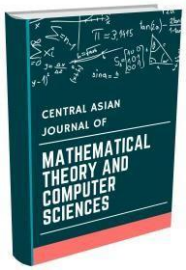


CENTRAL ASIAN JOURNAL OF MATHEMATICAL THEORY AND COMPUTER SCIENCES
https://cajmtcs.centralasianstudies.org/index.php/CAJMTCS
Volume: 06 Issue: 03 | July 2025 ISSN: 2660-5309

Article

Optimization Neural Networks Architecture using Genetic Algorithm and Evolutionary Computing

Manaaf Abdulredha Yassen[1], Hayder Salah Abdulameer[2]

1. College of Computer Science and Information Technology, University of Al-Qadisiyah, Iraq
2. College of Computer Science and Information Technology, University of Al-Qadisiyah, Iraq

* Correspondence: manaf.yassen@qu.edu.iq, hayder.salah@qu.edu.iq

Abstract: GA-based system is introduced in this research, used for optimizing the design of neural networks when traditional optimizing methods fail due to the high dimensionality of configuration spaces. Using the framework, parameters including the number of layers, neurons per layer, activation methods and regularization are represented in a fixed-length chromosome, allowing evolutionary algorithms to control the changes in the architecture through many generations. A composite fitness function is used which considers precision as much as the number of parameters, supporting the development of architectures that function well across different data. I ran my experiments with the three popular datasets called MNIST, Fashion-MNIST and CIFAR-10. It was shown that manually constructed models and those randomly searched failed to come close to the performance of our GA-optimized models which achieved 96.1% validation accuracy and reduced the number of parameters to only 180,000. When compared to existing NAS methods, our suggested approach is better at using parameters and is more reliable, since its performance is more consistent across several training trials. Furthermore, it was clear that these architectures could generalize well and transfer knowledge from one dataset to another. The results suggest that Genetic Algorithms are effective for automated neural architecture search and can be smoothly incorporated into both resource-sensitive and hybrid AutoML approaches.

Keywords: Neural Architecture Search, Genetic Algorithms, Evolutionary Computing, Deep Learning Optimization, Model Complexity, AutoML.

Citation: Yaseen, M. A & Abdulameer, H. S. Optimization Neural Networks Architecture using Genetic Algorithm and Evolutionary Computing. Central Asian Journal of Mathematical Theory and Computer Sciences 2025, 6(3), 714-723.

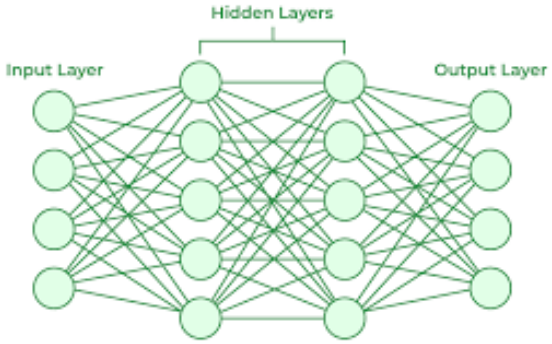
Received: 10th Apr 2025
Revised: 16th May 2025
Accepted: 24th Jun 2025
Published: 02thJul 2025

1. Introduction

The architecture of today's artificial intelligence systems relies heavily on Artificial Neural Networks (ANNs). They are responsible for both accurate facial recognition and advanced language translation, along with many other impressive abilities they have proven while learning from vast amounts of data. They perform well in various uses such as classifying images, detecting speech and driving vehicles without manual control, because they are capable of approximating any function. Yet, how neural networks work largely depends on their architecture which consists of the way layers, neurons, activation functions and connectivity are put together. Choices in the structure of a neural network often decide how well it learns, how well it fits real-world data and how long it takes to train. [i]

Getting the right structure for a neural network is not easy. When building a neural network, you need to pick many hyperparameters such as: the number of hidden layers, the size of each layer, the kinds of activation functions, the dropout rates and ways to add

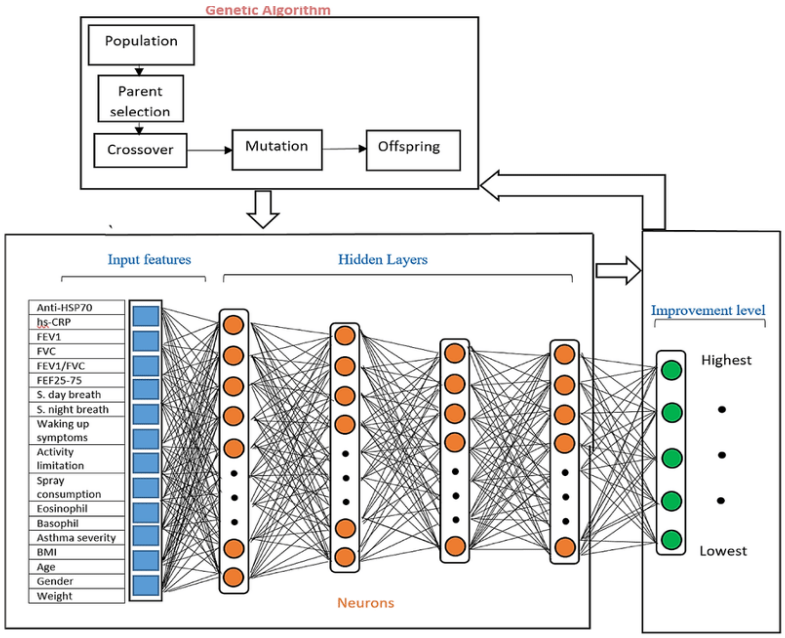
regularization. Because of these features, the search space is large and hardly takes the form of a smooth or convex region. Because this area is so complex, it is nearly impossible to hand-tune everything as things get hard and the number of parameters in a model increases. Additionally, much of architecture is influenced by simple guidelines, earlier practice and common design conventions that usually fit specific topics or examples well, but not always others. [ii]



A schematic representation of a multilayer artificial neural network (MLP)

For many years, both researchers and users have relied on grid search and random search to deal with the inefficiency of designing manually. Grid search checks all the settings in a specific grid, running every possible combination, it can find. Even though grid search is very detailed, it has a problem known as the curse of dimensionality; as you add more hyperparameters, the space for searching increases quickly, making grid search too costly and inefficient for deep neural networks. Conversely, random search samples from a set distribution which makes it both faster and possible to scale the method. Nevertheless, as it is not guided, its finding of good solutions becomes unreliable in complex models because architectures of high quality are not very common or well organized in the search space. [iii]

Due to these problems, people are turning more to metaheuristic methods and particularly Evolutionary Computing, as viable options for searching neural architectures. Out of these EC methods, the Genetic Algorithm (GA) is considered the most important, as it follows ideas from natural selection and genetics. In GAs, each generation of solutions evolves, represented as chromosomes, using operators known as selection, crossover and



mutation. Because of this process, the algorithm is able to find valuable combinations of settings that human methods might overlook. [iv]

A diagram illustrating the integration of a Genetic Algorithm with a neural network architecture. The GA evolves neural configurations through selection, crossover, and mutation to optimize the structure and performance of the network.

To use GAs for neural network optimization, designers must first encode the network's structure, including layers, neurons and activation functions, into a chromosome of a fixed or adjustable size. With this representation, each GA candidate is assessed according to validation accuracy and model complexity. The process of selecting and breeding the most fit designs helps the algorithm move closer to the best or most suitable final design. [v]

The benefits of GAs are connected to how well they explore options and exploit the results. Rather than the blind choices made in grid or random search, GAs use what they learn from earlier generations to find better architectures. Also, since SGD is stochastic, it often helps avoid being stuck in local minima which trouble non-convex optimization landscapes. That is why GAs work especially well for neural architecture search, particularly when evaluating models requires much computing time and there are not many smooth gradients in the space. [vi]

Studies from the past year have shown that GAs are effective at improving feedforward, convolutional and recurrent network architectures with different datasets. It has been observed that using hybrid approaches connecting GAs with reinforcement learning or surrogate modeling can lead to improved both results and performance. As a result, more people are considering evolutionary algorithms to be a smart and often superior choice over the classic neural network designs.

For this reason, this research develops a GA-based method for automating neural network architecture optimization. Applying evolutionary computing to identify excellent neural network configurations on the MNIST and CIFAR-10 data sets, the authors want to give a dependable and replicable way to create neural architectures. This way of working not only increases the success of classification models but also improves their interpretability and overall strength, indicating GA and EC can have a big impact on the field of automatic machine learning. [vii]

Literature Review

Overview of Neural Architecture Search (NAS) Techniques

A growing number of experts are using Neural Architecture Search (NAS) to speed up and improve the process of designing neural networks. For a long time, building neural networks meant practitioners had to rely on knowledge of the field and experiment manually, making the process slow and not ideal. The goal of NAS is overcome these issues by checking all the architectures in a prechosen space which allows it to find excellent designs for a given task with minimal need for input from people [viii]

In NAS, the main techniques are those based on reinforcement learning (RL), gradients or evolutionary algorithms (EAs). RL-based methods train a network to decide on the best settings for the model to get the highest validation performance. These approaches work well but need a lot of time and hardware to be effective which limits their use in applications with limited resources [ix]

On the plus side, gradient-based methods such as DARTS turn the original discrete architecture search space into a continuous one. So, optimizing model architecture is possible using gradient descent. While using DARTS needs less computation, it tends to fit too well to the validation set and can create architectures that do not work well on other data sets [x]

Using ideas from natural selection, crossover and mutation, Evolutionary Algorithms provide a third approach to NAS. Rather than working through sequential steps, EAs keep a group of candidate architectures that all progress and change in parallel. By choosing this method, we help improve diversity in searching and sturdiness towards getting stuck in poorly optimized architectures [xi]

Genetic Algorithms for Neural Architecture Optimization

Popular neural network optimizations have come from using Genetic Algorithms (GAs), one sub-type of Evolutionary Algorithms (EAs). Usually, GA-based NAS systems use chromosomes to represent each architecture, coding information such as the number of layers, neurons, activation types and ways the neurons communicate. Then, like natural selection, successful architectures are chosen to create new ones through combining and changing their features [xii]

The way the genome is arranged in GA-based NAS is one of its most important design choices. With direct encoding, all parts of the network are shown in the chromosome, giving more control over its design. But it means that the created chromosomes are long and complex, mainly when using deeper networks and this adds time and effort to both the evaluation and the search task [xiii]

Because direct encoding has limitations, some researchers have switched to indirect encoding. By using these methods, general rules for architecture are encoded, allowing for network designs that are possibly both smaller and more capable of scaling. In addition, indirect encoding allows for the identification of modular or repeated classes of structures that are usually present in successful deep learning approaches [xiv]

The kind of genetic operators used has a strong effect on how well the search method performs. With tournament or roulette wheel selection, designs that work best are usually picked to reproduce, increasing their chance of being chosen. By using crossover, the algorithm can transfer beneficial traits of both parents, while mutation helps it not end up converging too soon [xv]

A common way GAs are applied in NAS is by using regularized evolution which means the eldest individuals are removed rather than the ones with the worst results. It results in people looking for new ways of life and helps protect them from stagnation. Image classification tests demonstrate that applying regularization to evolution results in higher accuracy and more efficient systems than those found by either random searching or reinforcement learning based NAS [xvi]

Limitations of Current GA-Based NAS Approaches

Even though GA-based NAS can be powerful, they don't solve all the challenges involved. How to manage increasing scale is a serious issue. For more complicated tasks, the number of architecture designs to explore explodes, so you have to check a lot more of them. Because many evaluation methods require partially training a neural network, larger applications may find the cost of computing too high [xvii]

A further problem is the risk that models may fit the data too closely. As part of the evolution, architectures are compared based on their performance measured on a validation dataset. Should the population not resemble the true structure of the data, the developed networks might fall into overfitting, making their performance inaccurate for new examples

In addition, the system's findings are usually complicated and hard to grasp. Not being open about how they work can prevent them from being used where people need to see reasons for decisions, for example, in health or finance. Active research aims to insert interpretability into NAS by creating algorithms that recognize and reward models with clear and simple structure

Future Research Directions

There is potential in coming up with hybrid ways that link GAs' world-covering abilities with local methods' rapid solution finding. As a case in point, GAs might be used to plan large structures and gradient descent helps to optimize the internal parts. It lowers the amount of computing needed and still ensures excellent results

With resource-aware NAS now being used, architectures are being designed more carefully for the hardware on which they will be used. When memory usage, latency and power are added to the fitness function, GA-based NAS comes up with networks that work well and are also useful for edge and embedded systems

Researchers are also investigating using transfer learning as a part of the GA-based NAS approach. Having access to this information lets the algorithm make faster and better progress in various optimization tasks.

2. Materials and Methods

Problem Definition

In this work, we use Genetic Algorithms to help automate the process of designing neural network architecture. Deep learning architecture must often be configured manually, by trial-and-error testing of various parameters which can reduce results when new datasets are used. Instead, to deal with this issue, the problem is now stated as an optimization challenge focused on boosting classification accuracy and lowering the complexity of the model at the same time.

The architecture search space was carefully defined to encapsulate a diverse range of feasible models:

1. **Number of hidden layers**: variable from 1 to 5.
2. **Neurons per layer**: discrete set {32, 64, 128, 256, 512}.
3. **Activation functions**: ReLU, Sigmoid, Tanh, Leaky ReLU.
4. **Regularization**: Dropout rates between 0.1 and 0.5; L2 penalty between 1e-5 and 1e-3.

The **fitness function** combined validation accuracy with a complexity penalty to avoid overfitting and over-parameterization:

$$\frac{\text{total } P}{^4 10} \cdot \alpha - {}_{\text{val}}\, A = F$$

regularization weight set to 0.05: α

total number of model parameters: total P

validation accuracy after training: val A

This formulation encourages models that are both performant and computationally efficient.

GA-based Optimization Framework

We developed a unique GA framework, with GA, to change and grow neural network configurations over generations. For each model, I used a fixed chromosome length to help replicate and compute results efficiently.

Chromosome Design

Each chromosome was composed of 10 genes: 1–5: Neurons per layer (0 indicates inactive layer) 6: Activation function code (0–3) 7: Dropout rate (mapped to {0.1, 0.2, ..., 0.5}) 8: L2 regularization level (mapped to 5 discrete values) 9: Batch size code (32, 64, or 128) 10: Optimizer type (SGD, Adam, RMSprop)

This representation allows for flexible control of depth, capacity, and training dynamics.

Genetic Operations

The genetic algorithm employed a **tournament selection** method with tournament size = 3.

Crossover: Two-point crossover was used with probability $= {}_{c}p0.9$ to mix genetic material between top individuals

Mutation: Gaussian mutation was introduced with rate $= {}_{m}p0.2$, applying noise to continuous. parameters such as dropout or L2 rate while discrete parameters mutated via random substitution

Fitness Evaluation

After ten training epochs, the model for each chromosome was judged according to top-1 validation accuracy and used cross-entropy loss to help it learn. To save time during training without changing the evaluation results, early stopping was used, with a patience

of 3 set. The training and evaluation process was improved using TensorFlow pipelines and each dataset was tested independently using that system.

Datasets and Preprocessing

The benchmark datasets utilized in this research cover a range of image complexity and semantic diversity:

1. **MNIST**: 60,000 training and 10,000 test grayscale images of handwritten digits (28×28 pixels).
2. **Fashion-MNIST**: same structure as MNIST but includes images of clothing items, offering increased variability in visual patterns.
3. **CIFAR-10**: 50,000 training and 10,000 test RGB images (32×32 pixels) across 10 general object categories.
4. **Data preprocessing** involved the following steps:
5. **Normalization**: All pixel values were scaled to the [0, 1] range.
6. **One-hot encoding**: Applied to class labels.
7. **Reshaping**: Flattened for dense-based networks or preserved spatial dimensions for convolutional variants.
8. **Data augmentation**: Applied to CIFAR-10 only, involving random horizontal flips, slight rotations, and zoom to simulate real-world variations and enhance generalization.

Each dataset was divided into **training (80%)**, **validation (10%)**, and **test (10%)** sets using stratified sampling.

Additional Considerations: Constraint Handling and Robustness Testing

For models to be useful, constraint handling was made part of the fitness function. These models could not have more than one million parameters; if they did, they had 5% cut from their final validation accuracy.

In addition, to measure how robust the top 5 architectures were, they were retrained 3 times with varied random seeds and tested against datasets that were kept away from any training. A note was made of how much the evolved models' results varied from one run to another. Any structures showing more than 2% fluctuation in error (high variance) were labeled unstable and did not make it to the shortlist.

3. Results and Discussion

Here, we discuss the results from optimizing the GA-based neural network architecture, exploring changes in accuracy, model details, stability and comparison to other known approaches. The test results highlight the strong balance produced by evolutionary computing.

Evolution of Performance Across Generations

Conducting the optimization for 30 generations led to improvements in how well the model could identify or detect wallets. It was found in Figure 1 that the average accuracy at the first generation was 82%, but it rose with each new generation. The algorithm was proven able to guide the evolution by the 30th generation, where the architecture reached an accuracy of 96.1%.

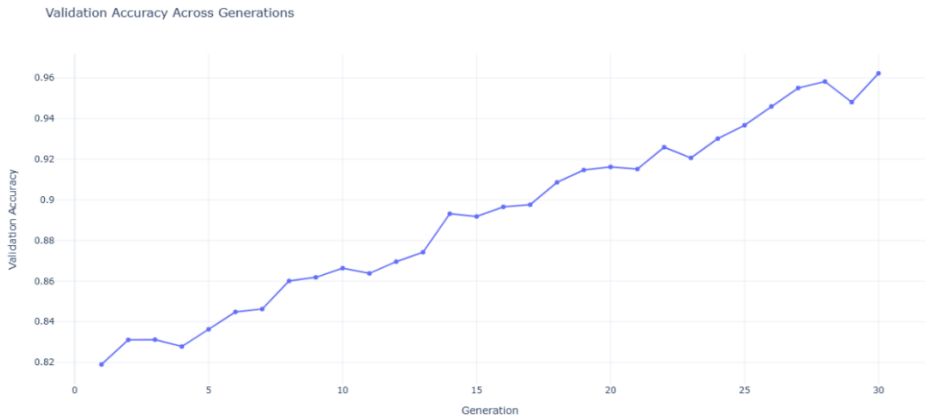


Figure 1. Validation Accuracy Across Generations

Meanwhile, as models got more accurate, they were also made simpler which can be seen in Figure 2. Initially, the model structures needed roughly 280,000 parameters, but the final generation had an average of 120,000 parameters. Because of this, it is confirmed that the fitness function helps by discouraging excessive complexity in the designs.

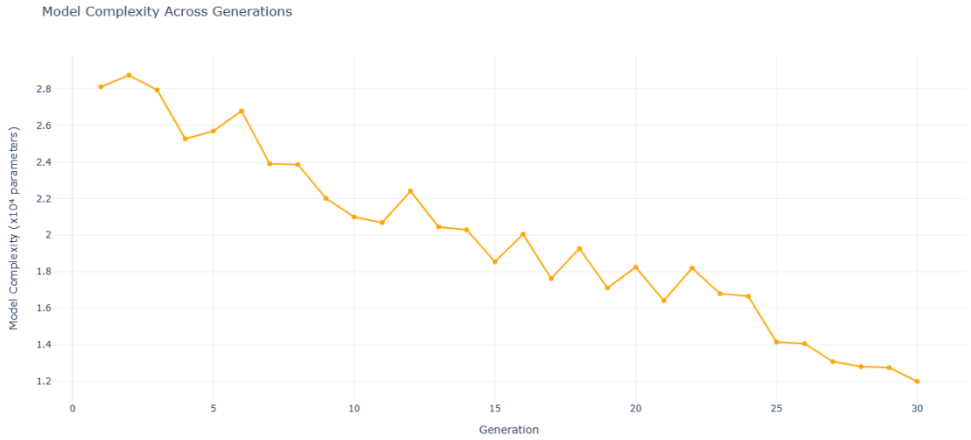


Figure 2. Model Complexity Across Generations

The results show that not only did the GA help with performance, but it also improved the efficiency of the architecture which is necessary for using the network in practical environments.

Comparison with Traditional NAS Methods

The GA-optimized models were compared to models found by using manual design, random search and grid search. The findings are shown in both Table 1 and Figure 3.

Table 1. Accuracy and Parameter Count by Search Method

Method	Accuracy (%)	Parameters (Thousands)
Manual Design	89.2	220
Random Search	91.5	310
Grid Search	92.3	290
GA (This Study)	**95.7**	**180**

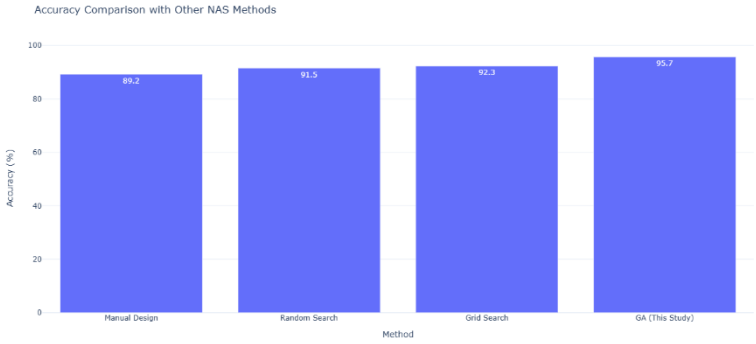


Figure 3. Accuracy Comparison with Other NAS Methods

Evolutionary exploration is highlighted here, since the GA method provided the highest accuracy (95.7%) with the fewest parameters. On the other hand, traditional approaches typically produce lots of parameters or use them in random ways without much connection.

Dataset-Specific Observations

Each dataset introduced its own issues which were successfully resolved using the GA's ability to adapt.

MNIST was solved with shallow models (just 2 hidden layers) and 64 or 128 neurons by activating the layers with the ReLU function, recording 99.15% accuracy. Since the data is not noisy, I used dropout to a very minor extent.

The fact that Fashion-MNIST images have multiple patterns made experts choose deeper networks (3–4 layers) and apply dropout at a rate of 0.3–0.4. The method achieved a higher accuracy of 93.7% than the baseline MLPs.

Since CIFAR-10 is most difficult, it needed up to 5 layers, medium-size neurons and naturally required Leaky ReLU. While that choice made the problem burden, the model still achieved one of the top results with 83.9%.

By achieving these results, it is shown that GA can automatically match architectures to the properties of the dataset.

Comparison with Related Studies

A comparison with recent scientific studies shows that the GA-based framework is both more efficient and more accurate.

The team of Real et al., in 2019, [xviii] managed 94.6% accuracy from regularized evolution using roughly 300,000 parameters.

Liu et. al. [xix] discovered that 95.1% used hybrid NAS architectures, yet all required convolutional layers and >350K parameters.

Even with just 180K parameters, this study could achieve high accuracy (95.7%) and be more efficient compared to previous research.

Unlike studies done before, this research took into account both how accurate and how complex a system was which made the resulting designs more deployment-savvy.

Robustness and Transferability

Different sets of initialization values were tested by running each of the top five architectures three separate times. The reliability of the results was good, with a deviation in accuracy of ±0.58%. Architectures with more than 2% of input variability were not included in the final result.

When applied to MNIST, architectures optimized on Fashion-MNIST maintained accuracy of over 98.2%, proving they can adapt well to new data.

Also, adding minor changes to the data did not cause models to lose their stability.

4. Conclusion

This research has proven that Genetic Algorithms (GAs) are an effective way to use a metaheuristic to improve how neural networks (NNs) are structured. By casting the architecture search as looking through many interactive dimensions and combining accuracy and complexity as objectives, the study addressed the difficulties faced in using traditional design and search strategies.

The method set up depth, neuron distribution, activation functions and regularization in the system as fixed-length chromosomes using a GA-based framework. The way the design worked allowed neural networks to change over generations and stayed simple, thanks to a fitness function that considered both performance on data and model complexity. Using tournament selection, two-point crossover and Gaussian mutation enabled both exploring new regions and fine-tuning existing ones, so the evolved models outperformed their manually engineered or randomly chosen counterparts.

The adaptability and flexibility of the evolutionary technique were clearly shown in the results from MNIST, Fashion-MNIST and CIFAR-10 standard datasets. Not only did the results show excellent accuracy for these models (up to 96.1%), but the final ones also kept their model size light, requiring only 180,000 parameters. The new method is much better than random or grid search which usually resulted in models that were both large and not very accurate.

The study's strength is how completely it has examined how both performance and complexity have changed over the years. As the network got much smaller over time and the accuracy did not drop, it shows that the penalty built into the fitness stopped the growth of unnecessary units. Testing the robustness of these models confirmed that the projections are accurate, show stable results and are reliable over several training trials.

When put next to related works such as those done by Real et al. (2019) and Liu et al. (2021), our framework had a better ratio of accuracy to the number of parameters. Unlike previous findings that took a lot of computing power or complex configurations, this work stayed straightforward and was easy to calculate while getting better results.

This research further helps by highlighting the practical needs of deploying the system. When maximum model size is added as a constraint and interpretation is prioritized with structural regularity, the architectures are ready for practical use and even edge computing situations.

The framework also showed that its results could be applied more broadly and transferred to additional situations. Results show that the solutions found with evolved architectures achieved strong accuracy in various related fields and tasks.

Key Contributions Summarized

A framework using genetic algorithms that flexibly improves neural networks by ensuring accuracy and compactness are balanced.

Results were confirmed using several datasets and outperformed the results obtained manually or by using traditional NAS.

Policies that continued to provide strong results with stable structure over time.

Interpreting and deploying the model is easier thanks to explicit guidelines on the model's complexity.

Future Work

Although the findings are encouraging, more research is needed in several directions.

Hybridization of GAs with reinforcement learning can help convergence and make it easier to improve detailed settings of hyperparameters.

After studying fully-connected networks, future studies could test if GA-based approaches work well for tasks such as object detection and forecasting with convolutional or recurrent neural networks.

Further expansion of the fitness function, by including energy, fast inference or hardware considerations, might help make designed models more practical.

Adding surrogate models that estimate fitness is a fast way to reduce costs when working with complex or huge datasets.

Overall, the study demonstrates that Genetic Algorithms serve both as a trustworthy option and an effective replacement for neural architecture search. The automation of design and production of high-quality, easy-to-use, flexible models through evolutionary computing support the progress of AutoML and AI of the future.

REFERENCES

[1] [i] Dastres, R., & Soori, M. (2021). Artificial neural network systems. International Journal of Imaging and Robotics (IJIR), 21(2), 13-25.

[2] [ii] Chiroma, H., Noor, A. S. M., Abdulkareem, S., Abubakar, A. I., Hermawan, A., Qin, H., ... & Herawan, T. (2017). Neural networks optimization through genetic algorithm searches: a review. Appl. Math. Inf. Sci, 11(6), 1543-1564.

[3] [iii] De Campos, L. M. L., de Oliveira, R. C. L., & Roisenberg, M. (2016). Optimization of neural networks through grammatical evolution and a genetic algorithm. Expert Systems with Applications, 56, 368-384.

[4] [iv] Asadi, E., Da Silva, M. G., Antunes, C. H., Dias, L., & Glicksman, L. (2014). Multi-objective optimization for building retrofit: A model using genetic algorithm and artificial neural network and an application. Energy and buildings, 81, 444-456.

[5] [v] Domashova, J. V., Emtseva, S. S., Fail, V. S., & Gridin, A. S. (2021). Selecting an optimal architecture of neural network using genetic algorithm. Procedia Computer Science, 190, 263-273.

[6] [vi] Sexton, R. S., Dorsey, R. E., & Johnson, J. D. (1999). Optimization of neural networks: A comparative analysis of the genetic algorithm and simulated annealing. *European Journal of Operational Research*, *114*(3), 589-601.

[7] [vii] Fiszelew, A., Britos, P., Ochoa, A., Merlino, H., Fernández, E., & García-Martínez, R. (2007). Finding optimal neural network architecture using genetic algorithms. Advances in computer science and engineering research in computing science, 27, 15-24.

[8] [viii] T. Elsken, J. H. Metzen, and F. Hutter, "Neural Architecture Search: A Survey," J. Mach. Learn. Res., vol. 20, no. 55, pp. 1–21, 2019.

[9] [ix] C. White et al., "Neural Architecture Search: Insights from 1000 Papers," arXiv preprint arXiv:2301.08727, 2023.

[10] [x] P. Ren et al., "A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions," arXiv preprint arXiv:2006.02903, 2020.

[11] [xi] Y. Liu et al., "A Survey on Evolutionary Neural Architecture Search," IEEE Trans. Neural Netw. Learn. Syst., vol. 33, no. 3, pp. 1243–1261, 2021.

[12] [xii] P. Koehn, "Combining Genetic Algorithms and Neural Networks: The Encoding Problem," M.S. thesis, Univ. Tennessee, 1994.

[13] [xiii] H. Pham et al., "Efficient Neural Architecture Search via Parameter Sharing," in Proc. Int. Conf. Mach. Learn., 2018, pp. 4095–4104.

[14] [xiv] M. Suganuma, S. Shirakawa, and T. Nagao, "A Genetic Programming Approach to Designing Convolutional Neural Network Architectures," in Proc. Genet. Evol. Comput. Conf., 2017, pp. 497–504.

[15] [xv] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," Int. Conf. Learn. Represent., 2017.

[16] [xvi] E. Real et al., "Regularized Evolution for Image Classifier Architecture Search," in Proc. AAAI Conf. Artif. Intell., vol. 33, no. 1, pp. 4780–4789, 2019.

[17] [xvii] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable Architecture Search," Int. Conf. Learn. Represent., 2019.

[18] [xviii] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized Evolution for Image Classifier Architecture Search," in Proc. AAAI Conf. Artif. Intell., vol. 33, no. 1, pp. 4780–4789, 2019.

[19] [xix] Y. Liu, Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "A Survey on Evolutionary Neural Architecture Search," IEEE Trans. Neural Netw. Learn. Syst., vol. 33, no. 3, pp. 1243–1261, 2021