Optimizing Hyperparameters for Deep Learning Models Using Evolutionary Algorithms: Solving the Four-Class Intertwined Spiral Classification Problem

Siri Sri Churakanti
Dept of Math & Computer Science
Lawrence Technological University
Southfield, MI, USA
schurakan@ltu.edu

Batyr Kenzheakhmetov
Dept of Math & Computer Science
Lawrence Technological University
Southfield, MI, USA
bkenzheak@ltu.edu

Chan-Jin Chung
Dept of Math & Computer Science
Lawrence Technological University
Southfield, MI, USA
cchung@ltu.edu

Abstract— **This paper presents novel approaches to optimizing Deep Neural Networks (DNN) for the four-class intertwined spiral classification problem using an Evolution strategy algorithm with 1/5 success rule and a Genetic Algorithm (GA) implemented via the DEAP framework. Unlike the simpler two-class spiral, this four-class variant remains largely unexplored in machine learning fields. Evolutionary algorithms can provide a dynamic and adaptive mechanism that optimizes hyperparameters of DNNs for non-linear classification tasks and this study leverages ES(1+1) with the 1/5 success rule and DEAP based Genetic Algorithms. Without hyperparameter optimization, the baseline model achieved an average accuracy of 34.1% with an average loss of 1.32. GA with DEAP optimizing hyperparameters achieved the best classification accuracy of up to 95%. ES(1+1) with the 1/5 success rule found a model that delivers 93% accuracy. These findings establish evolutionary algorithms as powerful tools for enhancing DNN performance and provide valuable insights for future advancements in deep learning optimization.**

Keywords— Deep neural networks (DNN), Evolutionary Computation, Evolutionary Algorithms, Genetic Algorithms, Evolution strategy, 1/5 success rule, 4-class spiral classification, DEAP, Hyperparameter optimization, Evolutionary Deep Learning

I. Introduction

The two-class spiral classification problem [1] is a well-established classic benchmark for assessing the capabilities of machine learning models in handling nonlinear separability. The two-spiral task became famous because it was regarded as an extremely difficult non-linear classification task for multilayer perceptrons to solve [2]. However, this four-class spiral problem, with its four interlocking, nonlinearly separable spirals, presents a significantly greater challenge and remains uncharted in the evolutionary deep learning domain.

This paper introduces applications of two Evolutionary Algorithms (EAs) that use the idea of natural selection to optimize hyperparameters of deep neural networks for solving the four-class intertwined spiral problem. The motivation for this study arises from the limitations of conventional optimization methods, such as gradient-based techniques, which often struggle in high-dimensional, nonlinear landscapes like that of the four-class spiral problem. These traditional methods frequently become trapped in local minima, hindering their ability to find global optimal solutions in complex, non-convex search spaces. EAs are less susceptible to local minima and excel at identifying global optima, making them particularly well-suited for high-dimensional, nonlinear optimization tasks.

Section II describes related similar works, section III describes hyperparameter optimization methodologies using 2 evolutionary algorithms after explaining how we generate datasets, section IV describes the results of testing algorithms developed, and section V concludes this research.

II. Related Works

Chalup and Wiklendt in 2007 [1] offer a detailed review of various approaches applied to this task, including constructive neural networks, support vector machines, and evolutionary computation techniques. Their analysis highlights the problem’s importance as a testbed for advancing classification algorithms and applications of evolutionary algorithms were introduced as a promise for the two-class spiral classification problem.

Yang and Kao in 2001 [14] proposed an evolutionary approach called the Family Competition Evolutionary Algorithm (FCEA) using both crossover and mutation operators to find optimized weights of a DNN on the two-spiral data with 97x2=194 points [1]. The result shows that the FCEA was successfully classified 180 points out of 194 successfully (92.7%).

Miranda in 2017 [15] used Particle Swarm Optimization (PSO) to train neural networks, achieving an 85.53% recall rate. PSO’s approach balances exploration and exploitation, highlighting evolutionary methods’ potential.

Recently in 2022, a two-class spiral problem has been studied using ES to optimize neural network architecture hyperparameters [16]. The best model found using an evolutionary search was able to make predictions on the test set with 98.1% accuracy.

However, their works focuses only on the two-class variant, leaving the four-class problem unaddressed as far as we have found so far.

III. Methodology

To tackle the four-class spiral problem, we develop two different algorithms. As described in subsection C, ES(1+1) with 1/5th success rule algorithm using mutation operators only is implemented from scratch using the knowledge-based self-adaptation ideas introduced in [8]-[10]. These works offer valuable insights into guiding search processes with generalized knowledge, achieving robust results in non-linear function optimization tasks.

Our 2nd algorithm, described in subsection D, is based on GA using both crossover and mutation operators using the Distributed Evolutionary Algorithms in Python (DEAP) framework [3]. DEAP is a versatile and flexible tool for evolutionary algorithms, ideal for rapid development due to its modular design.

The generation of four-class spiral datasets is explained in subsection A. Subsection B describes a baseline DNN model architecture with randomly chosen hyperparameters.

A. *Dataset Generation*

The dataset is constructed as four intertwined spirals [17], each represented in 2D polar coordinates. Although synthetic, the four-class spiral dataset serves as a well-established benchmark for evaluating machine learning models on nonlinear separability [1]. Its controlled complexity allows for a clear understanding of how evolutionary algorithms enhance deep neural network (DNN) performance in challenging classification scenarios. For a class, the spiral points are generated using the equations shown in (1).

$$x(t) = a.t.\cos(t+b)\,,\ y(t) = a.t.\sin(t+b) \qquad (1)$$

where, a = scaling factor, t = angle in radians (range: [0, 10]), and b = angular offset for each class (b = 0, π/2, π, 3π/2).

These parameters ensure proper generation of the spiral classes, with $t \in [0, 10]$ representing the angle. Gaussian noise $N(\mu, \sigma^2)$ was added to simulate real-world imperfections. The Gaussian noise with a mean (μ) of 0.0 and a variance (σ^2) of 0.03, which introduces variability to the dataset, making it more representative of real-world data. The input data $X \in \mathbb{R}^2$ is standardized to zero mean and unit variance using:

$$X' = \frac{(X-\mu_x)}{\sigma_x} \qquad (2)$$

where μ_x and σ_x are the mean and standard deviation of the input data, respectively. Standardization ensures that all features are on the same scale, which is crucial for aiding convergence during training. Here in Fig. 1 shows the visualization of training dataset with 4 classes in red, blue, green and purple.

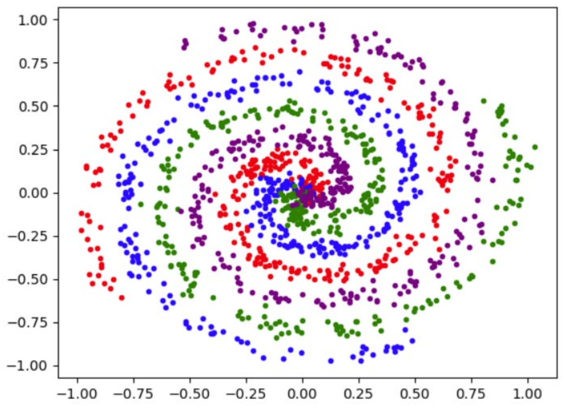

Fig. 1. A sample training dataset visualization

B. *Baseline model*

As a baseline model, a deep neural network (DNN) with three hidden layers was implemented and tested using Keras with TensorFlow as the backend. The network architecture consists of:

- Input layer: 2 neurons (x, y coordinates).
- Hidden layers: Three layers with 64, 32 and 16 neurons, respectively. Each hidden neuron is activated using ReLU function for hierarchical feature extraction.
- Output Layer: 4 neurons with SoftMax activation function for classification.

The model was trained using the RMSprop optimizer with a learning rate of 0.01, a batch size of 16, and a maximum of 50 epochs. Early stopping was employed to prevent overfitting by monitoring validation loss and restoring the best model weights.

C. *ES (1+1) with 1/5 Success Rule Algorithm*

Unlike conventional evolutionary strategies or genetic algorithms, ES(1+1) generates one offspring per iteration from a single parent using the 1/5 success rule. This rule dynamically adjusts the mutation step size: if more than 20% of previous mutations improve fitness, the step size increases to promote exploration; otherwise, it decreases to exploit the search space. This adaptive mechanism is critical for optimizing DNN hyperparameters in complex, nonlinear tasks like the four-class spiral, where static approaches often fail to converge efficiently. Hyperparameter structure of a candidate solution includes: (1) number of neurons between 8 and 512 per hidden layer, (2) activation function randomly selected from ReLU, ELU, Sigmoid, and Tanh for hidden neurons, (3) optimizer randomly selected from SGD, RMSprop, and Adam to find connection weights, (3) learning rate between 0.01 and 0.1, and (4) training batch size between 16 and 64. Algorithm 1 below overviews how the ES algorithm processes in pseudo code.

Algorithm 1. ES(1+1) with 1/5 Success Rule:

```
Initialize a parent solution with random hyperparameters
Evaluate the parent using Keras
Set success_window = 10, successes = 0
Iterate through each generation until the termination condition is met:
  Generate an offspring solution using current stepsize, σ:
    Mutate number of hidden neurons (using eq. 3)
    With 20% probability,
      randomly select new activation function and optimizer
    Mutate learning rate (using eq. 3)
    Mutate batch size (using eq. 3)
  Evaluate the offspring using Keras
  If offspring_fitness < parent_fitness: (min problem)
    Offspring becomes the parent for next generation
    successes += 1
  If current_generation%success_window == 0
    Update stepsize, σ using 1/5 success rule (see eq. 4)
    Reset successes = 0
End-of-Iterate
Output best model with optimized hyperparameters
```

As described in Algorithm 1, the ES(1+1) optimization begins with a single parent solution. To generate a new offspring solution, Gaussian random number is added to the parent solution's parameters using the formula:

$$\hat{h} = h + gauss\left(\sigma.\left(max(h) - min(h)\right)\right) \tag{3}$$

where h is a hyperparameter (e.g., learning rate, neuron count), σ is the mutation step size, and *gauss* adds Gaussian noise. The mutation step size (σ) is adapted using the 1/5 success rule, which adjusts success rate of offspring over the past 10 generations [4]:

$$\sigma_{t+1} = \begin{cases} \sigma_t.\alpha, & if\ success\ rate > \frac{1}{5} \\ \sigma_t.\beta, & otherwise \end{cases} \tag{4}$$

where α > 1 and β < 1 are scaling factors, and the success rate is defined as the fraction of offspring outperforming their parents. If at least 20% of the offspring outperform their parents, the mutation step size increased to encourage exploration. Otherwise, it is reduced to refine exploitation. For this experimentation, initial σ = 0.1, α = 1.2, β = 0,85 were used.

For discrete parameters (e.g., activation function and optimizer), a 20% mutation probability triggers a random switch to another option.

In the selection step in evolutionary process, the offspring replaces the parent if it achieves better fitness, defined as lower validation loss, ensuring iterative improvement.

The ES(1+1) algorithm experiments using Python 3.6.18 were performed on a system with the following specifications:

- OS: Linux 6.8.0-49-generic (#49-Ubuntu SMP)
- Architecture: 64-bit, x86_64 processor with 16 logical CPU cores
- RAM: 245 GB
- GPU: NVIDIA A100 80GB PCIe with 81920 MiB VRAM

The parent and offspring solutions with hyperparameters are evaluated using the DNN models implemented with *Keras* as the backend. The DNN architecture as shown in Fig. 2 as an example includes:

- The input layer consists of 2 neurons, corresponding to x and y co-ordinates in the normalized spiral dataset. This layer processes the 2D features as the initial input to the network.
- Three hidden layers are used for all the experiments here. Each layer contains between 8 and 512 neurons. The exact configuration is determined dynamically during hyperparameter optimization.
- Activation function for hidden neurons include ReLU, ELU, Sigmoid, and Tanh, providing flexibility to model the non-linear relationships in the data.
- The output layer comprises 4 neurons with SoftMax activation function, one for each spiral class.

For training, we use the sparse categorical crossentropy loss function, optimized for multi-class classification with integer-encoded labels. The optimizer (SGD, RMSprop, or Adam) and learning rate (between 0.01 to 0.1) are set dynamically. Training employs a batch size between 16 and 64, with an EarlyStopping callback monitoring validation loss to prevent overfitting, restoring the best weights when training halts.

The validation process involves reserving 20% of the training data as a validation dataset during model training. The EarlyStopping callback ensure generalization by stopping training if validation loss stagnates.

Post-training, the models is evaluated on a separate test set to compute final test loss and accuracy, confirming robust performance on unseen data. Fitness is defined using the test loss from Keras' evaluation() method.

D. Genetic Algorithm (GA) using DEAP

Genetic Algorithms (GAs) are powerful optimization techniques inspired by natural evolutionary processes, including selection, crossover, and mutation, to solve complex problems. In this study, the DEAP framework is utilized to optimize the hyperparameters of a neural network designed for the challenging Four-Class Spiral problem. DEAP's flexibility and efficient computational capabilities make it well-suited for navigating large, nonlinear search spaces. The GA evolves a population of candidate solutions, termed "individuals," over multiple generations to minimize the neural network's test loss.

Each individual represents a unique configuration of hyperparameters, defined in the search space as $x \in R^4$:

$$x = [n, \eta, b, d] \tag{5}$$

where n, η, b, and d represent the number of neurons, learning rate, batch size, and dropout rate, respectively. These parameters are constrained within predefined bounds to ensure feasible and stable neural network training. The integration of DEAP with Keras' deep learning tools facilitates a robust and efficient hyperparameter search, leveraging DEAP's parallelization capabilities to accelerate the process. This makes it an effective tool for complex optimization tasks like the four-class spiral classification.

The DEAP Genetic Algorithm experiments were executed using Google Colab, a cloud-based platform, with the following configuration:

- Operating System: Linux (Google Colab environment)
- Python Version: 3.8.18 (typical Colab default, unless modified)
- RAM: 12 GB (standard Colab allocation)
- GPU: Tesla T4 (commonly provided in Colab's free tier)

Algorithm 2. Genetic Algorithm using DEAP

```
Initialize DEAP: bounds, FitnessMin, Individual, Toolbox (attr generators, individual, population)
Define eval_function(individual): Train Keras model that includes 2 hidden layers with individual params, return (loss,) from test set
Setup Toolbox: Register eval_function, mate (see eq. 7), mutate (eq. 8), select (eq. 6); Decorate mutate with bounds (eq.9)
Initialize population = toolbox.population(pop size)
While termination not met:
  Evaluate unevaluated individuals using eval_function
  offspring = select, clone;
  mate (p=0.5, eq. 7)
```

```
        mutate (p=0.3, eq. 8), bounds (eq. 9)
        population = offspring
        Track best individual and its loss
    End-While
    Output best model with optimized hyperparameters
```

Description of key genetic operators used in DEAP are described in the following paragraphs.

Selection: A tournament selection method selects individuals for reproduction. For a tournament size T, a subset of T individuals is randomly sampled, and the individual with the lowest validation loss is chosen:

$$x_{best} = arg\min_{x \in T} f(x), \tag{6}$$

where $f(x)$ denotes the fitness value (validation loss).

Crossover: To create offspring, the Blend Crossover (BLX - α) operator generates offspring by linearly interpolating between two parent solutions:The $x_{offspring}$ is computed as

$$x_{offspring} = x_1 + \alpha(x_2 - x_1), \tag{7}$$

where x_2 and x_1 are the parent individuals, α controls the interpolation extent.

Mutation: Gaussian mutation introduces diversity by perturbing parameters with noise from a Gaussian distribution.

$$x_i' = x_i + N(\mu = 0, \sigma), \tag{8}$$

where $N(\mu, \sigma)$ has mean $\mu = 0$ and standard deviation σ, where each parameter is x_i adjusted by adding noise sampled from a Gaussian distribution. Mutation occurs with a specified probability, and bounds are enforced:

$$x_i' = max(min(x_i', HIGH_i), LOW_i) \tag{9}$$

The fitness evaluation, implemented in DEAP, constructs and trains a Keras Sequential model for each individual. Training employs categorical cross-entropy loss, the Adam optimizer, and early stopping to mitigate overfitting. The validation loss serves as the fitness metric, with lower values indicating superior performance. DEAP's parallelization enables simultaneous evaluation of multiple configurations, enhancing scalability and efficiency. The best individual,x^*, is identified as:

$$x^* = arg\min_{x \in P} f(x), \tag{10}$$

where P represents the final population.

IV. Results

The results of this study summarized in Table 1 highlight the effectiveness of both evolutionary strategies (ES) and Genetic Algorithm (GA) in optimizing neural network architectures for the complex 4 class spiral classification problem.

Table 1. Result comparisons between Baseline model, ES(1+1) with 1/5 success rule, and GA-DEAP.

Algo-rithm	*Run No.*	*Acc-uracy*	*Loss*	*A*	*Num neurons for each layer*	*LR*	*Bat. Size*	*O*
Baseline	40	0.341	1.31	R	64, 32, 16	0.01	16	M
ES(1+1) 1/5 rule	4	0.941	0.15	R	157,153,160	0.01	53	A
	5	0.937	0.19	R	8, 512, 512	0.01	62	A
	3	0.934	0.17	E	272, 86, 214	0.01	44	S
GA-DEAP	0	0.962	0.22	R	68, 95	0.003	22	A
	5	0.956	0.17	R	187, 35	0.009	32	A
	6	0.943	0.19	R	190, 95	0.007	31	A

LR means Learning Rate, *A* means Activation Function, *O* means Optimizers. R = ReLU, E = ELU, M = RMSprop, A=Adam, S=SDG

From the Table 1, we can understand the model performance better, as they are

- Baseline model: Without evolutionary optimization, the baseline model achieved average accuracy of 34.1% with 1.31 as loss value over 50 runs. This serves as a reference point, underscoring the significant improvements gained through evolutionary algorithms.
- ES(1+1) with 1/5 rule: Models demonstrated robust performance across multiple runs. Run 4 achieved the highest accuracy among ES(1+1) configurations at 94.1%. These results showcase the adaptability of ES(1+1) in finding effective architectures across varying network sizes and its robustness for complex 4 class spiral datasets with noises. A DNN architecture found at run No. 4 is drawn in Fig. 2.
- GA using DEAP: The best accuracy is 95.6% with loss of 0.17, but the highest accuracy is 96.2% with 0.22 loss, balancing complexity and performance effectively. This demonstrates its efficiency in resource utilization. The slightly lower learning rate (0.009 vs. 0.01) likely contributed to finer optimization adjustments, resulting in a low loss.

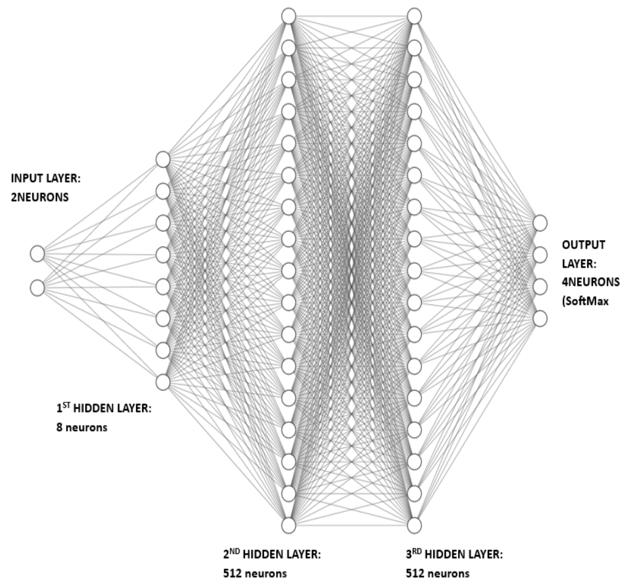

Fig. 2. Optimized DNN for ES(1+1), showcasing from Run No. 5 model with 8, 512, 512 hidden neurons.

Fig. 4 shows a loss over generation graph from Run No. 5 to illustrate the ES(1+1) with 1/5 succees rule algorithm's effectiveness in finding models with optimized hyperparameters.

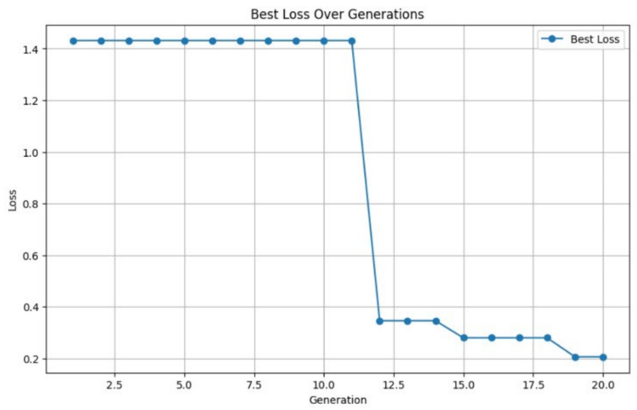

Fig. 4. Loss over Generations for ES(1+1) with the 1/5 success rule from Run No. 5

Fig. 5 plots a loss over generation graph from Run No. 5 to illustrate the GA-DEAP algorithm's effectiveness in finding models with optimized hyperparameters. Note that the best loss was achived at generation 5.

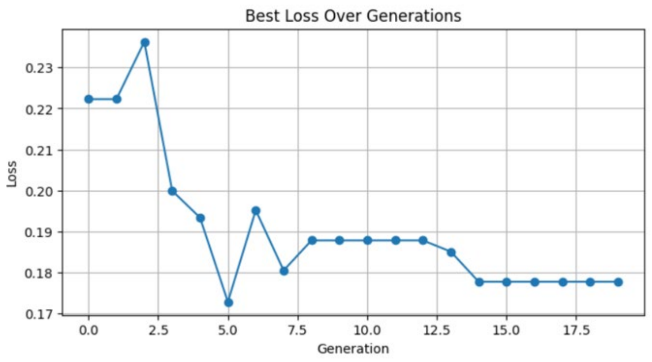

Fig. 5. Loss over generation for GA-DEAP, Run No. 5

A confusion matrix shown in Fig. 6 offers a comprehensive evaluation of accuracy, precision, recall, and prediction distribution across the four classes, highlighting the model's strengths and limitations. This confusion matrix is from the model found by ES(1+1) at Run 5 with 93.7% accuracy. It shows strong performance, with most predictions along the diagonal, indicating correct classifications. However, minor misclassifications occur, notably between Class 3 and Class 4, where 5 instances of Class 4 were predicted as Class 3. This suggests overlapping decision boundaries in regions where the spirals are closely intertwined, a challenge inherent to the four-class spiral problem's nonlinearity.

Decision Boundary diagrams are plotted on a dense feature space grid to showcase the optimized model's classification performance. The decision boundaries for the ES(1+1) using 1/5 success rule and the Genetic Algorithm (GA) using DEAP are drawn in Fig. 7 and Fig. 8 to visualize how the optimized models classify the dataset. Each visualization provides unique insights into model performance.

The best hyperparameter configurations for each algorithm, based on their highest accuracy, are used to train the models, and the decision boundaries are generated by predicting class labels across a grid of the feature space.

In Fig. 7 ES(1+1) model's decision boundary is plotted using its best configuration, which achieved (Run No. 4) 94.1% accuracy with 470 (157+153+160) hidden neurons, a learning rate of 0.01, and a batch size of 53. To plot Fig. 7 graph, a neural network is trained on the spiral dataset with these optimized hyperparameters. Next, a grid of points is generated to span the feature space, and the trained model predicts the class label for each point on this grid.

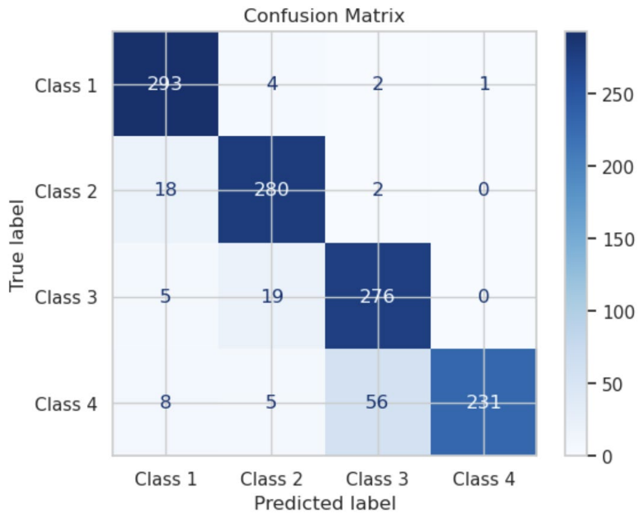

Fig. 6. Confusion matrix from the ES(1+1) optimized model (Run 5), showing true labels vs. predicted labels. Diagonal elements indicate correct classifications, while off-diagonal elements highlight misclassifications

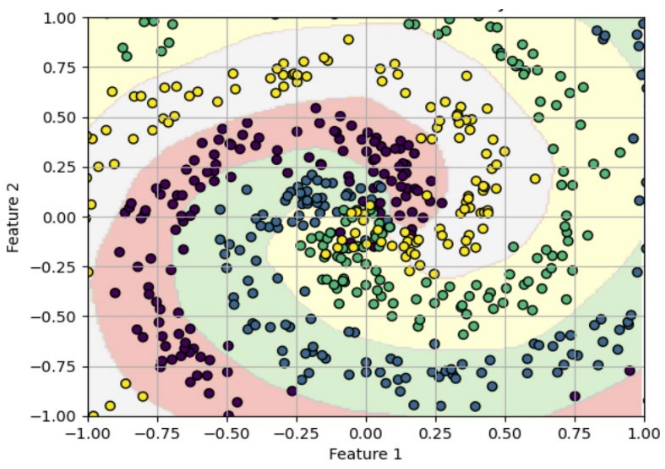

Fig. 7. Decision boundary of the best-performing model (ES(1+1) with 1/5 success rule of 94% , illustrating the separation of the four spiral classes

Fig. 8 shows a decision boundary graph from DEAP Genetic Algorithm. It is plotted using the best individual from the final population, which achieved 95.6% accuracy with 222 (187+35) neurons, a learning rate of 0.009, and a batch size of 32. The model is trained with these hyperparameters, and a mesh grid covering the input space is constructed.

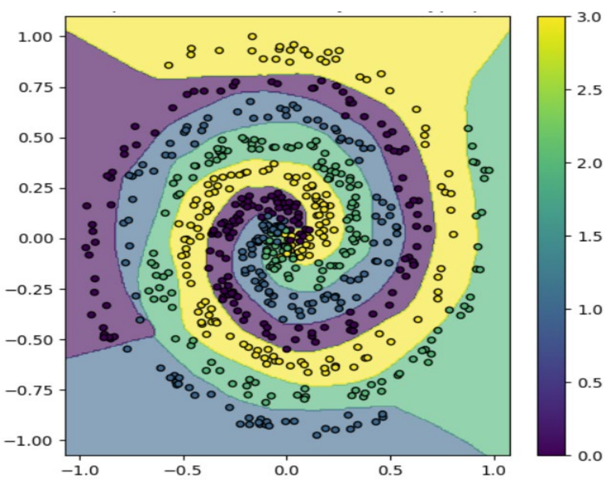

Fig. 8 Decision Boundary of a model found by Genetic Algorithm using DEAP which achieved 95.6% (Run 5)

V. Conclusion and Future Work

This research demonstrates the effectiveness of evolution strategy ES(1+1) with 1/5 success rule and genetic algorithm (GA) using DEAP in optimizing deep neural network hyperparameters (DNNs) for the four-class intertwined spiral classification problem. Starting with a baseline model achieving an average accuracy of 34.1% without optimization, the introduction of hyperparameter optimization using ES(1+1) with 1/5 success rule achieved a peak accuracy of 94.1%, and the GA-DEAP algorithm achieved the highest accuracy of 96.2%, showcasing the effectiveness of these methods.

By optimizing key hyperparameters such as number of hidden neurons, learning rate, batch size, optimization function for training and activation functions, the study highlights the robustness of evolutionary methods in handling nonlinear and complex classification tasks. These findings establish evolutionary algorithms as powerful tools for enhancing DNN performance and offer valuable insights for future advancements in developing deep learning models.

References

[1] Lang, K. J., Witbrock, M. J., 1988. Learning to tell two spirals apart. In: Touretzky, D.,Hinton, G., Sejnowski, T. (Eds.), Proceedings 1988 Connectionist Models SummerSchool. Morgan Kaufmann, Los Altos, CA, pp. 52–59.

[2] Chalup, S. K., & Wiklendt, L. (2007). *Variations of the two-spiral task.* School of Electrical Engineering and Computer Science, The University of Newcastle, Australia.

[3] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," Journal of Machine Learning Research, vol. 13, pp. 2171–2175, 2012.

[4] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," Evolutionary Computation, vol. 9, no. 2, pp. 159–195, 2001.

[5] Manngård, M., Kronqvist, J., & Böling, J. M. (2017). "Structural learning in artificial neural networks using sparse optimization." Neurocomputing, 272.

[6] Lee, S. E., Yoo, H., Chang, J., & Chung, K. (2020). "Feedback-Based Evolutionary Spiral Learning Method for Reducing Class Ambiguity." Supported by the National Research Foundation of Korea

[7] F.-M. De Rainville, F.-A. Fortin, M.-A. Gardner, and M. Parizeau, "DEAP: A Python framework for Evolutionary Algorithms," Proceedings of the Genetic and Evolutionary Computation Conference Companion, July 2012

[8] Chan-Jin Chung and Robert G. Reynolds, "Knowledge-Based Self-Adaptation in Evolutionary Search", International Journal of Pattern Recognition and Artificial Intelligence, Vol. 14 No. 1 (2000), pp. 19-33

[9] Chan-Jin Chung and Robert G. Reynolds, "CAEP: An Evolution-Based Tool for Real-valued Function Optimization Using Cultural Algorithms," International Journal on Artificial Intelligence Tools, Vol. 7, No. 3, (1998), pp. 239-291

[10] Chan-Jin Chung, "Knowledge-Based Approaches to Self-Adaptation in Cultural Algorithms", Ph.D. thesis, Wayne State University, May 1997

[11] Loussaief, S., & Abdelkrim, A. (2018). Convolutional Neural Network Hyper-Parameters Optimization based on Genetic Algorithms. *(IJACSA)* International Journal of Advanced Computer Science and Applications*, 9*(10), 252-261.

[12] Yüzgeç, U., & Karakuzu, C. (2018). Training Multi-Layer Perceptron using Opposition based Learning Spiral Optimization Algorithm. In International Conference on Advanced Technologies, Computer Engineering and Science , Safranbolu, Turkey, May 11-13, 2018.

[13] Al-Hyari, A., & Abu-Faraj, M. (2022). Hyperparameters Optimization of Convolutional Neural Networks using Evolutionary Algorithms. In 2022 International Conference on Emerging Trends in Computing and Engineering Applications*,* Karak, Jordan, November 23-24, 2022.

[14] Yang, J.-M., Kao, C.-Y., 2001. A robust evolutionary algorithm for training neural net-works. Neural Computing & Applications 10 (3), 214–230. Accessed on Mar. 7 at: https://ir.lib.nycu.edu.tw/bitstream/11536/29997/1/000173916700003.pdf

[15] Miranda, L. J. (2017). Training a neural network using particle swarm optimization, Accessed on Mar 7 at: https://ljvmiranda921.github.io/notebook/2017/01/17/pso-trained-neural-network-for-solving-the-two-spiral-problem

[16] Mark Kocherovsky and Chan-Jin Chung, Using Evolutionary Algorithms to Optimize Hyperparameters for Keras Deep Learning Models to Solve the Two Intertwined Spiral Problem, 2022 Research Day, Lawrence Technological University, DOI: 10.13140/RG.2.2.16932.69767

[17] Nihan Kazak and Mehmet Koc. 2018. Some variants of spiral LBP in texture Nihan Kazak and Mehmet Koc. 2018. Some variants of spiral LBP in texture recognition. IET Image Processing 12, 8 (2018), 1388–1393.