NEAT-PSO: A Hybrid Approach to Evolving Convolutional Neural Network Architectures

Fall 2024 CS394N Final Report

Benson Ngai
UT CS Department
benson.ngai@utexas.edu

Brian Kim
UT ECE Department
briankim31415@gmail.com

***Abstract*—This paper explores the integration of Particle Swarm Optimization (PSO) with NeuroEvolution of Augmenting Topologies (NEAT) to enhance the evolution of neural network architectures. NEAT evolves both network topologies and weights, while PSO optimizes the NEAT evolution hyperparameters. The proposed hybrid approach leverages NEAT's capability to generate complex, adaptive structures and PSO's fast convergence on continuous optimization tasks. Experiments demonstrate that the NEAT-PSO framework can optimize NEAT performance on CIFAR-10 image classification tasks, providing a novel method for evolving neural networks with higher efficiency and adaptability.**

I. Introduction

Neural networks have revolutionized various industries, playing a pivotal role in advancing fields such as computer vision, natural language processing, and robotics. These models range from simple feedforward architectures to vast, intricate systems requiring meticulous design and optimization. Interestingly, deeper or more complex networks do not always outperform their shallower counterparts. For example, the GoogLeNet architecture not only halved the error rate from the previous year's best models but also achieved this with a significantly reduced parameter count [1]. This underscores an essential question in deep learning: What is the most optimal network architecture?

To address this question, researchers often turn to Neural Architecture Search (NAS). NAS involves automating the process of discovering the best-performing neural network architecture for a given task. However, this is a challenging problem due to the exponential growth of the search space with each additional parameter. Effectively navigating this space demands innovative algorithms that balance exploration and exploitation. As such, Evolutionary Computation (EC) based NAS approaches (ENAS) have been taken [2]. ENAS is a population-based computational paradigm that models each individual model architecture as an organism and simulates their evolution using evolutionary operators such as mutations and crossover, and evaluates their fitness through a task-specific metric. Their fitness, along with an organism's genetic diversity, is used to determine whether an organism survives to the next iteration. This paradigm encourages a balanced tradeoff between searching for elite, high-performing architectures as well as novel, diverse architectures. After evolving for some number of iterations, the highest-fitness model architecture individual is evaluated on realistic data to determine the effectiveness of the model in the task it's trained on.

In this project, we propose a novel hybrid ENAS framework that combines two existing approaches: NeuroEvolution of Augmenting Topologies (NEAT) and Particle Swarm Optimization (PSO). NEAT has been shown to be capable of evolving the topology (and weights) of neural networks, which can lead to the exploration of diverse-topology architectures, while PSO has been shown to be capable of fine-tune hyperparameter optimization. As such, these methods approach NAS from different perspectives, and by leveraging their complementary strengths, we aim to develop a framework capable of efficiently generating high-fitness and low model complexity neural network architectures.

Our main goals are as follows:

- **NEAT-CNN:** To explore an approach for applying NEAT on convolutional neural networks (CNNs), investigating its ability to evolve architectures that can achieve reasonable performance on the CIFAR-10 [3] image classification task.
- **NEAT-PSO Integration:** To integrate NEAT-evolved CNN architectures with PSO, using the latter to fine-tune their hyperparameters. This hybrid framework aims to combine topology evolution with hyperparameter optimization, seeking a synergistic improvement in performance and efficiency.

II. Related Work

A. NEAT

NEAT [4] is a neuroevolution algorithm that evolves neural networks by optimizing both their topologies and weights (TWEANN). Unlike conventional approaches that require a fixed network structure, NEAT starts with simple networks, typically consisting of only input and output nodes, and gradually evolves them by introducing new nodes and connections.

The algorithm begins with designing a genetic encoding, known as a genome (genotype), that represents a particular individual in the population. The genome contains all the necessary information to construct the neural network architecture topology, including the weights, which is represented as the phenotype. The original paper involves creating an encoding for simple ANNs; these networks largely consist of perceptron nodes and weighted edges. As such, the genome for a single individual consisted of Node Genes and Connection Genes, as shown in Figure 1. Each node is either an input node, hidden node, or output node. Connection genes specify which nodes the edge connected, as well as their weight and a global innovation number, which are used to track the history of genes over evolution and helpful in comparing the similarity between genomes.

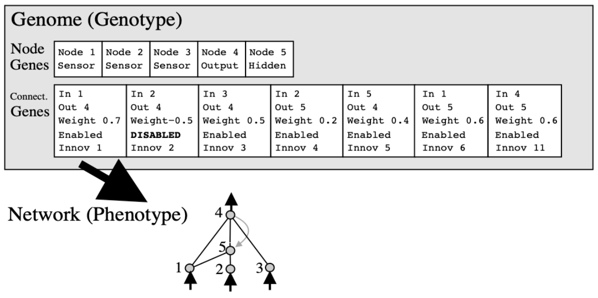

Genome (Genotype)
Node Genes
Node 1 Sensor
Node 2 Sensor
Node 3 Sensor
Node 4 Output
Node 5 Hidden
Connect. Genes
In 1 Out 4 Weight 0.7 Enabled Innov 1
In 2 Out 4 Weight−0.5 DISABLED Innov 2
In 3 Out 4 Weight 0.5 Enabled Innov 3
In 2 Out 5 Weight 0.2 Enabled Innov 4
In 5 Out 4 Weight 0.4 Enabled Innov 5
In 1 Out 5 Weight 0.6 Enabled Innov 6
In 4 Out 5 Weight 0.6 Enabled Innov 11
Network (Phenotype)
1
2
3
4
5

Fig. 1. Genetic Encoding of ANNs in Original NEAT. [4]

This process is guided by mutation and crossover operations that enable NEAT to progressively discover complex and efficient architectures. The main mutation operations involve adding nodes along existing edges, as well as adding connections between existing nodes that are not yet connected (Figure 2). Further possible weight-evolving operations involve shifting the weight and bias terms by some random value drawn from a uniform distribution, for example.

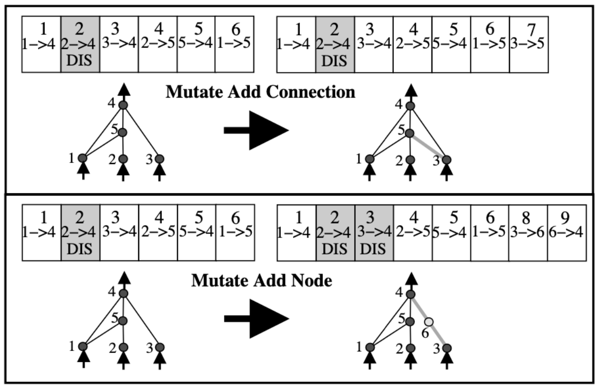


Fig. 2. Mutations of ANNs in Original NEAT. [4]

Crossover occurs by lining up the connection genes between two parent genomes based on their global innovation numbers and selecting genes from fitter parents or genes that currently only belong to one parent, as shown in Figure 3. The algorithm ensures innovation preservation through speciation, grouping similar networks into species to prevent premature convergence.

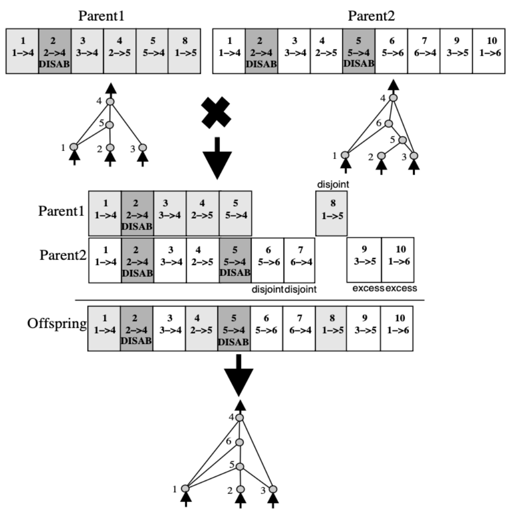


Fig. 3. Crossovers of ANNs in Original NEAT. [4]

NEAT has been successfully applied to a wide range of tasks, including reinforcement learning and supervised learning. It has been further extended in an attempt to evolve different forms of deep neural networks, as seen in CoDeepNEAT [5]. However, there has been little prior work to exactly how NEAT can be extended to evolve CNNs; with different node and layer types, different meanings behind edges between nodes, it is necessary to design a new genetic encoding scheme with new evolutionary operators to effectively evolve CNNs using NEAT in a systematic manner. Further, NEAT's performance is highly dependent on its hyperparameters (e.g., mutation rates, crossover probabilities). Fine-tuning these hyperparameters manually is tedious and often suboptimal, motivating the need for an automated method like PSO to enhance NEAT's performance.

B. PSO

Inspired by the collective behavior of flocking birds and schooling fish, Particle Swarm Optimization (PSO) was introduced by Kennedy, Eberhart, and Shi in the 1990s as an optimization algorithm [6]. Each potential solution in the search space is represented by a "particle", and these particles collectively form a "swarm". The particles move through the search space, guided by three components of their velocity:

1) Inertia (the particle's previous velocity)
2) Cognitive component (the particle's best-known position)
3) Social component (the swarm's best-known position)

These components work together to iteratively direct each particle toward optimal regions of the search space. Over time, the swarm converges on solutions that minimize (or maximize) the objective function. This dynamic is depicted in Figure 4. The previous velocity is represented in green, the cognitive component in blue, and the social component in red. All three vectors are utilized to calculate the new velocity and resultingly, the new particle position (in yellow).

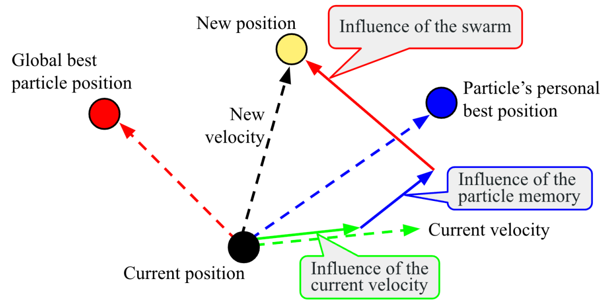

New position
Influence of the swarm
Global best particle position
Particle's personal best position
New velocity
Influence of the particle memory
Current velocity
Current position
Influence of the current velocity

Fig. 4. Breakdown of PSO position update

However, the original PSO algorithm was designed for single-objective optimization, limiting its applicability to problems involving trade-offs between competing objectives. To address this, several extensions of PSO have been developed. For instance, Multi-Objective PSO (MOPSO), as proposed by Li et al. [7], adapts PSO for multi-objective optimization in traffic flow forecasting by placing particles on the Pareto frontier.

The Pareto frontier is a key concept in multi-objective optimization. It represents the set of non-dominated solutions, where no single solution can be improved in one objective without worsening another. In the context of MOPSO, particles on the Pareto frontier reflect optimal trade-offs between competing objectives, such as minimizing bias and variance in Li et al.'s work [7]. This allows the algorithm to maintain diversity in the solutions while exploring the search space more effectively. By guiding the swarm toward the Pareto frontier, MOPSO ensures a balance between exploration and exploitation, making it particularly suited for complex, multi-objective optimization problems like NAS.

III. Approach

Our approach consists of two steps: 1) designing and implementing an extension of NEAT to evolve convolutional neural networks, and 2) integrating NEAT-CNN with PSO.

A. NEAT-CNN

The original NEAT approach [4] works for evolving fully connected feedforward neural networks, similar to the topology of Multi-layer Perceptrons (MLP). However, it cannot be directly used to evolve CNNs. CNNs are primarily composed of convolutional operators, pooling operators, activation operators, and a fully-connected image classifier layer [8]. Convolutional operators contain a $c * k * k$ kernel, where c represents the number of channels and k is the kernel size. These operations slide the kernel over input feature maps to detect local patterns and features. Further, pooling operations, typically max or avg pooling, reduce the spatial dimensions of each feature map while retaining important information. They enable CNNs to capture hierarchical feature representations at different levels of abstractions. Convolutional, pooling, and activation layers are typically grouped together and stacked to iteratively extract features of the original input image and feature maps in the previous layer, allowing CNNs to learn complex patterns for tasks such as image classification, object detection, and segmentation.

As such, it's clear that the genetic encoding scheme, as well as the evolutionary operators, in the original NEAT paper do not naturally accommodate the spatial and local connectivity inherent in convolutional layers. For example, it would not make sense for multiple convolutional nodes to be connected to a single successive convolutional node. We introduce two different approaches in an attempt to evolve CNNs via NEAT and search for high-fitness architectures. In this case, fitness is defined by classification accuracy on the CIFAR-10 image dataset. Both approaches were implemented by extending an existing NEAT-Python implementation [9], which was already capable of running NEAT on fully-connected ANNs on examples such as learning XOR.

*1) **Approach 1: Evolving the Entire CNN**:* Our first approach involves designing a new genetic encoding scheme for CNNs, as well as defining mutation and crossover evolutionary operators to make NEAT functional for evolving CNNs.

Genetic Encoding. Our genome consisted of Convolutional Nodes (ConvNode), Pooling Nodes (PoolNode), and the dense layers genome (NeuralNetGenome). To start simple, the NeuralNetGenome was the existing NEAT genome capable of evolving a fully-connected ANN; we used this for our last dense layer in order to receive learned feature maps as input and output class probabilities for image classification. The ConvNodes and PoolNodes each contained the mutable parameters as shown in Table I. Besides the actual weights, each parameter contained a value initialized or randomly chosen, say during mutation, out of a possible set of parameter values. For example, the kernel size of a convolutional layer was either 2, 3, or 4. This prevented outputs with extremely small and uninsightful feature maps. Unlike the dense layers, the ConvNodes and PoolNodes were simply expressed as an ordered list of ConvNodes and PoolNodes to ensure that each node was at most connected to another node in a linear fashion, as shown in Figure 5.

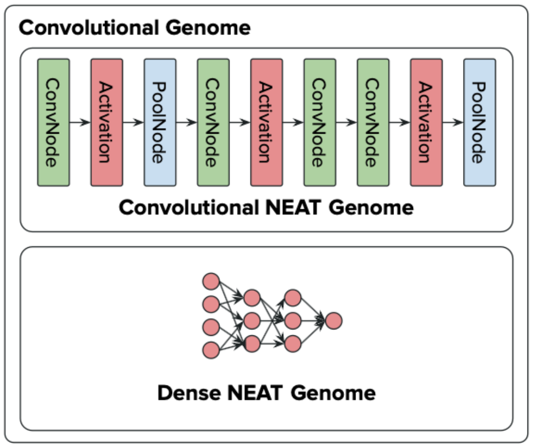


Fig. 5. NEAT-CNN Convolutional Genome

Convolutional genomes are compared by comparing structural and weight differences between CNN architectures. Differences in types/number of ConvNodes and PoolNodes are compared for structural differences, and weight differences are accounted for by comparing kernel weight/size, stride value, and pooling operation differences. The entire genomic distance is a sum of the distance between convolutional layers and distance between dense layers of the two genomes. The genomic distance is useful for identifying genomes that belong to the same species in preserving diversity during evolution.

Mutations. Mutation operations on convolutional and pooling layers involved adding nodes, deleting nodes, shifting kernels, and mutating pooling operations. ConvNodes and PoolNodes can each be added or deleted with equal probability at a random index in our list of nodes. To mutate a kernel for a ConvNode, a random convolutional layer is chosen, and uniform noise drawn from $[-1, 1]$ is drawn and added to some percentage of weights in the kernel. PoolNodes are mutated by randomly changing the existing operation to one of $[max, avg]$. Finally, activation functions can be randomly mutated as well between $[sigmoid, relu]$. By default, $relu$ activation and max pooling are used.

Node	**Parameters**
ConvNode	• kernel size: [2, 3, 4] • stride: [1, 2, 3] • activation: 'sigmoid', 'relu' • filter weights: numpy array
PoolNode	• kernel size: [2, 3, 4] • stride: [1, 2, 3] • activation: 'sigmoid', 'relu' • pool operation: 'max', 'avg'

TABLE I
PARAMETERS OF CONVOLUTIONAL AND POOLING NODES.

Crossover. Crossover between convolutional genomes is highly similar to how it's handled by the original NEAT algorithm. Given two parents, common nodes are inherited from the fitter parent. The main issue lies in ensuring matching channel dimensions between consecutive layers after crossover of ConvNodes/PoolNodes has occurred. As such, for each node in the list of convolutional/pooling nodes in the child, we create and initialize filter weights of layer i so the input/output shapes are to be compatible with layer $i-1$ and $i+1$.

Phenotype Generation. As each convolutional genotype (list of ConvNodes and PoolNodes) can produce different dimension feature maps at each layer, it was highly important to guarantee some dimensional commonality between all CNNs to ensure that the final feature map could be passed to the NEAT-evolved dense layers for classification. The phenotype generation process begins with the first layer being the input layer, a $32 * 32 * 3$ image from CIFAR-10. The ordered list of ConvNodes and PoolNodes are converted into their respective PyTorch layers with the stored parameters and weights in each node. To facilitate compatible input/output dimensions between convolutional and pooling layers, $1 * 1$ convolutions are used. Further, mutations and crossover operations could cause layers to have kernels with larger sizes than the input's size, so these layers were ignored. This process is outlined in Algorithm 1.

Algorithm 1 NEAT-CNN Generation

```
1: Input: [ConvNodes, PoolNodes]
2: Output: PyTorch CNN layers
3: for each ConvNode/PoolNode in list do
4:     if conv node is a ConvNode then
5:         Create convolutional layer with given kernel size, stride, filter weights, and activation layer
6:     else if conv node is a PoolNode then
7:         Create pooling layer with given kernel size, stride, pooling operation, and activation layer
8:     end if
9:     if next node is ConvNode and channel mismatch then
10:        Insert 1x1 convolution to match channels
11:    end if
12: end for
13: Return: PyTorch CNN Layers
```

Forward Pass. To classify a CIFAR-10 image, the image is loaded and propagated through the convolutional and pooling layers generated by Algorithm 1. Any layers that violate shape and size constraints, such as a convolutional layer containing a kernel size too large for the existing input shape, are ignored during the forward pass. Adaptive pooling and padding are applied to ensure all final feature maps result in a flattened feature vector of the same size; this allows all individuals to pass the same feature shape to our dense layers that are also being evolved by NEAT. Thus, no matter the CNN topology, the final dense layers have the same input size. The feature vector is passed through the dense layers, softmax is applied, and the class with the highest probability is returned. This is outlined in Algorithm 2.

As the number of parameters and size of CNNs to optimize is significantly larger than evolving simple feed-forward ANNs, NEAT-CNN will take significantly more generations, as well as larger populations, in order to find reasonably-high fitness architectures. With limited time and computational resources, we employ approach 2 to find higher-fitness architectures in a significantly smaller search space by focusing on solely evolving NEAT for the dense layer classifier.

*2) **Approach 2: Evolving Dense Classifier**:* In this approach, we use a pre-trained feature extractor, and we're simply searching for the architecture and weights of the last dense layers in a CNN. The dense layers take in a $1*N$ feature embedding produced by the feature extractor, and outputs the class prediction with the highest probability. In this approach, the original NEAT algorithm can be directly applied to evolve the dense layer image classifier. We use a pretrained ResNet-18 [10] as our feature extractor. We remove the last dense layer from the existing architecture, and initialize all individuals with the same ResNet-18 feature extractor with pretrained weights. We flatten the output of the last ResNet-18 layer

Algorithm 2 NEAT-CNN Forward Propagation

```
1:  Input: Input $32x32x3$ image $x$
2:  Output: Predicted class
3:  Run NEAT-CNN Generation (Algorithm 1)
4:  Apply forward pass:
5:  for each layer in model do
6:      if layer is convolutional or pooling and input size is too small for kernel then
7:          Skip this layer
8:      else
9:          Pass input through layer
10:     end if
11: end for
12: Remove invalid layers (if any)
13: Adaptive pooling to match desired output size
14: Flatten output
15: Padding (if applicable) to match target feature embedding output size
16: Apply dense layers to the flattened feature embedding
17: Softmax output
18: Return: Class with highest probability.
```

and feed it into the fully-connected ANN we're evolving via NEAT, as shown in figure 6.

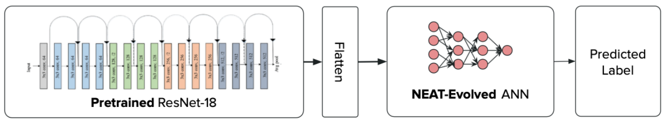


Fig. 6. Pretrained ResNet-18 with NEAT-Evolved ANN

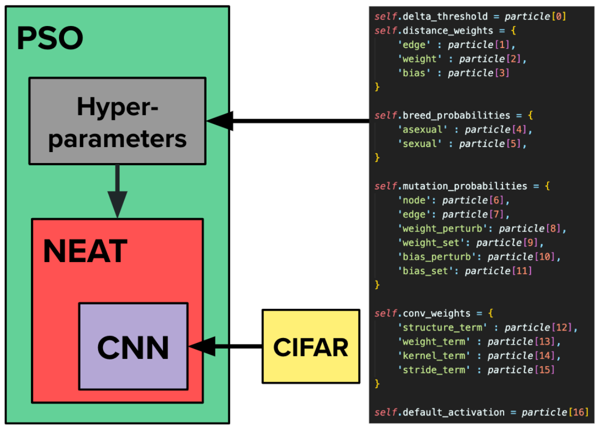
B. NEAT-PSO Integration

Our proposed framework integrates NEAT and PSO into a hybrid NAS pipeline (Figure 7). The framework consists of two evolution loops:

1) Outer Evolution Loop: This loop employs MOPSO (green box) to optimize the hyperparameters of NEAT, such as mutation probabilities, crossover probabilities, and speciation thresholds.
2) Inner Evolution Loop: Using the hyperparameters provided by PSO (right side image), NEAT (red box) evolves CNN architectures (purple box) tailored for the CIFAR-10 image classification task (yellow box).

The resulting CNNs are evaluated on their classification accuracy as well as their complexity (computed as number of parameters), and the performance metrics are fed back to the PSO loop for further optimization of NEAT's hyperparameters. To implement MOPSO, the algorithm will be minimizing error (1 - accuracy) and complexity on the Pareto frontier.

The rationale for this design is twofold. First, NEAT's performance is significantly influenced by its hyperparameters, and PSO excels at efficiently searching through high-dimensional continuous and discrete spaces. Second, this



```
self.delta_threshold = particle[0]
self.distance_weights = {
    'edge' : particle[1],
    'weight' : particle[2],
    'bias' : particle[3]
}

self.breed_probabilities = {
    'asexual' : particle[4],
    'sexual' : particle[5],
}

self.mutation_probabilities = {
    'node': particle[6],
    'edge': particle[7],
    'weight_perturb': particle[8],
    'weight_set': particle[9],
    'bias_perturb': particle[10],
    'bias_set': particle[11]
}

self.conv_weights = {
    'structure_term' : particle[12],
    'weight_term' : particle[13],
    'kernel_term' : particle[14],
    'stride_term' : particle[15]
}

self.default_activation = particle[16]
```

Fig. 7. Diagram for the NEAT-PSO pipeline

hybrid approach allows the strengths of both algorithms to complement each other: PSO optimizes the broader hyperparameter landscape, while NEAT focuses on the intricate task of network evolution.

Despite its promise, this framework presents computational challenges. Evolving CNNs over multiple NAS loops requires substantial resources. Although using TACC's high-performance computing was attempted, it could not be fully utilized within the project's timeline. Consequently, our results focus on showcasing NEAT's ability to evolve effective CNNs as well as PSO's potential to optimize NEAT over successive iterations. These results demonstrate the viability of our approach, while leaving room for future exploration of its full potential.

IV. Experiments

A. NEAT-CNN

We implemented and tested both of our NEAT-CNN approaches to validate that after evolution, our highest-fitness image classifier was sufficiently accurate. For both approaches, we tested on a variety of hyperparameters used to run a NEAT simulation, resulting in "most-fit" organisms after evolution with varying classification accuracies. Examples of hyperparameters that were tuned could be max generations to run, breed probabilities, mutation probabilities, default activation values, and genomic distance thresholds. To evaluate organisms during each evolution generation, we evaluated classification accuracy on 100 sample CIFAR-10 images due to limited compute and time. Ideally, we would be able to evaluate on at least 1000 images.

For approach 1, we ran 50 generations of NEAT with a population of 50 organisms. This experiment was conducted on an M1 Pro chip. A sample run is shown in Figure 8, where the architecture was unable to significantly improve classification accuracy, achieving only marginal improvements and eventually plateauing. Besides the population size, this was run on the same set of hyperparameters that achieved the highest fitness in approach 2.

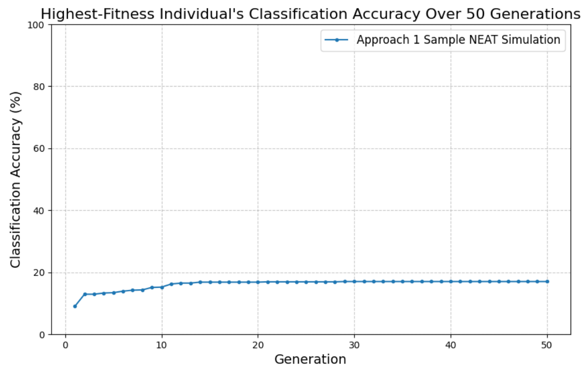


Fig. 8. Sample Approach 1 NEAT-CNN Architecture Performance

Mutation Probability Key	Parameters
Add Node	[0.01, 0.05, 0.1, 0.09, 0.01, 0.15, 0.2, 0.1]
Add Edge	[0.09, 0.05, 0.1, 0.01, 0.09, 0.15, 0.1, 0.15]
Weight Perturb	[0.4, 0.4, 0.3, 0.4, 0.3, 0.2, 0.2, 0.25]
Weight Set	[0.1, 0.1, 0.1, 0.1, 0.2, 0.1, 0.1, 0.1]

TABLE II
MUTATION PROBABILITY EXPERIMENTAL PARAMETERS

For approach 2, we ran 50 generations of NEAT with a population of 200 organisms. This experiment was conducted on an M1 Pro chip. After initial experimentation, we noticed that for evolving feed-forward dense layers, modifying the mutation probabilities for adding nodes, adding edges, and setting/perturbing weights resulted in the largest differences in the improvement of classification accuracy for evolution. As such, we tested on 8 sets of parameters outlined in Table II. For the first 5 sets of hyperparameter, classification accuracy for the highest-fitness individual is plotted after each generation in Figure 9. The highest fitness-individual across all tested hyperparameter sets achieved a classification accuracy of 57% (out of 100 CIFAR-10 images during evolution).

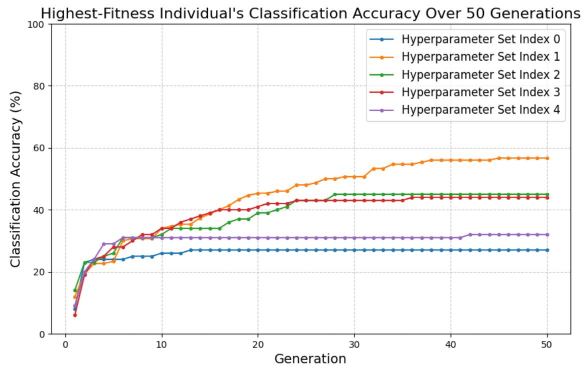


Fig. 9. Sample Approach 2 NEAT-CNN Architecture Performances

B. NEAT-PSO Integration

To evaluate the performance of our proposed hybrid framework, we conducted a small-scale experiment due to computational limitations. Specifically, we ran 7 iterations of PSO with a swarm of 10 particles, each particle representing a unique set of NEAT hyperparameters. For the inner NEAT evolution loop, 7 genomes were evolved over 15 generations, producing CNN architectures that were evaluated on the CIFAR-10 classification task using architectures produced by approach 1, as that was implemented and available to experiment with first. This experiment was conducted on an M2 Pro chip, with a total runtime of approximately 4.5 hours.

The resulting classification errors and network complexities across the 7 PSO iterations are summarized in Figure 10 and Figure 11, respectively. While this limited experimental setup is not sufficient to fully validate the framework, it provides key insights into its potential performance and scalability.

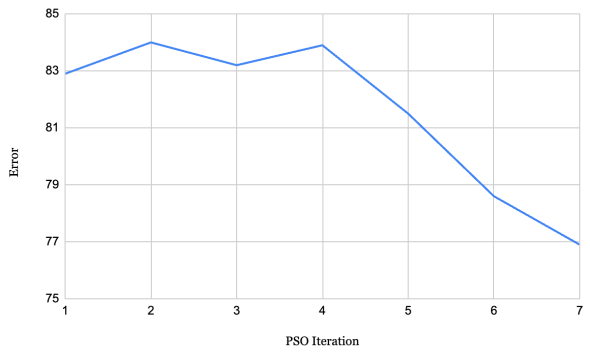


Fig. 10. NEAT-optimized CNN Error over PSO iterations

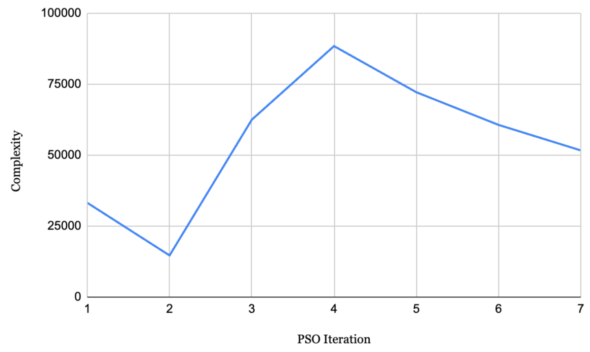


Fig. 11. NEAT-optimized CNN complexity over PSO iterations

V. DISCUSSION & FUTURE WORK

A. Discussion

From our experiments involving NEAT-CNN, we find that with the same compute available, along with the same sets of hyperparameters, approach 2 shows more promising high-fitness individuals after evolution, if fitness was solely based on classification accuracy. Specifically, the highest fitness individual from approach 2 achieved approximately 57% in classification accuracy during training (on 100 CIFAR-10 images). When evaluating the most-fit architecture at the end on 1000 images, we observe top-1, top-2, and top-3 accuracies of 24.6%, 37.2%, and 47.1%. Even with a pretrained feature extractor, our classification accuracy is still far from optimal. On the other hand, after running for 50 generations, the highest individual between sample NEAT simulations using approach 1 ranged from 15% to 30%. Figure 8 shows a sample NEAT simulation of an individual achieving 17% accuracy after evolution. Evaluating on 1000 images, we observe top-1, top-2, and top-3 accuracies of 16.5%, 23.2%, 31%. These accuracies are each approximately 10% lower than those produced by Approach 2 CNN architectures, but still show evidence of learning the classification task.

In approach 1, the highest-fitness individual, as expected, began classifying at a random rate of 10% accuracy, which is expected. Over 50 generations, it experienced marginal increases in accuracy between each generation, which were often within 1-2% of one another. The small improvement can be attributed to the large search space we are trying to optimize over. Considering all kernels across all convolutional layers, mutations to their weights could potentially lead them to a better result, but only by a little at each iteration when mutating a particular layer. Furthermore, as the $1 * 1$ convolutions were largely meant to facilitate matching input/output channel dimensions between layers, and the $1 * 1 * c$ kernel filter was not a part of the evolution process, taking a randomly weighted sum of a particular pixel across all channels could also cause some adverse effect in creating meaningful output feature maps. As such, we believe that for evolving CNNs, crossover brings the majority of improvement in evolution, where meaningful feature maps from stronger parents are inherited, and when combined with layers from both parents, has the opportunity to improve fitness and diversity. We believed that the search space, given our compute, was too large to achieve reasonably high classification accuracy using approach 1 when our evolutionary operators were not strong enough to influence species to move towards optimal architectures. As such, we tested a variety of alterations to approach 1, such as constraining our set of convolutional layers to not be of a random kernel with random weights, but rather randomly sampled from layers from pretrained models with pretrained weights. However, this approach would also be infeasible, as feature maps from different models have unique semantic meanings that capture different local relationships, and performing crossover between such layers does not make sense. Despite the novelty in designing NEAT-CNN with this approach, where evolution takes place in the convolutional layers, we explored approach 2 as it presented more potential for achieving higher classification as optimization was done in a significantly more constrained space.

In approach 2, as plotted in figure 9, all simulations began with the highest-fitness individual classifying slightly better or worse than random guessing (10% for CIFAR-10 classification), which is expected behavior. This is because although we were using a pretrained ResNet-18 feature extractor, the final dense layer classifier is simply composed of input nodes ($1 * N$ flattened feature vector) and output nodes (10 class softmax prediction), where all input and output nodes are connected to each other with weights and biases initialized to 1.0. Despite different sets of hyperparameters producing different most-fit individuals at the end of 50 generations, each producing varying levels of classification accuracy from 27% to 57%, we see that in all hyperparameter sets tested, most populations experience a plateau in classification accuracy for long periods before experiencing any form of minute improvement in classification accuracy. Even in a significantly reduced search space as compared to approach 1, the implemented mutation and crossover operations appear to only marginally improve architecture performance. This could be attributed to a variety of factors, such as: 1) a delta threshold or genomic distance function that doesn't group individuals into species finely enough, or 2) an insufficient number of image samples evaluated for each individual at every generation. Reason 1 could cause individuals that should belong in the same species to be classified as different species, resulting in less diversity in the next generation's population and eventually causing most architectures to be highly similar. Reason 2 could cause lucky architectures that realistically have significantly lower classification accuracy (if tested on a larger sample of images) to dominate the population, when other better-performing architectures should be considered more fit. Nevertheless, we still see an improvement and some form of learning over our evolution amongst all NEAT simulations. It's important to note that the fitness of these architectures could be improved by scaling up the population size and number of generations that the evolution was run for.

As highlighted in the Experiments section, even a minimal implementation of our hybrid NEAT-PSO framework required significant runtime, limiting the scale and scope of our testing. The 4.5-hour runtime on the M2 Pro chip was sufficient to run only a small number of PSO and NEAT iterations, restricting our ability to explore the framework's full capabilities. Scaling this framework to more particles, genomes, and generations—or integrating the complete NEAT evolution—would demand much greater computational resources.

Despite these limitations, the classification error rates exhibit a clear downward trend over successive PSO iterations. Although the scale of the error graph is relatively constrained (ranging from 75% to 85%), the largest observed improvement—a 7.1% decrease in error—is noteworthy. This improvement suggests that PSO is effectively optimizing NEAT's hyperparameters, enabling it to produce CNN architectures with better accuracy.

However, the complexity of the evolved networks (as reflected in their node and connection counts) displayed high variability, with no discernible correlation to the error rates. This result may reflect the inherent stochasticity of NEAT's evolutionary process or the small number of particles and genomes in our experimental setup. Larger-scale experiments with more extensive NEAT runs could help clarify whether a trade-off between complexity and performance emerges over time.

Our results also emphasize the computational challenges of combining NEAT and PSO at scale. A more full implementation of the framework—incorporating our tested NEAT evolution and 10 iterations of PSO on 10 particles—was estimated to take approximately 2.5 days of continuous runtime on the current setup. Such a timeline leaves little room for debugging or addressing potential errors, highlighting the need for more powerful hardware or distributed computing environments to fully explore the framework's potential.

Nonetheless, the observed trends are promising and provide proof of concept for the hybrid NEAT-PSO framework. The results validate the theoretical basis of the approach, demonstrating that PSO can optimize NEAT's hyperparameters to improve CNN performance. While we could not fully implement or validate the combined framework at scale, the observed trends suggest that integrating these methods has the potential to yield even greater improvements in both accuracy and efficiency.

B. Future Work

One potential avenue of expanding this work is to integrate PSO dynamically within NEAT's evolution cycle. In other words, instead of running PSO exclusively in the outer loop, it could be invoked periodically to fine-tune genome parameters mid-evolution. This hybridization would allow NEAT to remain the primary evolutionary algorithm while leveraging PSO's efficiency to accelerate convergence.

While our experiments demonstrate the effectiveness of combining NEAT and PSO, the computational limitations restricted us from exploring deep evolutionary runs. A promising direction for future research would involve deploying this framework on large-scale computing resources to evaluate its scalability and robustness over extended generations. Such experiments could validate the assumption that the performance gains are scalable and lead to even more efficient architectures.

And although this work focuses on image classification with CIFAR-10, the framework could be adapted for other domains, such as natural language processing or reinforcement learning. Investigating its performance across diverse tasks would help establish the generalizability of the proposed approach.

By addressing these directions, future research could fully realize the potential of combining NEAT and PSO for hybrid NAS, opening new avenues for optimizing neural network architectures.

VI. Conclusion

In this paper, we presented a novel hybrid framework that integrates NEAT with PSO to enhance the process of NAS. Our experiments demonstrated the feasibility of evolving CNNs using NEAT, as well as the potential of PSO to optimize NEAT's hyperparameters.

While the results achieved in our experiments were modest, with the highest fitness individual reaching a classification accuracy being far from optimal, they provide a valuable proof of concept for our proposed approach due to the improvement in accuracy over a limited population size and number of generations. The observed trends indicate that the integration of NEAT and PSO can lead to improvements in CNN performance, albeit within a limited experimental setup.

The hybrid framework's ability to optimize hyperparameters dynamically through PSO while leveraging NEAT's strengths in architecture evolution suggests a promising direction for future research. Despite the computational challenges encountered, the downward trend in classification error rates across PSO iterations highlights the potential for further optimization and refinement of the framework.

Moving forward, we aim to scale our experiments to explore the full capabilities of the NEAT-PSO integration, potentially leading to more efficient and effective neural network architectures. Additionally, adapting the framework for diverse tasks beyond image classification could further validate its generalizability and impact in the field of neural architecture optimization.

References

[1] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[2] Y. Liu, Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "A survey on evolutionary neural architecture search," *CoRR*, vol. abs/2008.10937, 2020. [Online]. Available: https://arxiv.org/abs/2008.10937

[3] K. Alex, "Learning multiple layers of features from tiny images," *https://www. cs. toronto. edu/kriz/learning-features-2009-TR. pdf*, 2009.

[4] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[5] R. Miikkulainen, J. Z. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat, "Evolving deep neural networks," *CoRR*, vol. abs/1703.00548, 2017. [Online]. Available: http://arxiv.org/abs/1703.00548

[6] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-international conference on neural networks*, vol. 4. ieee, 1995, pp. 1942–1948.

[7] L. Li, L. Qin, X. Qu, J. Zhang, Y. Wang, and B. Ran, "Day-ahead traffic flow forecasting based on a deep belief network optimized by the multi-objective particle swarm algorithm," *Knowledge-Based Systems*, vol. 172, pp. 1–14, 2019.

[8] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *CoRR*, vol. abs/1511.08458, 2015. [Online]. Available: http://arxiv.org/abs/1511.08458

[9] SirBob01, "Neat-python," 2024, accessed: 2024-12-14. [Online]. Available: https://github.com/SirBob01/NEAT-Python

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385