overall optimization process. This collaboration can lead to more efficient convergence towards Pareto-optimal solutions, making the collective intelligence approach highly promising.

However, while collaboration among agents in multi-objective optimization has its advantages, it also presents significant limitations. One notable limitation is the increased demand for computational resources as the number of agents or the complexity of the optimization problem grows as (Gupta and Raskar, 2018). Additionally, managing communication and coordination among a large number of agents can become complex, potentially leading to efficiency and scalability issues. This complexity may hinder the practicality of collective intelligence approaches.

To address these challenges, our approach introduces an agent-based AUTODL (Automated Deep Learning) model that distributes the optimization of neural network search across multiple agents. The evolution of this search process leverages metaheuristic search techniques, specifically genetic algorithms, to mitigate the time required for exploration. In this setup, each agent is responsible for optimizing a specific objective and applies genetic operators, such as mutation and cross-over, to refine the solutions. Additionally, agents collaboratively share learnable parameters and engage in structured interactions with one another. This structured collaboration significantly reduces search time and expedites the convergence towards the optimal Artificial Neural Network (ANN) model, all without incurring high computational costs.

3 OUR CONTRIBUTION

This section introduces our solution MOCA(Multi-Objective and Collaborative Auto-DL) approach. It describes a rapid multi-objective NAS algorithm that employs an elitist genetic algorithm, incorporating a collective intelligence strategy. The primary objective of MOCA is to produce neural network architectures that are both high-performing and cost-effective. Drawing inspiration from biological concepts like natural selection and the wisdom of the crowd, our algorithm initiates by generating a population of networks and applies operations such as mutation and cross-over, weight sharing and parameter optimizer to produce an offspring of candidate network architectures. These candidates interact and exchange knowledge through the aforementioned operators, resulting in an emergent intelligence that accelerates their learning process. Our aim is to strike a balance between multiple objectives such as accuracy, inference time, and resource consumption. Across generations, our Neural Network Candidates (NNCs) continuously optimize their outcomes, striving towards a shared micro architecture goal where all agents pursue the aforementioned objectives individually. Each candidate/agent acts as a single player, sharing their acquired knowledge with others. Additionally agents operate within a macro architecture, each focusing on optimizing a specific goal. The top-performing agents with different roles are subsequently com-

Table 1: Comparison of Collective Intelligence Methods for Multi-Objective Optimization.

Methods	**Advantages**	**Limits**
EGA Algorithm (Cetin and Gundogmus, 2019)	Decentralized architecture	High computational cost, coordination complexity.
Reinforcement Learning (Zoph et al., 2018)	Controller-based architecture, concurrent training	Slow convergence, substantial computational resources.
Distributed Agents (Gupta and Raskar, 2018)	Collaboration among agents, robust optimization process	Increasing computational resources with more agents.

bined to align with our multiple objectives. To enhance the organization of our approach, we have established a structure comprising three primary components: **Nodes**,**Operations** and **Search Strategy** as shown in the figure 1MOCA components..

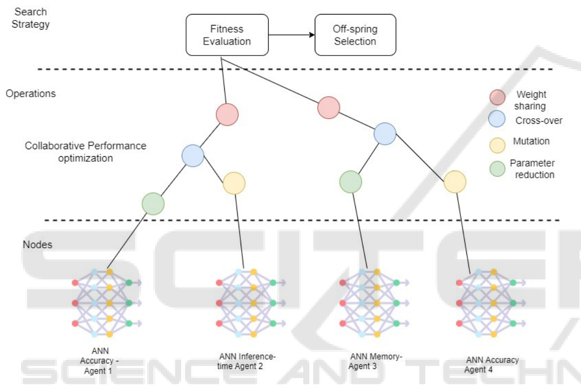


Figure 1: MOCA components.

3.1 Nodes

The first component represents the agents of our search space. It includes our candidate neural network architectures. Since it contains a set of neural networks, it inherits its complexity and variation in terms of parameters and learning algorithms. We should also take into account additional design considerations because there are multiple nodes. These include determining the degree of similarity or diversity among nodes, deciding which input data to be passed to each node, and defining the shared output data between nodes. Therefore, we associate each agent with a set of features labeled **PRIC** (**P**arameters, **R**ole, **I**nteraction, **C**ontribution), we detail next each feature.

Parameters (P): This includes both the architectural and non-architectural hyperparameters of the node, as well as the parameters acquired through training. We encode each network as a genome, consisting of a subset of genes, where each gene represents an architectural hyper-parameter, parameters that are learned through training process and optimization functions.

Role (R): This defines the objective or purpose of the agent associated with the node. It indicates what the agent aims to optimize such as maximizing accuracy, minimizing inference time.

Interactions (I): This provides insights into the communication and information exchange among different agents. It describes how agents interact together. Interactions may represent mutation, crossover or parameter sharing.

Contribution (C): This is a score assigned to each agent based on its level of participation and value in the search process. It quantifies the agent's contribution to the overall exploration and optimization.
By considering these PRIC features, we can obtain a more comprehensive understanding of our agents within the system and better analyse their roles, interactions, and contributions.

3.2 Operations

In our proposed approach, our agents can perform the following operations:

Mutation: The mutation process involves randomly changing the parameter of the parent model to generate the offspring network. For example, MOCA randomly changes the parameters by random selection (the number of layers changes from 10 to 15). Mutation mainly focuses on exploring the solution space in the neighborhood of the original solution. Our goal is to use Pareto-dominated models and maximize their performance as much as possible.

Crossover: To generate an offspring network, we employ crossover by selecting two networks and splitting their corresponding genomes at a random architectural hyperparameter. One genetic fragment from each parent model is exchanged to produce the offspring network. In MOCA, two parent models

are randomly chosen, and their hyper-parameters are crossed over at a random index. For instance, the first fragment of parent network ”i” is combined with the second fragment of parent network ”j”, resulting in the offspring genome with crossover. The purpose of crossover is to increase the diversity of the population and explore novel solutions. As the population's performance improves over generations, crossing over random models provides an opportunity to generate better solutions by allowing their parent models to exchange their beneficial architectural hyperparameters.
To successfully achieve agents' behaviors, we add the following parameters:

- Weight sharing: To reduce the training time of the candidate network and reduce the resource consumption, we use the candidate networks that give the best performance in terms of accuracy and distribute their weights among the remaining candidate networks with the same topology to speed up the divergence by obtain satisfactory accuracy. Therefore, MOCA iteratively evaluates the performance of the search space and propagates the weights of the best performing network to the remaining networks to improve the overall performance.
- Parameter reduction: It refers to techniques that aim to reduce the number of parameters in a machine learning model. The main motivation behind parameter reduction is to achieve model simplification, improve model efficiency, and mitigate the risk of over-fitting. This process can be guided by various approaches, the approach we used for parameter reduction was magnitude-based pruning (Park et al., 2020). This approach involved identifying and removing parameters with magnitudes below a certain threshold. Specifically, after training the models, the weights of the top-performing models were pruned by setting small-magnitude weights to zero. This resulted in a reduction in the number of non-zero parameters, thereby reducing the overall parameter count of the models.

3.3 Search Strategy

We use genetic algorithm as our optimization technique stems from its ability to provide flexibility in determining fitness criteria and representing computational costs of models. Our primary aim is to decrease the search time for Neural Architecture Search (NAS). By employing genetic algorithm, we establish a fitness criteria that ensures the preservation and development of high-performing networks by passing all the Pareto non-dominated models from each generation to the subsequent one. This approach allows us to prioritize short-term rewards while simultaneously enhancing the population's diversity, enabling the exploration of new solutions that could potentially lead to better networks in future generations. MOCA starts by randomly sampling a population of a predetermined ”N” number of networks. These networks represent our agents (N agents). As we mentioned, agents may have various PRIC specifications. Therefore, we can have agents with different architectures and different goals. We may have agents that follow CNN architectures (let's call them CNN-agents), and another may follow RNN architectures (RNN-agents). These agents may be divided into sub-networks depending on the objective they're trying to optimize. For instance, we may have CNNA-agent (CNN-Accuracy-agent) and CNNI-agent (CNN-Inference-Time-agent). Thus, communication between these agents may take more than one form. Within MOCA, we can have mainly three communication forms as showcased in the figure

- Form 1: Agents that have the same architecture and same objective (they work to optimize their accuracy). After E epochs, these agents will operate cross-over to exchange hyper-parameters. The top performing agent will share his weights with his colleges.
- Form 2: Agents that have the same architecture but different objectives. After training, the top-performing agents in terms of accuracy will be cloned. The resulting agents will represent a new agent that optimizes its inference time by reducing its number of parameters.
- Form 3: Agents that have different architectures and same objective (they work on optimizing their accuracy). These agents will operate cross-over by swapping the layers at a random crossover point. Thereby, they generate new offspring and add diversity to the search space. Moreover, in order to enrich our search space, agents can operate mutation at a random point. The mutated artificial neural network represents the child network, which will be passed down to the next generation along with its parent network.

3.4 MOCA Search Strategy Algorithm

The algorithm 1 outlines the search strategy used in our solution MOCA. We start by sampling a population consisting of K random architectures. We define different randomly selected hyper-parameters,

```
Data: Random neural network architectures
Result: Efficient artificial neural network in terms
        of accuracy, inference time, and memory
        footprint
Initialization;
while Generation < G do
    Calculate fitness for each model;
    Sort models by fitness;
    Clone the top N models;
    for model in top N models do
        if model is original then
            Reduce the parameters of the original
              set;
            Pass the original set to the next
              generation;
        end
        if model is a clone then
            Share weights between original and
              cloned models;
            Mutate the cloned models;
            Perform cross-over;
            Pass the generated offspring to the
              next generation;
        end
    end
end
Select the best model;
```

Algorithm 1: MOCA Algorithm.

and then we run the algorithm 1 for G generations. For every generation, we evaluate the trained models and duplicate them into two sets: the parent set and the cloned set. We pass the parent set without genetic modifications to allow the good solutions to reproduce and evolve as much as possible over the generations. For the cloned set, we generate M networks by crossing over two randomly selected parent networks that achieve Top N accuracy. The goal of this step is to enhance the diversity of the population. We exploit the top-performing networks by mutating the second set in an attempt to further develop their performance. We also propagate the weights of these parent networks to speed up the learning process for the upcoming generations. On the other hand, these top performing agents will be cloned and act as new agents, trying to optimize their running time by reducing their number of parameters. We repeat this process for the G generation until a unique model is selected.

4 EXPERIMENTAL SETUP

In preparation for our experiments, we subjected the CIFAR-10 dataset [1] to essential preprocessing steps to ensure its compatibility with deep learning models. The dataset is organized into 10 categories. Each category contains 6,000 images divided into 40,000 for training, 10,000 for testing, and 10,000 for validation. This division scheme ensures that the model is trained on a substantial portion of the data and validated on a distinct subset to assess generalization performance. Then, the process of data normalization was employed to scale pixel values within a standardized range of [0, 1].
For our experiments, we opted to use CNN (Convolutional Neural Networks) as the underlying model architecture. CNNs have demonstrated exceptional performance in image classification tasks due to their ability to capture spatial hierarchies and features.
The code and the different parameters and settings are available. [2]

4.1 Evaluation Metrics

We selected multiple evaluation metrics to align with our research goals and provide a comprehensive assessment of model quality: **Accuracy**, **F1score**, **Inference time** and **Memory footprint** and the **Fitness function** 6.

$$\begin{aligned} Fitness = {} & \text{Accuracy} \times w1 - \text{Memory Footprint} \times w2 \\ & - \text{Inference Time} \times w3 + \text{F1-score} \times w4 \\ & + \text{Contribution} \times w5 \end{aligned} \tag{6}$$

where w1, w2 w3, w4 and w5 represent weights varying in the interval [0,1] that determine the priority of each constraint; the higher the value, the more important the constraint. These values are problem specific and user definable.
Following each iteration, we identify the top 5 models with the highest fitness values. From the selected top models, we make duplicate copies of these 5 models. This step ensures that we retain and continue to work with the best-performing architectures. Then, we undertake weight pruning for these duplicated models. This involves selectively removing unnecessary connections or weights within the model architecture. Weight pruning aims to simplify the models while preserving their performance. To introduce diversity and further refine the models, we apply mutation to the cloned models. Simultaneously, we transfer weights from parent models to assist in the learning process. This combination of mutation and weight transfer contributes to the optimization of model architectures. We promote knowledge exchange and exploration by facilitating cross-over operations between models. This genetic-inspired tech-

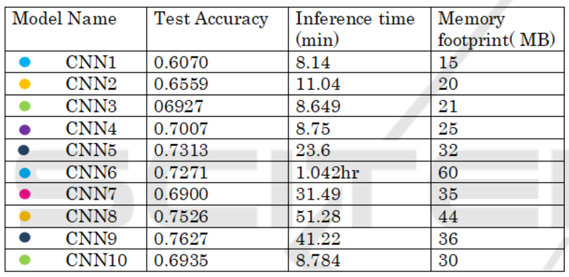
[1]https://www.cs.toronto.edu/ kriz/cifar.html

[2]https://www.kaggle.com/arouahedhili/moca-algorithm

nique allows us to create novel architectures by combining features from different high-performing models. Through these sequential steps, we iteratively advance and fine-tune our model population. This iterative process leads to the discovery of architectures that excel in terms of our chosen evaluation metrics. In the next section, we present the results obtained following the implementation of our solution.

4.2 Results and Comparison

For the first generation, we randomly sampled 10 CNN models using the parameters detailed in table 2, then we trained the models for 10 epochs. We used the same number of epochs during our experiment so we reduce the algorithm search time and propagate the models that prove good performance in these epochs to the next generation. The figure 2Initial generation performance. represents the performance of the initial population. Over 5 generations, We evalauted the



Model Name	Test Accuracy	Inference time (min)	Memory footprint(MB)
● CNN1	0.6070	8.14	15
● CNN2	0.6559	11.04	20
● CNN3	06927	8.649	21
● CNN4	0.7007	8.75	25
● CNN5	0.7313	23.6	32
● CNN6	0.7271	1.042hr	60
● CNN7	0.6900	31.49	35
● CNN8	0.7526	51.28	44
● CNN9	0.7627	41.22	36
● CNN10	0.6935	8.784	30

Figure 2: Initial generation performance.

population based on accuracy, memory footprint, F1-score, we ranked them based on the weighted sum of these scores and assigned the rank of each model as it's contribution to the search process.

In our experimental findings, we observed promising outcomes when transferring weights from a more accurate model to a less proficient counterpart. The superior model, having demonstrated excellence in the same image classification task on an identical dataset, provided a reservoir of learned features aligned with our research objectives. The shared representations within the architectures, especially in the convolutional layers, facilitated a smooth transfer of both low-level and high-level image features, hastening the convergence of the less successful model. The success of this knowledge transfer was reinforced by the intrinsic similarity of tasks, affirming the effectiveness of utilizing pre-trained weights to boost overall model performance. Notably, this approach also led to a reduction in memory consumption, as the models required less training, thereby mitigating computational resource demands. The table 2Performance of the best 5 models on Fashion MNIST dataset. represents the results of the best models performance over the 5 iterations:

In addition, we compare in Table 3Comparaison of MOCA and baseline algorithms performance. the performance of our approach with the performance of other approaches that considered using collective intelligence approach for NAS. Guerrero-Viu et al. (Guerrero-Viu et al., 2021) propose a baseline of collaborative multi-objective optimization algorithms. These algorithms tend to evaluate their algorithms on accuracy and number of parameters. So for this comparaison we will consider the use of these metrics. They run each algorithm 10 times on Fashion Mnist dataset. Therefore, we also implement MOCA algorithm with Fashion Mnist dataset [3]. They used a maximum budget of 25 epochs for training every model. Their search space is populated with randomly sampled CNN models.

The obtained results in (Guerrero-Viu et al., 2021) showcase that after 10 runs of tested algorithms, we can notice that they achieve high accuracy (superior to 0.9). This very close to our obtained results after the same number of runs. Furthermore, in their findings, it's notable that models achieving high accuracy still retain a significant number of parameters. As mention in the table 2Performance of the best 5 models on Fashion MNIST dataset., we could strike a balance between producing highly accurate models while significantly reducing the number of parameters and mitigating computational consumption.

5 CONCLUSIONS

In this paper, we describe MOCA, an agent based AUTODL approach based on the concept of multi-objective optimization and collective intelligence techniques. This solution describes a new search strategy for neural architecture search using agents and genetic algorithm starting from an initial randomly sampled generation and ending with finding the best model achieving high performance in term of multiple criteria. In our experimentation, we provided a proof of concept for image classification task. The achieved results highlight the effectiveness of the MOCA algorithm in optimizing multiple objectives simultaneously. We primarily explored CNN architectures due to their effectiveness in image classification. Future research could investigate the adaptation of the MOCA algorithm to different model types. Additionally, we propose incorporating transfer learning

[3]https://www.kaggle.com/code/imenkhelfa/moca-fashion-mnist

Table 2: Performance of the best 5 models on Fashion MNIST dataset.

Best Model	Accuracy	F1-Score	Memory Footprint(MB)	Inference Time
Fashion MNIST				
M1	0.895	0.850	28	10:22:35
M2	0.906	0.887	25	6:53:43
M3	0.914	0.890	21	7:10:12
M4	0.946	0.902	25.7	24:21:52
M5	0.961	0.941	22.3	17:15:25

Table 3: Comparaison of MOCA and baseline algorithms performance.

Algorithm	Performance
SH-EMOA	$\approx 0.92 / 15.3$
MO-BOHB	$\approx 0.93 / 32.0$
MS-EHVI	$\approx 0.90 / 9.5$
MO-BANANAS-SH	$\approx 0.93 / 19.3$
BULK & CUT	$\approx 0.94 / 15.3$
Random Search	$\approx 0.92 / 38.1$
MOCA	$0.96 / 22.3$

techniques into the MOCA algorithm could expedite model convergence and boost performance.

REFERENCES

Ahmadianfar, I., Adib, A., and Taghian, M. (2015). A multi-objective evolutionary algorithm using decomposition (moea/d) and its application in multipurpose multi-reservoir operations. *Iran University of Science & Technology*, 5:167–187.

Arora, J. (2017). Chapter 18 – multi-objective optimum design concepts and methods. In *Multi-objective Optimum Design Concepts and Methods*.

Bigham, J. P., Bernstein, M. S., and Adar, E. (2015). Human-computer interaction and collective intelligence. *Handbook of collective intelligence*, 57(4).

Cetin, U. and Gundogmus, Y. E. (2019). Feature selection with evolving, fast and slow using two parallel genetic algorithms. In *2019 4th International Conference on Computer Science and Engineering (UBMK)*, pages 699–703. IEEE.

Dong, J.-D., Cheng, A.-C., Juan, D.-C., Wei, W., and Sun, M. (2018). Ppp-net: Platform-aware progressive search for pareto-optimal neural architectures. *arXiv preprint arXiv:1806.08198v2*.

Elsken, T., Metzen, J. H., and Hutter, F. (2018). Efficient multi-objective neural architecture search via lamarckian evolution. *arXiv preprint arXiv:1804.09081*.

Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. (2015). Efficient and robust automated machine learning. *Advances in neural information processing systems*, 28.

Guerrero-Viu, J., Hauns, S., Izquierdo, S., Miotto, G., Schrodi, S., Biedenkapp, A., Elsken, T., Deng, D., Lindauer, M., and Hutter, F. (2021). Bag of baselines for multi-objective joint neural architecture search and hyperparameter optimization. *arXiv preprint arXiv:2105.01015*.

Gupta, O. and Raskar, R. (2018). Distributed learning of deep neural networks over multiple agents. *Journal of Network and Computer Applications*, 116:1–8.

Jin, H., Song, Q., and Hu, X. (2019). Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1946–1956.

Kahneman, D. (2015). Kahneman's thinking fast and slow: From bestseller to textbook: Thinking, fast and slow. *RAE Revista de Administracao de Empresas*.

Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., and Banzhaf, W. (2019). Nsga-net: neural architecture search using multi-objective genetic algorithm. In *Proceedings of the genetic and evolutionary computation conference*, pages 419–427.

Park, S., Lee, J., Mo, S., and Shin, J. (2020). Lookahead: a far-sighted alternative of magnitude-based pruning. *CoRR*, abs/2002.04809.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. (2019). Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789.

Ren, P., Xiao, Y., Chang, X., Huang, P.-Y., Li, Z., Chen, X., and Wang, X. (2021). A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys (CSUR)*, 54(4):1–34.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710.