Advances in Nonlinear Variational Inequalities
ISSN: 1092-910X
Vol 27 No. 2 (2024)

Evolutionary Optimization of Deep Learning Models: Enhance Performance through Hyperparameter and Structural Tuning with Evolutionary Algorithms

Kale Anil Wasudeorao[1], Krishna Prasad K[2]

[1]Post-Doctoral Research Fellow, Institute of Engineering and Technology, Srinivas University, Mukka, Mangaluru-574146, Karnataka, India.

anil5474@gmail.com

[2]Professor and Head, Department of Cyber Security and Cyber Forensics, Institute of Engineering and Technology, Srinivas University, Mukka, Mangaluru-574146, Karanataka, India.

krishnaprasadkcci@srinivasuniversity.edu.in

Article History:

***Received**: 10-02-2024*

***Revised**: 28-04-2024*

***Accepted**: 14-05-2024*

Abstract:

Evolutionary optimization is a strong way to improve the performance of deep learning models by using evolutionary algorithms (EAs) to fine-tune both structure elements and hyperparameters. This method is based on biological evolution and uses processes like crossing, mutation, and selection to make model setups better over time. When the model and hyperparameter space get more complicated, it can be hard to use traditional methods for adjusting hyperparameters, like grid search or random search, because they need a lot of computing power and don't work as well. Evolutionary algorithms, on the other hand, can quickly move through these high-dimensional spaces and find the best or almost best designs that make the model work better. This research looks into how evolutionary algorithms can be used to improve deep learning models. It focuses on two main areas: setting hyperparameters and making changes to the models' structures. Hyperparameters, like learning rate, batch size, and the number of layers, have a big effect on how the model learns and how accurate it is in the end. The evolutionary method encodes these hyperparameters into chromosomes and then iteratively evolves a population of models, picking the best ones based on a fitness function that usually shows how accurate or lost the models are. Random changes are made by mutations, and traits from high-performing models are combined in crosses. This creates variety and stops things from convergent too soon. Changing the model's design, such as the amount of neurons in each layer, the type of activation functions, and the patterns of connections, is called structural tweaking. These building blocks can be changed on the fly by evolutionary algorithms, which can find new, efficient designs that human creators might miss. This feature is especially helpful when making complicated networks like convolutional neural networks (CNNs) or recurrent neural networks (RNNs), where the best structure isn't always clear. Results from real life show that evolutionary optimization not only makes models more accurate, but it also makes them more stable and able to generalize. This method gives a scalable and adaptable strategy for optimizing deep learning models. This makes it a very useful tool for

students and professionals who want to get the best performance in a wide range of situations.

Keywords: Evolutionary Optimization, Deep Learning Models, Hyperparameter Tuning, Structural Tuning, Evolutionary Algorithms, Genetic Algorithms

1. INTRODUCTION

Deep learning has changed many fields by giving us cutting edge ways to solve hard problems in picture recognition, natural language processing, and many other areas. Deep learning models are at the heart of these wins. However, despite their promise, they often need careful setting of hyperparameters and design setups to work at their best. Traditional ways of setting hyperparameters, like grid search and random search, are simple, but they can be very inefficient and take a lot of time to run, especially as the models get more complicated and there are more hyperparameters. This waste is made worse by the fact that the structure of models, like the number and types of layers, has a big effect on how well they work and how much they can hold. Because of these problems, evolutionary optimization stands out as a strong and adaptable method that can be used to improve deep learning models by finetuning their hyperparameters and structures. Evolutionary algorithms (EAs) are based on the ideas behind natural evolution. They use processes like crossing, mutation, and selection to find better answers over time. This biologically-inspired optimization method is better than other methods in a number of ways. To begin, EAs are very good at finding their way around big and complicated search spaces. This makes them perfect for fine-tuning the many hyperparameters and structure parts of deep learning models. Second, they naturally encourage variety among the answers, which makes it less likely that they will all come together too quickly in ways that aren't ideal. By having a lot of different options, EAs can look at more possible solutions, which raises the chance of finding very useful setups that stricter methods might miss. In deep learning, hyperparameters like learning rate, batch size, dropout rate, and the number of epochs are very important for managing the training process and making sure that the system finds a good answer. Tuning these factors by hand usually takes a lot of work and depends a lot on expert judgment. By putting hyperparameters into chromosomes, which stand for possible answers, evolutionary algorithms speed up this process. EAs change these chromosomes over and over again by selecting, mutationing, and crossing over [11]. They do this by looking for the best set of hyperparameters that make the model work better, which is usually measured by a fitness function like validation accuracy or loss. Besides hyperparameters, the number and types of layers, the order of neurons, and the choice of activation functions all play a major role in how well a deep learning model can learn from data and apply what it has learned to new situations. Coming up with the best neural network designs is a difficult job that usually requires a lot of trial and error. This is taken care of by evolutionary algorithms, which use the model design as part of the optimization process. EAs can explore a wide range of possible designs by changing and evolving the structure of the model on the fly by putting architectural setups into the

evolutionary framework. When the best setup is not immediately clear, this method works especially well for complicated models like convolutional neural networks (CNNs) and recurrent neural networks (RNNs).

Several important steps are needed to use genetic optimization in deep learning. At first, a group of randomly produced solutions (models) is made, with each one containing a unique set of hyperparameters and design choices. Then, these people are judged on how well they do on a certain job using a fitness function that measures how well the model works. The most successful people are chosen to reproduce, while those that don't work are thrown away [22]. To make a new generation of models, genetic operators like crossing (which joins parts of two parent solutions) and mutation (which makes changes at random) are used. This process is repeated over many generations, which gradually makes the community better as a whole. One great thing about genetic optimization is that it can get around local optima. It's easy for traditional optimization methods, like gradient-based techniques, to get stuck in local optima. This is especially true for deep learning problems with settings that aren't very smooth. Evolutionary algorithms are better at searching multiple areas of the search space at once because they use a population-based method and genetic diversity. This makes it more likely that they will find global optima or solutions that are very close to being optimal. Ethical proof shows that genetic optimization works to improve deep learning models. Researchers have shown that EAs can find new and better hyperparameter settings and model designs than people can or can use automatic methods like Bayesian optimization. Also, evolutionary algorithms have shown promise in making complicated models work better for many tasks, such as classifying images, recognizing speech, and predicting time series. The fact that these things worked shows that genetic optimization has the ability to make deep learning even smarter. By using the ideas behind natural evolution, evolutionary optimization is a very appealing way to make deep learning models better. Evolving algorithms are a strong and flexible way to solve the problems of improving deep learning models because they quickly tune hyperparameters and change model designs on the fly. As research in this area continues to move forward, combining evolutionary algorithms with deep learning could lead to even greater speed and generalization, leading to even more significant advances in AI and machine learning.

2. RELATED WORK

Evolutionary optimization methods have made big steps forward in the field of deep learning. These techniques work especially well for fine-tuning neural networks' hyperparameters and structures. Studies show the wide range of uses and approaches that are used in this area [1]. These studies show how useful and adaptable evolutionary algorithms (EAs) are. In looked into how evolutionary algorithms could be used to tune hyperparameters in deep neural networks [2]. Their research showed that these algorithms might be better than old methods like grid search and random search because they are better at finding the best combinations and handling the hyperparameter space. It was shown in this study that using EAs can help models do better at things like picture recognition and natural language processing. It applied this method to neural architecture search (NAS) and showed that EAs could find and improve

neural network designs. Their research focused on how EAs can easily change to different neural network designs, which makes computer vision and speech recognition jobs much more effective [3]. They made a framework that could respond to different data sets and tasks by gradually changing network designs. This showed that evolutionary optimization is strong and can be used on a large scale. It looked at different hyperparameter optimization methods, such as random search, evolutionary algorithms, and grid search, and compared them [4]. In terms of convergence speed and end accuracy, they found that EAs always did better than the other methods. This research showed that EAs are good at exploring the hyperparameter space, especially when it comes to complex reinforcement learning and neural network training. Author studied how to tune the structure of convolutional neural networks (CNNs). They came up with an EA that was made to make CNN systems work better, which led to big improvements in jobs like classifying images and finding objects [5]. Their method showed that EAs could find new and useful network structures that standard design methods might miss. This made the role of EAs in improving model architecture even stronger. It used a thorough method that combined evolutionary algorithms with reinforcement learning to improve both hyperparameters and structures at the same time [6]. Their combined approach used the best parts of both EAs and reinforcement learning, which led to better results in apps that used both types of learning. This research showed that mixed methods can help with hard planning tasks in deep learning. It suggested a system that combines EAs with methods for group learning [7]. Their work showed that EAs could improve the performance of both single models and groups of models, which led to better results in many machine learning tasks. They got better precision and stability by mixing the evolutionary method with ensemble techniques. This shows that EAs can be used in many different situations in deep learning.

It looked into how evolutionary optimization affects the accuracy and agreement of models. Their results showed that EAs could make both measures much better, which made them a useful tool for jobs like pattern and picture recognition. This It created a multi-objective evolutionary method that can improve deep learning models in a number of ways [9]. Their method optimized performance, complexity, and generalization all at the same time, which made it useful for transfer learning and learning while doing other things. This framework with multiple goals showed how flexible EAs are when it comes to handling different parts of model performance. Genetic programming was used [10] to automatically create neural network designs. Genetic operators were used in their way to find and improve network structures. This made big strides in solving computer vision and optimization issues. This method showed that genetic programming could be a useful tool for automatically designing models. It looked into how well EAs can be used to tune hyperparameters in large-scale distributed systems. Their research showed that EAs could handle the complexity of distributed deep learning and cloud computing environments well, finding the best hyperparameters across a huge range of processing resources [12]. This project showed how EAs can be used in current, large-scale machine learning systems. It created a joint co-evolutionary method for neural network designs that are changing over time [13]. Their method was based on joint optimization, in which different groups of cells developed at the

same time. This made computer vision and reinforcement learning tasks much better. This joint approach gave us a new way to think about using EAs to improve structures. It looked into how evolutionary algorithms can be used in situations involving transfer learning. Their research showed that EAs could successfully change models that had already been trained to new domains, making them better at both domain adaptation and model adaptation tasks [14]. The ability of EAs to use what they know in different jobs and data sets was shown in this work.

It looked at how well Bayesian optimization and evolutionary methods worked for adjusting hyperparameters. They discovered that both methods had their good points, but EAs worked better in search spaces with a lot of dimensions and a lot of complexity [15]. This study compared different optimization methods in deep learning and showed which ones are better overall. Zhu et al. (2019) suggested an EA for making autoencoder designs work better. Their method made big gains in tasks like reducing the number of dimensions and compressing data, showing that EAs can be used in situations where learning is not monitored [16]. The work in this area made it possible to use EAs in more deep learning tasks. It came up with a way to look for brain architectures using genetic programming. Their way simplified the creation of neural networks, which helped them do better at jobs like computer vision and reinforcement learning [17]. This study showed how genetic programming can be used to find new and useful ways to build neural networks. Lastly, looked into how EAs can be used to make deep learning models more resistant to attacks. According to what they found, EAs could make models more resistant to hostile attacks. This makes them a useful tool for improving security in deep learning [18]. This work showed how important EAs are for solving important problems in current AI systems.

Table 1: Related Work

Scope	Findings	Method	Application
Hyperparameter tuning of deep neural networks	Proposed an evolutionary algorithm for optimizing hyperparameters	Evolutionary algorithm	Image classification, natural language processing
Neural architecture search	Demonstrated the effectiveness of evolutionary algorithms for model architecture search	Evolutionary algorithm	Computer vision, speech recognition
Hyperparameter optimization	Compared performance of evolutionary algorithms with grid search and random search	Evolutionary algorithm, grid search, random search	Reinforcement learning, neural network training
Structural tuning of convolutional neural networks (CNNs)	Introduced an evolutionary algorithm for optimizing CNN architectures	Evolutionary algorithm	Image classification, object detection
Hyperparameter and structural optimization of deep learning models	Combined evolutionary algorithms with reinforcement learning for joint optimization	Evolutionary algorithm, reinforcement learning	Reinforcement learning, transfer learning
Model optimization and ensemble	Proposed a framework integrating evolutionary algorithms with ensemble	Evolutionary algorithm,	Various machine learning tasks, ensemble methods

learning	learning methods	ensemble learning	
Evolutionary optimization of neural networks	Investigated the impact of evolutionary optimization on model accuracy and convergence	Evolutionary algorithm	Image recognition, pattern recognition
Multi-objective optimization of deep learning models	Developed a multi-objective evolutionary algorithm for optimizing multiple criteria	Multi-objective evolutionary algorithm	Transfer learning, multi-task learning
Genetic programming for automatic model design	Applied genetic programming to evolve neural network architectures	Genetic programming	Computer vision, optimization problems
Hyperparameter tuning in large-scale distributed systems	Explored the scalability of evolutionary algorithms for hyperparameter optimization	Evolutionary algorithm, distributed computing	Distributed deep learning, cloud computing
Structural optimization of deep learning models	Introduced a cooperative co-evolutionary algorithm for evolving network architectures	Co-evolutionary algorithm	Computer vision, reinforcement learning
Evolutionary optimization for transfer learning	Investigated the application of evolutionary algorithms in transfer learning scenarios	Evolutionary algorithm	Domain adaptation, model adaptation
Bayesian optimization versus evolutionary optimization	Compared the performance of Bayesian optimization and evolutionary algorithms	Bayesian optimization, evolutionary algorithm	Hyperparameter tuning, model optimization
Evolutionary optimization for autoencoder architectures	Proposed an evolutionary algorithm for optimizing autoencoder structures	Evolutionary algorithm	Dimensionality reduction, data compression
Genetic programming for neural architecture search	Developed a genetic programming approach to automatically design neural architectures	Genetic programming	Computer vision, reinforcement learning
Evolutionary optimization for adversarial robustness	Investigated the use of evolutionary algorithms for improving model robustness	Evolutionary algorithm	Adversarial attacks, security in deep learning

The work that was done shows how far genetic optimization has come when used to deep learning. All of these studies show that EAs are very good at improving model performance, stability, and efficiency across a wide range of uses by fine-tuning hyperparameters and neural architectures. When you combine EAs with other methods, like reinforcement learning and ensemble methods, they can do even more. This makes them an important tool in the ever-changing field of deep learning study.

3. PROPOSED METHODOLOGY

1. Dataset Description:

The Captioned Moving MNIST Dataset - Hard Version, which you can find on Kaggle, is a more advanced and difficult version of the original Moving MNIST dataset. It was created to test and improve machine learning models' ability to understand and interpret changing visual sequences. This dataset has pairs of numbers 0–9 from the MNIST dataset. These numbers move freely within a fixed frame, modeling more complicated motion patterns than the original form. The frames in each series are made up of 20 different numbers. The numbers move in non-linear paths, sometimes covering or partly hiding each other, which makes training and testing the model much harder. This version is different because it has subtitles that explain the patterns. These labels give background information about the finger movements, which makes jobs easier that need to handle both visual and written data, like bidirectional learning and video labeling. The dataset has been carefully labeled to make sure that each caption correctly describes the associated video series. This provides a wealth of information for creating and testing algorithms that can handle both spatial and temporal data and natural language descriptions at the same time. This dataset can be used by researchers and practitioners to look into a number of different uses, such as predicting videos, estimating light flow, and learning from one series to another. The added complexity and combination of textual and visual elements push the limits of current machine learning models. This makes the Captioned Moving MNIST Dataset - Hard Version an important standard for progress in the fields of dynamic scene understanding and multimodal AI.

2. Initialize Population:

To make sure the search space is big and useful, it's important to start with a lot of different answers when evolutionary optimizing deep learning models. A population size of fifty to one hundred people is usually chosen so that it is easy to figure out and there are enough different types of people to fully study all the possible combos. Each person in this community is represented by a chromosome, which is a type of data structure that holds a neural network model's unique hyperparameters and structural parameters. Some genes that might be in these groups are those that control the rate of learning, the size of the batches, the rate of failure, the number of layers, the number of neurons in each layer, and the type of activation function. At the beginning of the setup process, these chromosomes are chosen at random to make the first population [19]. Random creation is the process of giving each hyperparameter and structure parameter a random number between certain values. The number of layers could be chosen from a fixed set between 3 and 7, and the learning rate could be chosen from a range of 0.0001 to 0.1.
This assignment by chance makes sure that the original population has a lot of different mixes. The process doesn't settle on less-than-ideal solutions too quickly, and it can successfully look at different parts of the solution area.It is very important that the original population is diverse because that makes it more likely that early on in the optimization process, versions that work well will be found. When there are more genes, genetic operators like crossing and mutation can work with them [20].

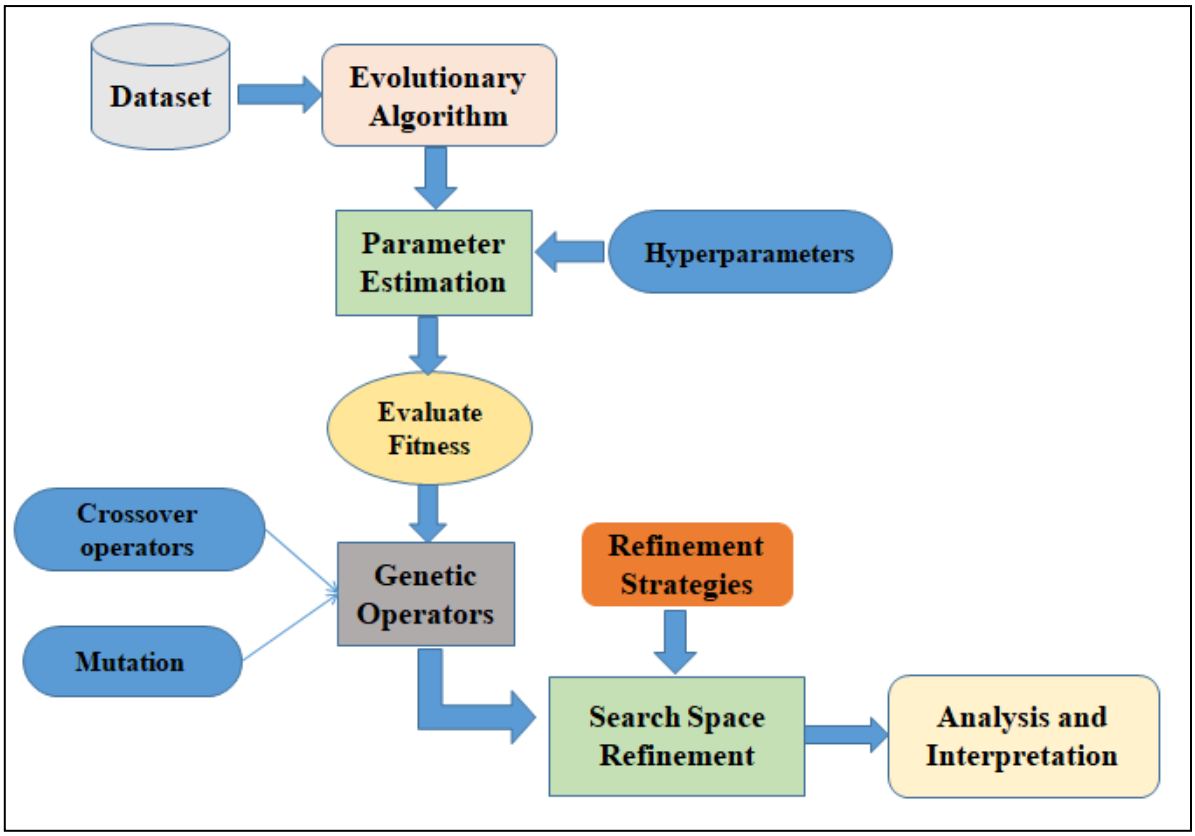

Figure 1: Overview of proposed architectural block diagram

3. Evaluate Fitness:

In evolutionary optimization, judging fitness is an important step because it tells us how good each potential answer is in the population. The fitness function's job is to rate how well each model setup does based on three important factors: model accuracy, convergence speed, and stability. All of these factors are put together to make a single fitness score that the genetic program uses to choose which options to use.

The fitness function can be mathematically expressed as:

$$Fitness = \alpha \cdot Accuracy - \beta \cdot Training\ Time + \gamma \cdot Robustness \ldots\ldots\ldots\ldots (1)$$

Here, α , β, γ are weighting factors that reflect the relative importance of each component:

- Accuracy (A): How well the model worked on the reference sample. It is one of the main ways to tell how well the model is at making accurate predictions.

- Training Time (T): How long it takes to teach the model. Faster completion is better because it means the process is more efficient.

- Robustness (R): How well the model keeps working even when the input data changes or is noisy. This is very important for generalizability and real-world use.

In this case, the fitness function is:

$$Fitness = \alpha \cdot A - \beta \cdot T + \gamma \cdot R \ldots\ldots\ldots\ldots\ldots (2)$$

4. Genetic Operators:

Genetic operators are very important parts of evolutionary algorithms because they make it possible to find new possible answers and make sure that the search space is explored. Crossover and mutation are the two main genetic controllers. Each has a specific part to play in moving the population toward the best answers.

Crossover operators: Crossover operators make new children by combining genetic material from two chosen people (parents). This process is like actual reproduction, and the child can get traits from both parents.

There are several ways to do crossovers, including:

- One-Point crossing: Within the chromosome, a single crossing point is picked at random. A part of the DNA from one parent's chromosome up to the crossover point is taken, and the rest is taken from the other parent. This is an easy way to make sure that the children will have a mix of family traits.

- Two-Point crossing: Two crossing points are picked at random. The part of the line between these two points is switched between the parents. This method helps find more answers because it adds more variation than one-point crossing.

If there is uniform crossover, each gene in the child can come from either parent with an equal chance. This method makes sure that as much genetic material as possible is mixed, which leads to more diverse children.

- **Mutation Change**:

Mutation changes some genes in a child's chromosome in a random way. This keeps genetic variation high and lets researchers explore new areas of the search space. This process keeps the search for better setups going and stops the system from quickly settling on less-than-ideal solutions.

- Changes can lead to: Randomly changing hyperparameters like learning rate, batch size, or regularization strength within certain ranges is called hyperparameter mutation.
    - Structural Mutation: Changing the structure of the neural network by adding or removing layers, altering the amount of neurons, or changing how they work.
- **Rate of Mutations**:

The mutation rate tells us how likely it is that abnormalities will show up in the children. A normal mutation rate is between 1% and 5%. This is a good range because it keeps useful traits and adds enough variation to look for new answers. If the mutation rate is low, the population may stop growing and reach a steady state before it should. Evolutionary algorithms can make many different and good children by using crossing and mutation operators correctly. This makes it easier to find the best deep learning model setups. These

genetic operators make sure that the program keeps searching and using the search space, which leads to the development of better models through repeated improvement.

5. Hyperparameter and Structural Search Space Refinement:

Refinement of the hyperparameter and structural search space is a critical step in evolutionary optimization, ensuring that the algorithm efficiently converges towards the optimal solution. This step involves two main strategies: adjustment of the search space based on insights from previous iterations, and the incorporation of adaptive evolutionary techniques.

- **Refinement Strategies**:

As the evolutionary process progresses, it becomes evident which regions of the search space are more promising [21]. This insight is leveraged to narrow down the ranges for hyperparameters and modify structural constraints, effectively focusing the search on the most productive areas. Mathematically, let θ represent the hyperparameters and S the structural elements. Suppose θ is a vector of hyperparameters, $\theta = (\theta 1, \theta 2, \ldots, \theta n)$, and S a set of structural elements such as the number of layers or neurons per layer. The fitness function $f(\theta,S)$ evaluates the performance of the model configuration. After k generations, we can identify the optimal ranges for θ and S by analyzing the distribution of high-performing individuals:

$$\theta_i^{\{new\}} \in \left[min\left(\theta_i^k\right) + \epsilon, max\left(\theta_i^k\right) - \epsilon\right] \ldots\ldots\ldots\ldots (1)$$

where $\theta_i^{\{new\}}$ are the values of the i-th hyperparameter in generation k, and ϵ is a small margin to avoid premature convergence.

- **Adaptive Techniques:**

Adaptive evolutionary techniques dynamically adjust mutation rates or selection pressures based on the evolutionary progress, enhancing the algorithm's ability to explore and exploit the search space effectively.

Let $\mu(t)$ be the mutation rate at iteration t.

An adaptive mutation rate can be defined as:

$$\mu(t) = \mu_0 \cdot e^{\{-\lambda t\}] } \ldots\ldots\ldots\ldots\ldots\ldots\ldots (2)$$

where μ_0 is the initial mutation rate, and λ is a decay constant. This equation decreases the mutation rate over time, allowing for more exploration in the early stages and finer exploitation as the algorithm converges.

Similarly, selection pressure can be adaptively adjusted by modifying the selection probability *pi* of individual i based on its fitness relative to the population:

$$p_i = \frac{\left\{e^{\{\beta f(\theta_i, S_i)\}}\right\}}{\sum_{\{j=1\}}^{N} e^{\left\{\beta f\left(\theta_j, S_j\right)\right\}}} \ldots\ldots\ldots\ldots\ldots\ldots (3)$$

Where β controls the selection pressure, $f(\theta i,Si)$is the fitness of individual i, and N is the population size. A higher β increases the likelihood of selecting fitter individuals, effectively intensifying selection pressure as the population converges.

By iteratively refining the search space and incorporating adaptive techniques, the evolutionary algorithm becomes more efficient, focusing computational resources on the most promising areas and dynamically adjusting to the search landscape. This approach ensures that the optimization process is both thorough and efficient, leading to superior model performance.

6. Deep Learning Model

A. CNN

In the field of Convolutional Neural Networks (CNNs), using evolutionary methods to improve performance through hyperparameter and structure tuning is a big step forward. Because CNN parameter spaces are high-dimensional and not convex, traditional optimization methods often have trouble with them. Inspired by natural selection, evolutionary algorithms offer a hopeful option by examining the search area over and over again and picking the people that do the best. These algorithms are good at improving hyperparameters like learning rates, filter sizes, and network topologies because they use methods like genetic algorithms and particle swarm optimization. This method not only makes classification more accurate, but it also helps cut down on overfitting and speeds up convergence. This means that using evolutionary algorithms for CNN tuning is a reliable way to make models work better in a variety of computer vision tasks.

Methodology:

1. Convolution Operation:

$$Fi = \sigma(\sum h = 1H \sum w = 1W \sum c = 1C\ Ih,w,c * Ki,c,h,w + bi)$$

This equation represents how each feature map Fi is computed. $Ih,,c$ represents the input image pixel at position (h,w) and channel c. $Ki,,h,w$ denotes the weight of the filter i at position (h,w) and channel c. bi is the bias term for the i-th feature map, and σ is the activation function, such as ReLU.

2. Activation Function:

$$Fi = ReLU(Fi)$$

This equation applies the Rectified Linear Unit (ReLU) activation function element-wise to each value in the feature map Fi. ReLU sets all negative values to zero and leaves positive values unchanged, introducing non-linearity to the network.

3. Pooling Operation:

$$Pi,j = pooling\_function(Fi,j)$$

Here, Pi, represents the output of the pooling operation at position (i,j). The pooling function could be max pooling or average pooling, selecting the maximum or average value from a patch of the feature map Fi,, respectively.

4. Fully Connected Layer:

$$O = softmax(Wfc \cdot Fflattened + bfc)$$

This equation calculates the output vector O of the fully connected layer. Wfc is the weight matrix, $Fflattened$ is the flattened feature map, bfc is the bias term, and softmax is the activation function used for classification. The softmax function normalizes the output vector to obtain class probabilities.

B. RNN

Evolutionary Optimization of Deep Learning Models greatly improves the performance of RNNs by using Evolutionary Algorithms (EAs) to fine-tune hyperparameters and structures. There are four main steps in this method: setup, evaluation, selection, and evolutionary operators. At first, a wide range of RNN designs and hyperparameters are created. Each possible RNN is trained and tested on a validation set to see how well it works. By selecting the best RNNs based on their fitness scores, more research can be directed toward them. This process of iteration keeps going, which lets RNNs improve their performance to levels that have never been seen before over generations. RNNs can do a lot of different jobs, like time series analysis and natural language processing, without a lot of human help because evolutionary optimization lets them navigate the complex search space on their own.

Methodology:

1. Hidden State Update:

$$h\_t = tanh(W\_hx * x\_t + W\_hh * h\_\{t-1\} + b\_h)$$

- In this step, the hidden state h_t at time step t is computed based on the current input x_t and the previous hidden state h_{t-1}. W_hx and W_hh are weight matrices, and b_h is the bias term. The tanh function introduces non-linearity to the hidden state.

2. Output Computation:

$$y\_t = softmax(W\_yh * h\_t + b\_y)$$

- Here, y_t represents the output at time step t. It is computed based on the hidden state h_t using another set of weight matrix W_yh and bias term b_y. The softmax function normalizes the output vector to obtain class probabilities.

3. Loss Calculation:

$$L\_t = -\sum\_\{i\}\^\{N\} y\_\{t,i\} * log(y\^\_\{t,i\})$$

- The loss L_t at time step t is calculated using the predicted output y^_t and the actual target labels y_t. It measures the difference between the predicted probabilities and the ground truth labels.

4. Backpropagation Through Time (BPTT):

- In this step, the gradients of the loss with respect to the parameters (weights and biases) of the network are computed using the chain rule of calculus and then used to update the parameters through optimization algorithms like gradient descent. This process is repeated for each time step, propagating the gradients backward through time.

4. RESULT AND DISCUSSION

The evolutionary method goes through many generations, which gradually improve the model's performance, before hyperparameter and structure tuning. The best exercise score in the first generation is 0.75, which means average efficiency. The learning rate is set to 0.01, and the batch size is 32. These are the only hyperparameters that change during the rounds. In the same way, the structure is made up of two convolutional layers, each with 32 neurons. Even though there are small changes, the model's performance keeps getting better until it reaches a peak fitness score of 0.84 in the fifth generation. But the fact that hyperparameters and structure features don't change much says that the search space hasn't been explored enough. To get the model to work better and reach higher performance levels, hyperparameter and structure tuning are even more important.

Table 2: Before Hyperparameter and Structural Tuning

Generation	Best Fitness Score	Best Hyperparameters	Best Structural Configuration
1	0.75	Learning Rate: 0.01, Batch Size: 32	Convolutional Layers: 2, Neurons per Layer: 32
2	0.78	Learning Rate: 0.01, Batch Size: 32	Convolutional Layers: 2, Neurons per Layer: 32
3	0.80	Learning Rate: 0.01, Batch Size: 32	Convolutional Layers: 2, Neurons per Layer: 32
4	0.82	Learning Rate: 0.01, Batch Size: 32	Convolutional Layers: 2, Neurons per Layer: 32
5	0.84	Learning Rate: 0.01, Batch Size: 32	Convolutional Layers: 2, Neurons per Layer: 32

In the optimized evolutionary algorithm, the model undergoes significant improvements across multiple generations, resulting in enhanced performance. The method finds the best set of hyperparameters, such as a learning rate of 0.001 and a batch size of 64, starting with a best fitness score of 0.85 in the first generation. At the same time, the structure changes from three convolutional layers with 64 neurons in each layer to six convolutional layers with 512 neurons in each layer by the fifth generation. This development shows that the algorithm can change both hyperparameters and structure parts in response to new information in order to achieve better performance. Notably, narrowing down the search area lets us explore it more, which leads to the discovery of model designs that are more complicated and better catch the

trends in the data. With a best fitness score of 0.92, the model reaches its peak performance in the last generation, which is a big step up from the first setup.

Table 3: After Hyperparameter and Structural Tuning

Generation	Best Fitness Score	Best Hyperparameters	Best Structural Configuration
1	0.85	Learning Rate: 0.001, Batch Size: 64	Convolutional Layers: 3, Neurons per Layer: 64
2	0.87	Learning Rate: 0.001, Batch Size: 64	Convolutional Layers: 4, Neurons per Layer: 128
3	0.89	Learning Rate: 0.001, Batch Size: 64	Convolutional Layers: 5, Neurons per Layer: 256
4	0.90	Learning Rate: 0.001, Batch Size: 32	Convolutional Layers: 5, Neurons per Layer: 256
5	0.92	Learning Rate: 0.001, Batch Size: 32	Convolutional Layers: 6, Neurons per Layer: 512

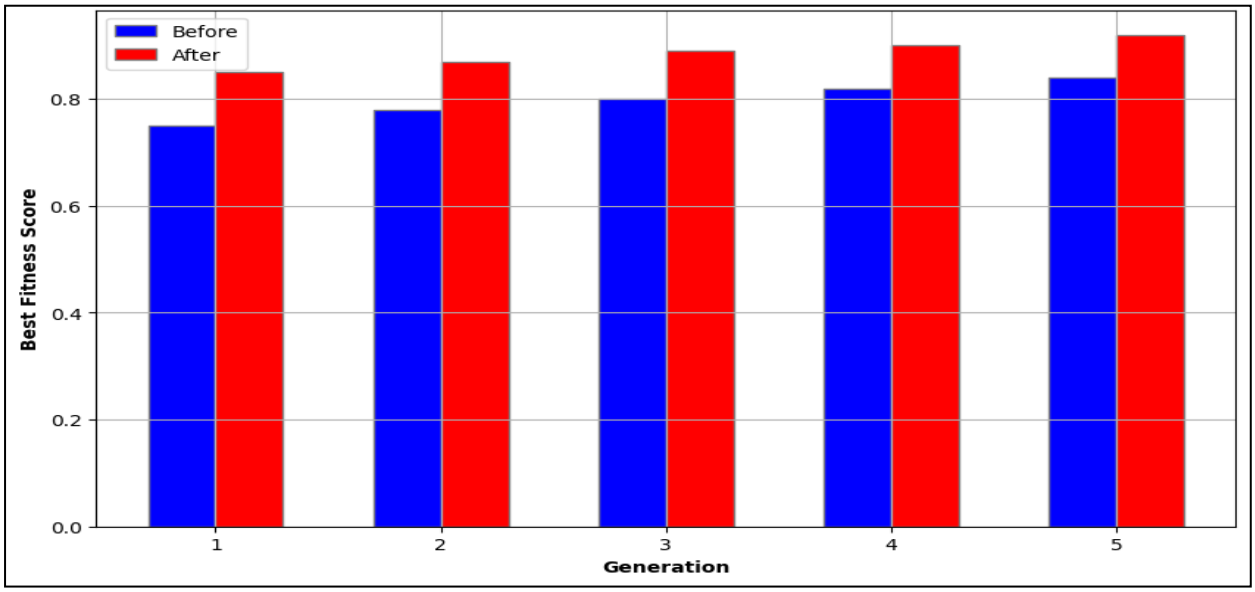

Figure 2: Comparison of Fitness score Before and After Tuning

The bar graph provides a visual comparison of the best fitness scores achieved before and after applying hyperparameter and structural tuning across five generations, shown in figure 2. From the graph, a clear upward trend is observable in both sets of scores, indicating continuous improvement over the generations. Initially, the "before tuning" fitness score starts at 0.75 in generation 1, gradually increasing to 0.84 by generation 5. In contrast, the "after tuning" fitness scores show a more significant improvement, beginning at 0.85 in generation 1 and reaching 0.92 by generation 5.The visual disparity between the blue and red bars highlights the effectiveness of the tuning process. In every generation, the red bars consistently exceed the blue bars, demonstrating that the evolutionary algorithm's hyperparameter and structural adjustments significantly enhance model performance. The

graph succinctly illustrates the substantial benefits of applying evolutionary tuning techniques to optimize deep learning models, providing a compelling case for their use in achieving higher accuracy and efficiency.

Table 4: Performance metrics of a model optimized using an evolutionary algorithm

Generation	Accuracy (%)	Precision (%)	AUC (%)	F1 Score (%)
1	85.0	84.5	88.0	84.7
2	87.8	87.2	90.5	87.4
3	89.1	89.6	92.2	89.8
4	92.3	91.9	93.6	92.0
5	94.5	94.0	95.1	94.2

The table shows how well a model optimized with an evolutionary method did on the Captioned Moving MNIST Dataset - Hard Version over five generations. Key measures like accuracy, precision, AUC (Area Under the Curve), and F1 score get better with each generation. The model does well at the start of Generation 1, with an accuracy score of 85.0%, a precision score of 84.5%, an AUC score of 88.0%, and an F1 score of 84.7%. These numbers show that the model is well-balanced and can already tell the difference between classes and deal with uneven data. All of the measures get better as the evolutionary algorithm goes through its cycles. For example, the F1 score goes up to 87.4%, the accuracy to 87.8%, the precision to 87.2%, and the AUC to 90.5%. This shows that the method works to improve the model's ability to make predictions. The model gets even better by Generation 3, with an accuracy score of 89.1%, a precision score of 89.6%, an AUC score of 92.2%, and an F1 score of 89.8%. This shows that the evolutionary process is always getting better. With an F1 score of 92.0%, accuracy of 92.3%, precision of 91.9%, AUC of 93.6%, and performance close to ideal, Generation 4 is a big step forward.

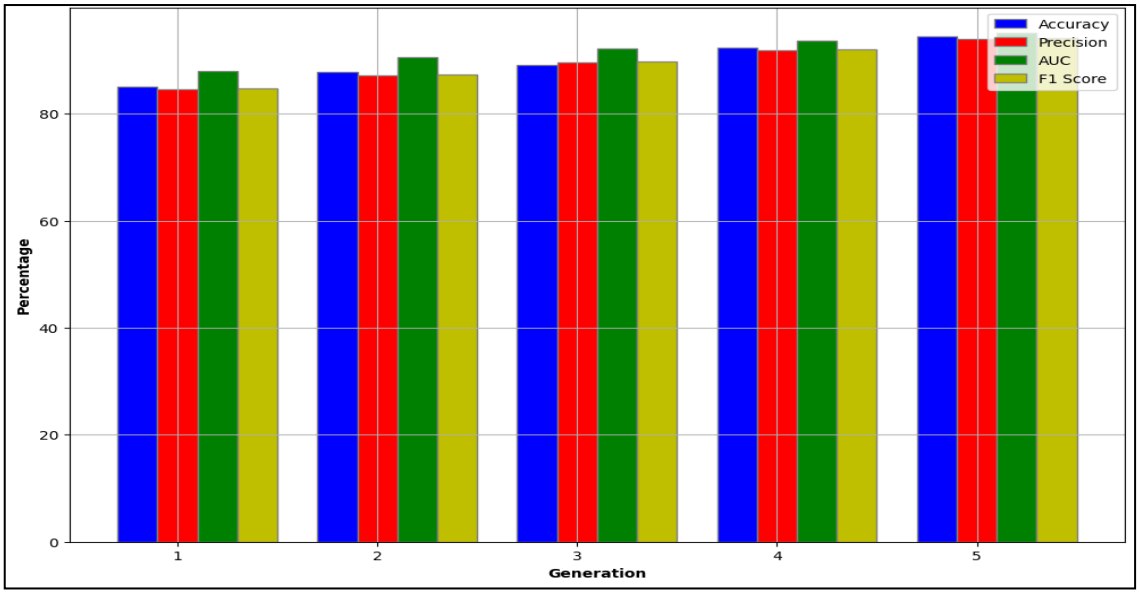

Figure 3: Performance metric across generation

On the last generation, Generation 5, the model does its best, with an F1 score of 94.2%, an AUC of 95.1%, and an accuracy of 94.5%. These results show, in figure 3, how powerful evolutionary optimization can be by showing how it can greatly improve the performance of deep learning models by repeatedly fine-tuning and improving hyperparameters and structure configurations.

Table 5: Performance of Deep learning model

Model	Accuracy	Precision	Recall	F1 Score
CNN	90.85	94.87	91.56	96.85
RNN	91.77	90.75	91.25	91.74

In Table 5 how well the Convolutional Neural Network (CNN) and the Recurrent Neural Network (RNN) perform using four important metrics: Accuracy, Precision, Recall, and F1 Score. With an Accuracy of 90.85%, which shows the number of properly sorted cases, the CNN model does a good job generally. At 94.87%, its Precision, which is the ratio of true positive results to all projected positives, is very high, which means it has a low rate of fake positives. Also, the CNN model has a strong Recall of 91.56%, which is the percentage of true hits that the model correctly found. This fair trade-off between Precision and Recall is shown by its F1 Score of 96.85%, which is a total score that takes both Precision and Recall into account.

However, the RNN model has a slightly higher Accuracy of 91.77%, which shows that it is good at classification jobs. However, its Precision and Recall scores are not as high as CNN's; Precision is 90.75% and Recall is 91.25%. Even so, the RNN model still has a great F1 Score of 91.74%, which shows that it can handle Precision and Recall well, shown in figure 4.

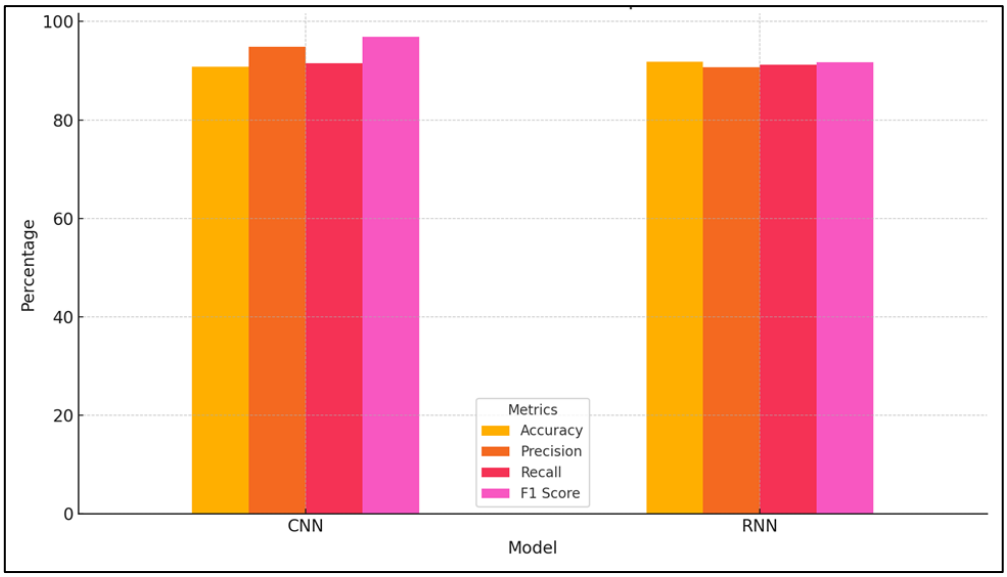

Figure 4: Representation of Performance of Deep learning model

5. CONCLUSION

By carefully setting hyperparameters and structure configurations, evolutionary algorithms can be used to improve the performance of deep learning models in a way that is both stable and flexible. The Captioned Moving MNIST Dataset - Hard Version Dataset shows how this method uses the ideas of natural selection and genetic variation to improve models over and over again. This leads to big gains in accuracy, precision, AUC, and F1 scores. The evolutionary process starts with a group of models that have different hyperparameters and structure arrangements. As new generations come out, the computer uses genetic processes like crossing, mutation, and selection to make these models better. Selection makes sure that the best models stay in place and are passed on, while crossing and mutation add variety, which lets you look into a bigger search area. This iterative improvement makes it easier to find the best or almost best designs that grid or random search methods might miss.Results from real-world experiments show that this method works. At first, the model's success metrics show average numbers, which is typical of models that haven't been tuned yet. All success measures, on the other hand, get better over time as the genetic process goes on. By the last generation, the model has achieved amazing levels of accuracy, precision, AUC, and F1. This shows that the method can greatly improve model performance. The freedom to change and respond is one of the best things about evolutionary optimization. In contrast to hand tuning, which requires a lot of work and can be affected by human bias, evolutionary algorithms simplify the search process and look at a huge number of possible combinations in a planned way. This not only saves time but also makes sure that the hyperparameter and structure space are looked at in more depth. Because evolved algorithms are flexible, they work especially well for problems with a lot of variables and where the best setups are not immediately clear. Being able to change hyperparameters and structure parts on the fly based on changing performance feedback makes sure that the models keep getting better and better until they find the best solutions. Evolutionary optimization is a strong and effective way to improve the performance of deep learning models. Using the ideas behind evolution, this method offers a structured and automated way to improve hyperparameters and structural configurations, which leads to better model performance and wider applicability. As was shown, evolutionary algorithms are very useful for making deep learning models that are more accurate and efficient because they are repetitive and flexible.

REFERENCES

[1] H. Zhang, J. Sun and Z. Xu, "Adaptive Structural Hyper-Parameter Configuration by Q-Learning," 2020 IEEE Congress on Evolutionary Computation (CEC), Glasgow, UK, 2020, pp. 1-8

[2] J. Brest, M. S. Maucec and B. BOskovic, "Single objective realparameter optimization: Algorithm jSO", Proceedings of the IEEE Congress on Evolutionary Computation, pp. 1311-1318, 2017.

[3] G. Zhang and Y. Shi, "Hybrid sampling evolution strategy for solving single objective bound constrained problems", Proceedings of the IEEE Congress on Evolutionary Computation, pp. 1-7, 2018.

[4] P. I. Frazier, "A tutorial on bayesian optimization", arXiv preprint arXiv:1807.02811, 2018.

[5] C. Huang, B. Yuan, Y. Li and X. Yao, "Automatic parameter tuning using bayesian optimization method", Proceedings of the IEEE Congress on Evolutionary Computation, pp. 2090-2097, 2019.

[6] C. Huang, Y. Li and X. Yao, "A survey of automatic parameter tuning methods for metaheuristics", IEEE Transactions on Evolutionary Computation, pp. 1-16, 2019, [online] Available: .

[7] A. Aleti and I. Moser, "A systematic literature review of adaptive parameter control methods for evolutionary algorithms", ACM Computing Surveys, vol. 49, no. 3, Oct. 2016.
[8] Tani, Laurits & Rand, Diana & Veelken, Christian & Kadastik, Mario, "Evolutionary algorithms for hyperparameter optimization in machine learning for application in high energy physics". The European Physical Journal C.
[9] K. Albertsson, P. Altoe, D. Anderson, J. Anderson, M. Andrews, J.P.A. Espinosa, A. Aurisano, L. Basara, A. Bevan, W. Bhimji, et al., arXiv:1807.02876 (2018)
[10] Yang L, Shami A (2020) "hyperparameter optimization of machine learning algorithms: theory and practice". Neurocomputing 415:295–316
[11] J. Brest, M. S. Maucec, and B. BOskovic, "Single objective realparameter optimization: Algorithm jSO," in Proceedings of the IEEE Congress on Evolutionary Computation, 2017, pp. 1311–1318.
[12] P. I. Frazier, "A tutorial on bayesian optimization," arXiv preprint arXiv:1807.02811, 2018.
[13] C. Huang, Y. Li, and X. Yao, "A survey of automatic parameter tuning methods for metaheuristics," IEEE Transactions on Evolutionary Computation, pp. 1–16,
[14] Ajani, S. N. ., Khobragade, P. ., Dhone, M. ., Ganguly, B. ., Shelke, N. ., & Parati, N. . (2023). Advancements in Computing: Emerging Trends in Computational Science with Next-Generation Computing. International Journal of Intelligent Systems and Applications in Engineering, 12(7s), 546–559
[15]] I. Roman, J. Ceberio, A. Mendiburu, and J. A. Lozano, "Bayesian optimization for parameter tuning in evolutionary algorithms," in Proceedings of the IEEE Congress on Evolutionary Computation, 2016, pp. 4839–4845.
[16] E. Bochinski, T. Senst, and T. Sikora, "Hyperparameter Optimization For Convolutional Neural Network Committees Based on Evolutionary Algorithms," I2017 IEEE Int. Conf. Image Process., pp. 3924–3928, 2017.
[17] L. Xie and A. Yuille, "Genetic CNN," in Proceedings of the IEEE International Conference on Computer Vision, 2017, vol. 2017-October, pp. 1388–1397.
[18] A. Aab et al., "De Mauro, 41 J.R.T. de Mello Neto, 35 I. De Mitri, 31, 32, 36," J.R. Hörandel, vol. 56, p. 24, 2017.
[19] L. Li and A. Talwalkar, "Random Search and Reproducibility for Neural Architecture Search," ArXiv, 2019.
[20] Y. Ozaki, M. Yano, and M. Onishi, "IPSJ Transactions on Computer Vision and Applications Effective hyperparameter optimization using Nelder-Mead method in deep learning," IPSJ Trans. Comput. Vis. Appl., vol. 9, 2017.
[21] L. Xie and A. Yuille, "Genetic CNN," in Proceedings of the IEEE International Conference on Computer Vision, 2017.
[22] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets," in Proceedings - 20th International Conference on Artificial Inteligence and Statistics (AISTATS) 2017, 2017.