University of Sarajevo
Faculty of Electrical Engineering
Department of Computer Science and
Informatics

Evolving Neural Networks with Particle Swarm Optimization

Artificial intelligence

Authors: Benjamin Bandić, Anes Ćenanović, Benjamin Hadžihasanović
Professor: Amila Akagić
Date: 15.06.2025

Table of Contents

Abstract

This project investigates the effectiveness of Particle Swarm Optimization (PSO) for the automated hyperparameter tuning of neural networks. The methodology was applied to a diverse collection of six datasets, covering classification and regression tasks across tabular, audio, and image data domains. The findings successfully demonstrate that PSO is a highly versatile tool. It serves as an effective **fine-tuner** on well-posed problems (e.g., NASA Exoplanets, CIFAR-100), a powerful **problem-solver** capable of making a non-learning model functional (FMA Audio), and a revealer of complex **performance trade-offs** on data with challenging characteristics like outliers (Autolist Cars) or severe class imbalance (Yu-Gi-Oh! Cards). This work concludes that PSO is a valuable technique for evolving neural networks, with its role and impact being highly dependent on the context of the data and the specific problem.

1 Introduction

1.1 Problem Description

The performance of deep learning models is critically dependent on the selection of appropriate hyperparameters, such as the network architecture, learning rate, and regularization techniques. The process of finding an optimal set of these hyperparameters, known as hyperparameter tuning, is a significant challenge. Traditional methods like manual tuning are often time-consuming, intuition-driven, and unlikely to discover the true optimal configuration within a vast search space. Automated methods like Grid Search or Random Search can be computationally prohibitive. This project explores the use of metaheuristic algorithms as an intelligent and efficient alternative to navigate this complex optimization landscape.

1.2 Project Goal

The primary objective of this project is to implement, analyze, and document the application of **Particle Swarm Optimization (PSO)** for evolving the hyperparameters of neural networks. We aim to demonstrate its effectiveness across a variety of machine learning tasks and data modalities. By establishing a baseline performance for each task and comparing it against a PSO-optimized model, we seek to provide a clear and nuanced evaluation of PSO's strengths, weaknesses, and practical utility in different scenarios.

1.3 Document Structure

This document is structured to guide the reader from the theoretical foundations to the final experimental results. Chapter 2 will provide a brief overview of Neural Networks and Particle Swarm Optimization. Chapter 3 will detail the six datasets used in our experiments and the specific preprocessing steps applied to each. Chapter 4 will present the experimental setup, results, and discussion for each dataset individually. Finally, Chapter 5 will synthesize these findings into a comprehensive conclusion about the role of PSO in evolving neural networks.

2 Theoretical Background

This chapter provides a concise overview of the fundamental concepts central to this project: Artificial Neural Networks (ANNs), the primary models we aim to optimize, and Particle Swarm Optimization (PSO), the metaheuristic algorithm used to perform the optimization.

2.1 Artificial Neural Networks

Artificial Neural Networks are computational models inspired by the structure and function of biological neural networks in the brain. They are composed of interconnected nodes, or ”neurons”, organized in layers. Each connection has an associated weight, which is adjusted during the training process to enable the network to learn complex patterns from data. This project utilizes two primary types of feed-forward ANNs.

2.1.1 Dense (Fully-Connected) Neural Networks

A Dense or Fully-Connected Network represents the most fundamental ANN architecture. In this structure, every neuron in a given layer is connected to every neuron in the subsequent layer. These networks are highly effective for tasks involving tabular data, where the relationships between input features are global rather than spatial. Our experiments on the NASA Exoplanets, S&P 500, Yu-Gi-Oh!, and Autolist datasets employ Dense networks to solve their respective classification and regression problems. The output layer is typically a ‘softmax‘ activation for multi-class classification or a linear activation for regression.

2.1.2 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a specialized class of neural networks designed primarily for processing grid-like data, such as images. Their architecture is inspired by the human visual cortex. Unlike Dense networks, CNNs use shared weights and local receptive fields to efficiently learn hierarchical patterns. The key layers include:

- **Convolutional Layer:** This layer applies a set of learnable filters (kernels) across the input data. Each filter is specialized to detect a specific feature, such as an edge, a color gradient, or a texture.
- **Pooling Layer:** Typically following a convolutional layer, the pooling layer reduces the spatial dimensions of the feature maps. This decreases computational complexity and creates a degree of translational invariance. Max Pooling is the most common form.
- **Flatten Layer:** After several convolutional and pooling blocks, this layer transforms the 2D feature maps into a 1D vector, preparing the data to be fed into standard Dense layers for final classification or regression.

Due to their effectiveness in learning spatial hierarchies, CNNs were the chosen architecture for the FMA audio genre classification (acting on Mel-spectrograms) and the CIFAR-100 image classification tasks.

2.2 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is a population-based stochastic optimization technique inspired by the social behavior of bird flocking or fish schooling. It is a metaheuristic, meaning it makes few assumptions about the problem being optimized and can search large candidate spaces for solutions.

The algorithm initializes a population of candidate solutions, referred to as a "swarm of particles." Each particle represents a potential solution to the optimization problem—in our case, a complete set of neural network hyperparameters (e.g., learning rate, number of neurons). Each particle has a position in the high-dimensional search space and a velocity.

The optimization proceeds iteratively. In each iteration, the position and velocity of each particle are updated based on three components:

1. **Inertia:** The particle's tendency to continue moving in its current direction.
2. **Personal Best (‘pbest‘):** The particle's memory of the best position it has personally discovered so far. This represents individual experience.
3. **Global Best (‘gbest‘):** The swarm's collective knowledge of the best position found by any particle in the entire population. This represents social learning.

By combining these influences, the swarm collaboratively explores the search space, gradually converging towards the global best position, which represents the optimal set of hyperparameters found during the search. PSO is particularly well-suited for this project because it does not require gradient information and can effectively explore complex, non-convex hyperparameter landscapes. For all experiments in this project, the standard default coefficients recommended by the `pyswarms` library (‘c1=0.5‘, ‘c2=0.3‘, ‘w=0.9‘) were used to ensure a consistent and balanced search between exploration and exploitation.

3 Datasets and Preprocessing

This chapter provides a detailed overview of the six datasets used for the experiments in this project. For each dataset, we describe its origin, characteristics, and the specific preprocessing and feature engineering steps that were applied to prepare it for modeling. The goal was to select a diverse range of tasks, including both classification and regression, on tabular, audio, and image data.

3.1 Yu-Gi-Oh! Card Attribute Classification

This task involves classifying the "Attribute" of a Yu-Gi-Oh! monster card based on its other features.

- **Source:** The data was custom-scraped from an online fan-made card database [1]. This represents a real-world scenario of building a dataset from an unstructured web source.
- **Format:** The raw data was collected and stored in a JSON file.
- **Characteristics:**
  - *Instances:* 8,500 monster cards.
  - *Features:* Name, Level, Type, ATK, DEF, and Description.
  - *Classes:* 7 unique "Attribute" classes (DARK, DIVINE, EARTH, FIRE, LIGHT, WATER, WIND).
  - *Identified Risks:* The dataset exhibits a severe class imbalance. The "DARK" attribute is the most common (2,380 instances), while the "DIVINE" attribute is extremely rare (only 5 instances), presenting a significant challenge for the classifier.
- **Preprocessing:**
  1. **Feature Engineering:** The card's "Primary Type" (e.g., "Warrior", "Fiend") was extracted from the multi-part "Type" string (e.g., "Warrior/Effect").
  2. **Feature Selection:** The "Name" and "Description" columns were dropped. The remaining features used for modeling were "Level", "ATK", "DEF" (numerical), and "Primary Type" (categorical).
  3. **Encoding and Scaling:** Categorical features were one-hot encoded, and numerical features were standardized using Scikit-learn's 'StandardScaler'. The target "Attribute" labels were label encoded.

3.2 NASA Exoplanet Classification

This task involves classifying celestial objects as exoplanets based on their physical and orbital characteristics.

- **Source:** This dataset was custom-scraped from the NASA Exoplanet Archive, a public data service [2].
- **Format:** The data was collected and stored in a CSV file.
- **Characteristics:**

  - *Instances:* Approximately 5,800 celestial objects.
  - *Features:* Includes numerical data such as planetary mass, radius, orbital period, and eccentricity, as well as categorical data like discovery method.
  - *Classes:* 4 planet types (‘Gas Giant‘, ‘Neptune-like‘, ‘Super Earth‘, ‘Terrestrial‘), presenting a multi-class classification problem.

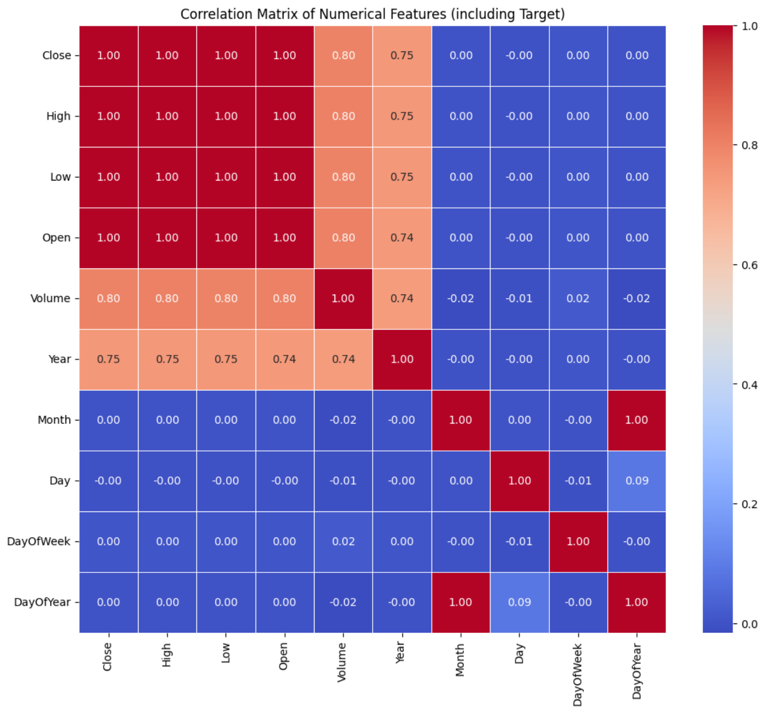
- **Preprocessing:**
  1. **Data Cleaning:** Custom string placeholders (e.g., ‘Not Found‘, ‘Unknown‘) were converted to standard NaN values. Rows with missing target labels were dropped.
  2. **Imputation:** Missing numerical values were imputed using the median, while missing categorical values were imputed using the most frequent value.
  3. **Encoding and Scaling:** Categorical features were one-hot encoded, and numerical features were standardized. The target labels were label encoded.

3.3 S&P 500 Stock Price Regression

This is a time-series regression task aimed at predicting the closing price of the S&P 500 index.

- **Source:** The historical data was acquired from Yahoo! Finance [3], a widely used source for financial market data.
- **Format:** The data was collected and stored in a JSON file.
- **Characteristics:**
  - *Instances:* Approximately 24,000 daily records.
  - *Features:* ‘Date‘, ‘Open‘, ‘High‘, ‘Low‘, ‘Close‘, and ‘Volume‘.
  - *Target:* ‘Close‘ (the daily closing price).
- **Preprocessing:**
  1. **Time-Series Feature Engineering:** The ‘Date‘ column was used to engineer new numerical features: ‘Year‘, ‘Month‘, ‘Day‘, ‘DayOfWeek‘, and ‘DayOfYear‘. The original ‘Date‘ column was then dropped.
  2. **Scaling:** All numerical features were standardized using ‘StandardScaler‘.

The correlation matrix for the preprocessed features, shown in Figure 1, confirms the strong linear relationships between the price variables (Open, High, Low, Close) and the engineered time-based features.



	Close	High	Low	Open	Volume	Year	Month	Day	DayOfWeek	DayOfYear
Close	1.00	1.00	1.00	1.00	0.80	0.75	0.00	-0.00	0.00	0.00
High	1.00	1.00	1.00	1.00	0.80	0.75	0.00	-0.00	0.00	0.00
Low	1.00	1.00	1.00	1.00	0.80	0.75	0.00	-0.00	0.00	0.00
Open	1.00	1.00	1.00	1.00	0.80	0.74	0.00	-0.00	0.00	0.00
Volume	0.80	0.80	0.80	0.80	1.00	0.74	-0.02	-0.01	0.02	-0.02
Year	0.75	0.75	0.75	0.74	0.74	1.00	-0.00	-0.00	0.00	-0.00
Month	0.00	0.00	0.00	0.00	-0.02	-0.00	1.00	0.00	-0.00	1.00
Day	-0.00	-0.00	-0.00	-0.00	-0.01	-0.00	0.00	1.00	-0.01	0.09
DayOfWeek	0.00	0.00	0.00	0.00	0.02	0.00	-0.00	-0.01	1.00	-0.00
DayOfYear	0.00	0.00	0.00	0.00	-0.02	-0.00	1.00	0.09	-0.00	1.00

Figure 1. Correlation Matrix for S&P 500 Features

3.4 Autolist Car Price Regression

This task involves predicting the price of a used car based on its listing details.

- **Source:** The data was custom-scraped from the public listings on Autolist.com [4].
- **Format:** The data was collected and stored in a JSON file.
- **Characteristics:**
  - *Instances:* 21,198 vehicle listings.
  - *Features:* A rich set including Name, Transmission, Drivetrain, Color, Condition, Body Style, and Mileage.
  - *Target:* “Price” (a continuous numerical value).
  - *Identified Risks:* The “Price” variable contains significant outliers (very high-priced luxury or rare vehicles), which can disproportionately affect regression metrics like RMSE.
- **Preprocessing:**
  1. **Feature Engineering:** The vehicle’s “Year” and “Make” (e.g., “Ford”, “Jeep”) were extracted from the “Name” string.
  2. **Feature Selection:** High-cardinality or identifier-like columns (e.g., ‘Location‘, ‘VIN‘, ‘Trim‘) and sparse-data columns were dropped to create a concise feature set.

3. **Handling Missing Values:** Rows with missing target (‘Price‘) or critical feature (‘Year‘, ‘Make‘) values were dropped. For other columns, missing values were imputed using the median (for numerical) or most frequent value (for categorical) within a Scikit-learn pipeline.
4. **Encoding and Scaling:** Categorical features were one-hot encoded, and numerical features were standardized.

3.5 FMA: Audio Genre Classification

This task involves classifying a music track's genre based on its audio content.

- **Source:** The Free Music Archive (FMA) dataset [5], a standard benchmark for Music Information Retrieval (MIR) research. The `fma_small` and `fma_metadata` subsets were used.
- **Format:** MP3 audio files and accompanying ‘.csv‘ metadata files.
- **Characteristics:**
  - *Instances:* 8,000 tracks.
  - *Classes:* 8 top-level genres (e.g., ‘Electronic‘, ‘Rock‘, ‘Hip-Hop‘), balanced with 1,000 tracks per genre.
  - *Identified Risks:* To manage computational requirements for this project, the audio clips were truncated from 30 seconds to 10 seconds. This reduction in duration limits the contextual information available for classification, making the task inherently more difficult.
- **Preprocessing:**
  1. **Feature Extraction:** This was the most critical step. Using the ‘librosa‘ library, each 10-second audio clip was converted into a **Mel-spectrogram**. A spectrogram is a 2D representation of the spectrum of frequencies in a sound as they vary with time. This process effectively transforms the audio problem into an image classification problem.
  2. **Normalization:** The resulting spectrograms were converted to a logarithmic (decibel) scale to normalize the power spectrum, a standard practice that improves model performance.

3.6 CIFAR-100 Image Classification

This is a classic computer vision task involving the classification of small images into a large number of classes.

- **Source:** The CIFAR-100 dataset [6], a standard academic benchmark provided by the Canadian Institute for Advanced Research. It was loaded directly via the ‘tensorflow.keras.datasets‘ API.
- **Format:** NumPy arrays of image pixel data and integer labels.
- **Characteristics:**
  - *Instances:* 60,000 color images (50,000 for training, 10,000 for testing).

  - *Image Size:* 32x32 pixels with 3 color channels (RGB).
  - *Classes:* 100 distinct object classes.
  - *Identified Risks:* The primary challenge is the high number of classes combined with the low resolution of the images, which requires a more powerful CNN architecture to effectively distinguish between fine-grained categories.

- **Preprocessing:**

  1. **Scaling:** The pixel values for each image, originally in the range [0, 255], were scaled to the range [0.0, 1.0] by dividing by 255. This normalization step is essential for stable and efficient training of neural networks.

4 Experimental Setup and Results

This chapter details the experimental process and presents the final results for each of the six datasets. It is important to note a key methodological distinction from some foundational neuroevolution approaches. Rather than using PSO to optimize the neural network's weights directly, this project employs a more modern and practical workflow: PSO is used to search for the optimal set of **hyperparameters** (e.g., network architecture, learning rate). For each candidate set of hyperparameters, the network itself is trained using a standard gradient-based optimizer (Adam).

The core methodology remained consistent: a baseline model was established, Particle Swarm Optimization (PSO) was used to search for optimal hyperparameters, and a final, optimized model was trained and compared against the baseline.

4.1 Yu-Gi-Oh! Card Attribute Classification

4.1.1 Model Architectures

A Dense Neural Network was used for this tabular classification task.

- **Baseline Model:** The network consisted of an input layer, two hidden layers with 64 and 32 neurons respectively, ReLU activation functions, and Dropout layers for regularization. The output layer used ‘softmax‘ activation for the 7 classes.
- **PSO-Optimized Model:** The architecture was determined by the PSO search, which optimized the number of neurons in the hidden layers, dropout rates, learning rate, and batch size.

4.1.2 PSO Configuration

The PSO was configured to search a 6-dimensional space with 20 particles over 30 iterations. The fitness function aimed to maximize the validation set's balanced accuracy score to account for severe class imbalance.

4.1.3 Comparative Results

The performance of the baseline and PSO-optimized models on the test set is summarized in Table 1. The results show a marginal trade-off, with the PSO model achieving a slightly higher balanced accuracy at the cost of overall accuracy, highlighting the difficulty of optimizing on this imbalanced dataset.

Metric	Baseline Model	PSO Optimized Model
Accuracy	0.4924	0.4894
Balanced Accuracy	0.3563	0.3588
Macro F1-score	0.3700	0.3800
MCC	0.3529	0.3483
Cohen's Kappa	0.3398	0.3386

Table 1. Results for Yu-Gi-Oh! Card Classification

The training history plots are presented in Figure 2. Both the baseline (left) and PSO-optimized (right) models show relatively flat learning curves after the initial epochs, which is characteristic of a challenging dataset where significant gains in accuracy are difficult to achieve.

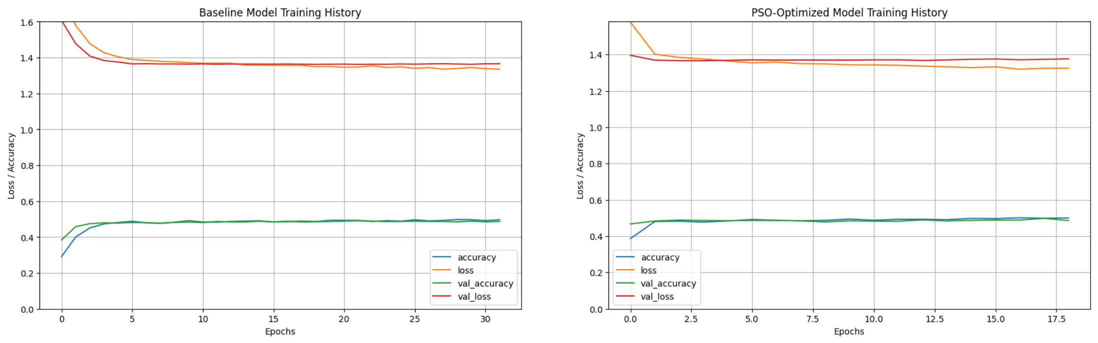
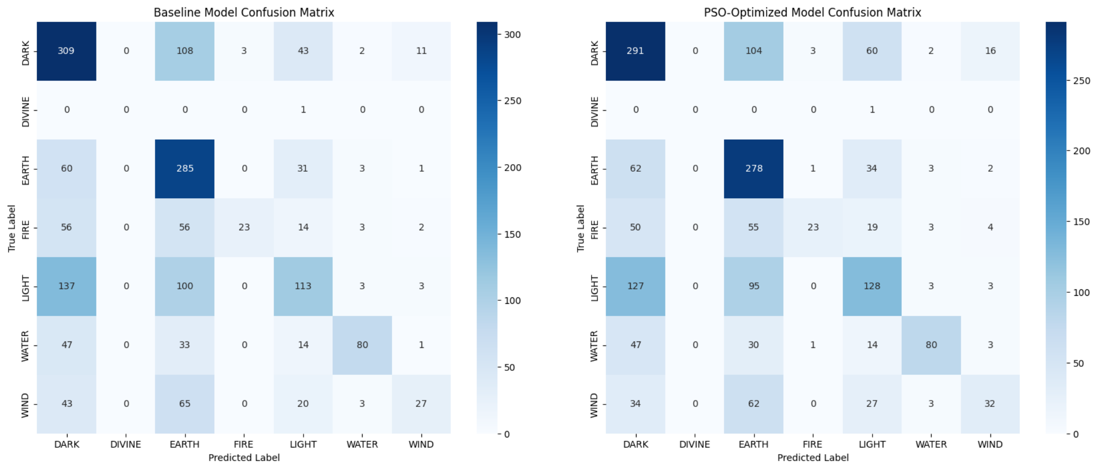


Figure 2. Training History for Yu-Gi-Oh! Classification

Figure 3 provides a visual comparison of the confusion matrices for both models. It illustrates how both models struggled with the rare "DIVINE" class and tended to misclassify other cards as the more dominant "EARTH" and "DARK" classes.



Baseline (True \ Predicted)	DARK	DIVINE	EARTH	FIRE	LIGHT	WATER	WIND
DARK	309	0	108	3	43	2	11
DIVINE	0	0	0	0	1	0	0
EARTH	60	0	285	0	31	3	1
FIRE	56	0	56	23	14	3	2
LIGHT	137	0	100	0	113	3	3
WATER	47	0	33	0	14	80	1
WIND	43	0	65	0	20	3	27

PSO-Optimized (True \ Predicted)	DARK	DIVINE	EARTH	FIRE	LIGHT	WATER	WIND
DARK	291	0	104	3	60	2	16
DIVINE	0	0	0	0	1	0	0
EARTH	62	0	278	1	34	3	2
FIRE	50	0	55	23	19	3	4
LIGHT	127	0	95	0	128	3	3
WATER	47	0	30	1	14	80	3
WIND	34	0	62	0	27	3	32

Figure 3. Confusion Matrices for Yu-Gi-Oh! Classification
Left: Baseline Model, Right: PSO-Optimized Model

4.2 NASA Exoplanet Classification

4.2.1 Model Architectures

A Dense Neural Network was employed for this task. The baseline architecture used two hidden layers (32 and 16 neurons) with ReLU activation and Dropout. The PSO model's architecture was evolved.

4.2.2 PSO Configuration

The search was configured with 10 particles and 10 iterations, optimizing the number of neurons, dropout rate, learning rate, and batch size to maximize validation accuracy.

4.2.3 Comparative Results

The optimization provided a clear and significant improvement over the already strong baseline, as detailed in Table 2. The PSO-optimized model shows superior performance across all metrics.

Metric	Baseline Model	PSO Optimized Model
Accuracy	0.9257	0.9436
Balanced Accuracy	0.8556	0.9105
Macro F1-score	0.8676	0.9159
MCC	0.8929	0.9185
Cohen's Kappa	0.8919	0.9181

Table 2. Results for NASA Exoplanet Classification

The training histories for both models are shown in Figure 4. The PSO-optimized model not only reaches a better validation accuracy but also demonstrates a more stable training curve.

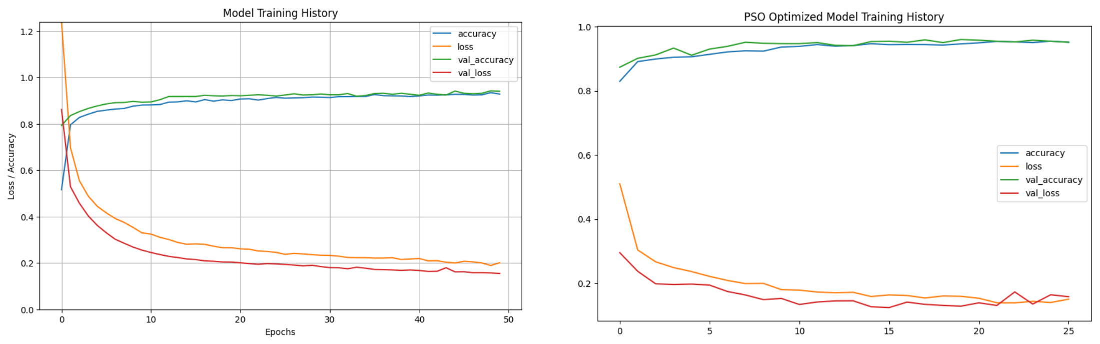


Figure 4. Training History for NASA Exoplanet Classification

The confusion matrices in Figure 5 further illustrate the improvement. The PSO-optimized model (right) shows fewer misclassifications overall, particularly reducing the number of 'Super Earth' planets mistaken for other types.

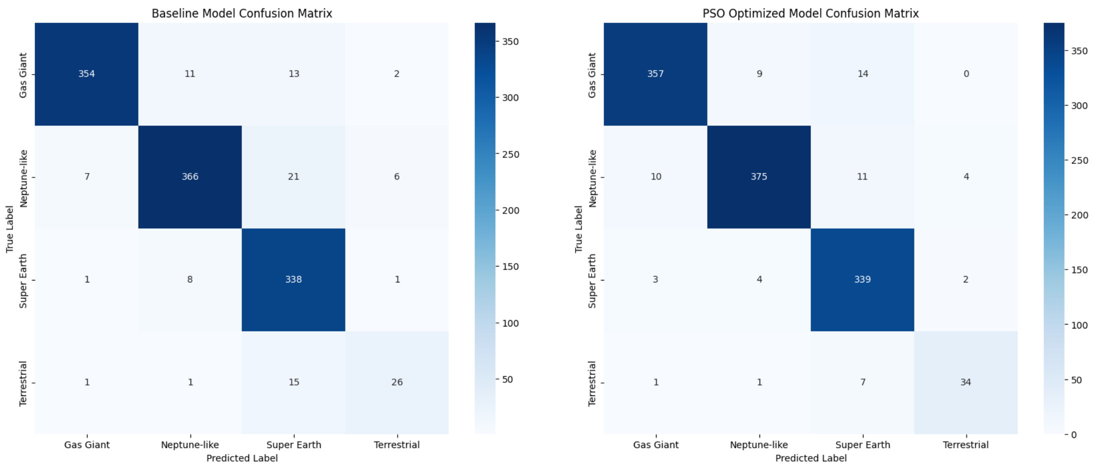


Figure 5. Confusion Matrices for NASA Exoplanet Classification

4.3 S&P 500 Stock Price Regression

4.3.1 Model Architectures

A Dense Neural Network was used, with the final layer being a single neuron with a linear activation function to output a continuous price value.

4.3.2 PSO Configuration

PSO was tasked with minimizing the Mean Squared Error (MSE) on the validation set by tuning the network's layer sizes, dropout rate, learning rate, and batch size. The search used 20 particles over 10 iterations.

4.3.3 Comparative Results

The S&P 500 data, being highly autocorrelated, was relatively easy for the models to predict, resulting in extremely high R-squared values for both. As shown in Table 3, PSO provided a slight reduction in MAPE.

Metric	**Baseline Model**	**PSO Optimized Model**
RMSE	20.28	20.13
MAE	14.22	14.46
R-squared (R^2)	0.9997	0.9997
MAPE	18.28%	17.57%

Table 3. Results for S&P 500 Regression

The training histories in Figure 6 show that both models converged very quickly to a low loss value, which is characteristic of a problem with strong predictive signals.

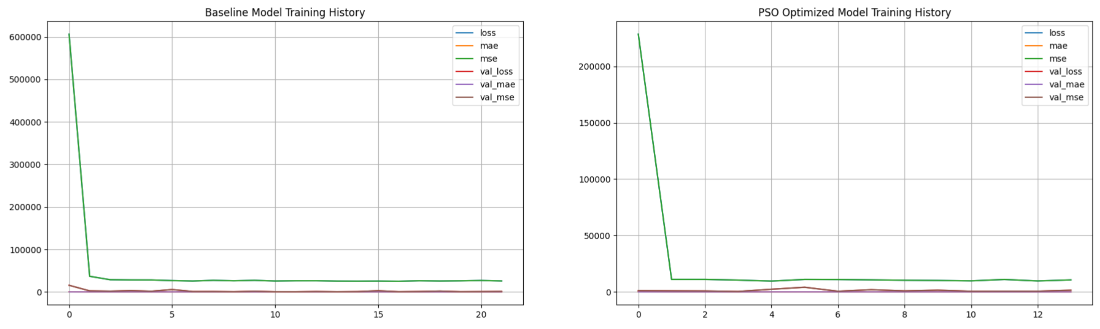


Figure 6. Training History for S&P 500 Regression

Figure 7 shows the Actual vs. Predicted values for both models. The points for both models fall very tightly along the diagonal line, indicating high accuracy, consistent with the R-squared value of approximately 0.9997.

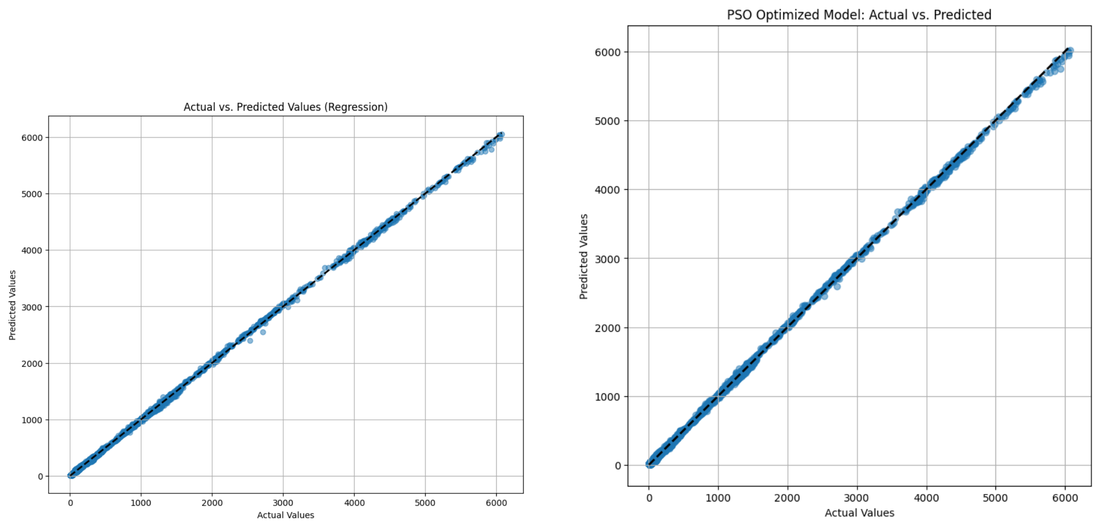


Figure 7. Actual vs. Predicted Values for S&P 500 Regression

4.4 Autolist Car Price Regression

4.4.1 Model Architectures

Similar to the S&P 500 task, a Dense network with a final linear output layer was used. The feature space was significantly larger due to one-hot encoding of many categorical features.

4.4.2 PSO Configuration

The PSO search (20 particles, 15 iterations) was configured to find hyperparameters that minimized the validation set MSE.

4.4.3 Comparative Results

The results, summarized in Table 4, revealed a complex performance trade-off. The PSO-optimized model achieved a better typical-case error (lower MAE and MedAE) but performed worse on outlier-sensitive metrics like RMSE and R^2.

Metric	**Baseline Model**	**PSO Optimized Model**
RMSE	66,498.13	78,908.99
MAE	21,050.61	20,431.55
R-squared (R^2)	0.6229	0.4690
MedAE	6,451.30	5,052.66

Table 4. Results for Autolist Car Price Regression

The regression evaluation plots are shown in Figures 8 and 9. The left plot for each model shows the predicted values vs. the actual values, while the right plot shows the distribution of residuals. The tighter clustering of points around the diagonal line for the bulk of the data in the PSO model (bottom) aligns with its superior Median Absolute Error.

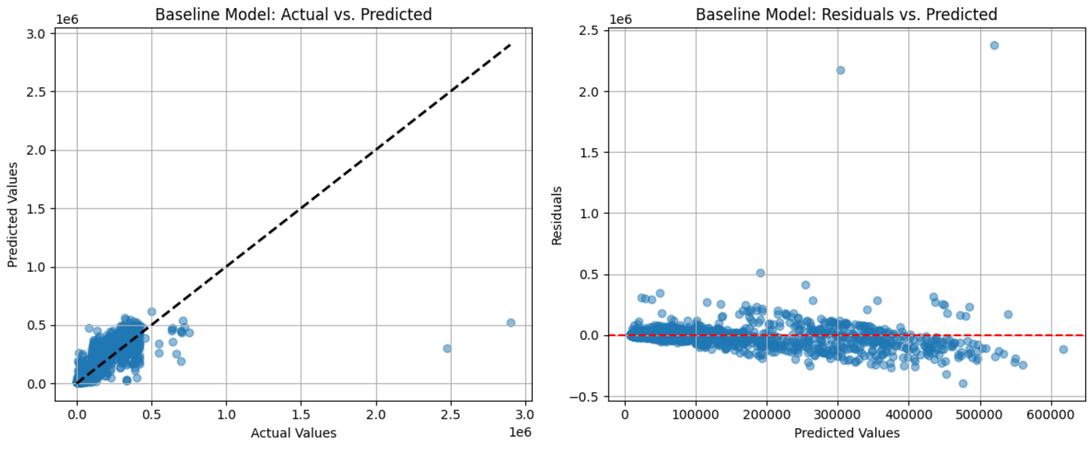


Figure 8. Regression Evaluation Plots for Autolist (Baseline)

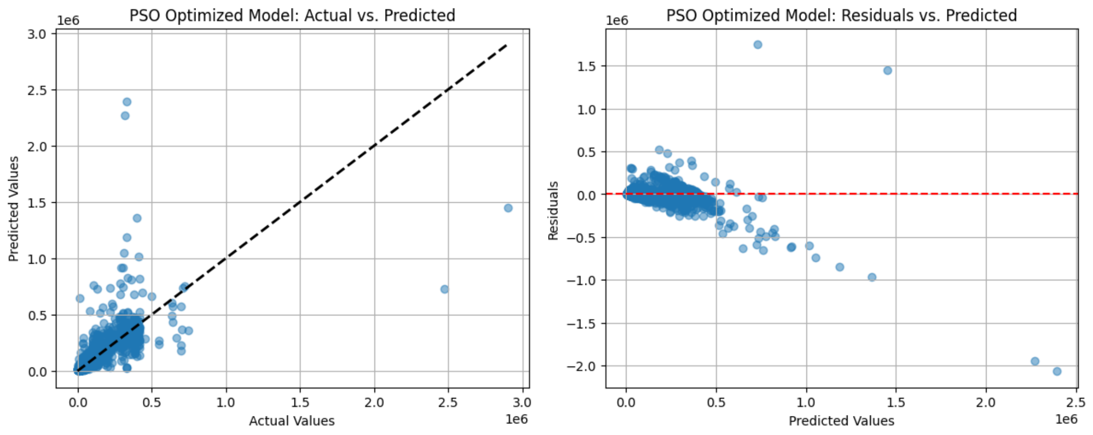


Figure 9. Regression Evaluation Plots for Autolist (PSO-Optimized)

4.5 FMA Audio Genre Classification

4.5.1 Model Architectures

A Convolutional Neural Network (CNN) was required for this task, operating on Mel-spectrograms derived from the audio clips. The baseline architecture consisted of two convolutional blocks followed by a dense head.

4.5.2 PSO Configuration

The PSO search (5 particles, 4 iterations) was configured to find the optimal number of filters in the convolutional layers, the number of units in the dense layer, and the learning rate, with the goal of maximizing validation accuracy.

4.5.3 Comparative Results

This experiment yielded the most dramatic results of the project. The baseline model failed to learn, whereas the PSO-optimized model showed a significant performance increase, as seen in Table 5.

Metric	Baseline Model	PSO Optimized Model
Accuracy	0.1251	0.2545
Balanced Accuracy	0.1256	0.2543
Macro F1-score	0.0302	0.1100
MCC	0.0052	0.1334
Cohen's Kappa	0.0007	0.0735

Table 5. Results for FMA Genre Classification

The confusion matrices in Figure 10 provide further evidence. The baseline matrix (left) shows predictions scattered almost randomly. The PSO-optimized matrix (right), while not perfect, provides a much stronger result, indicating that it is correctly classifying a significant number of tracks for each genre.

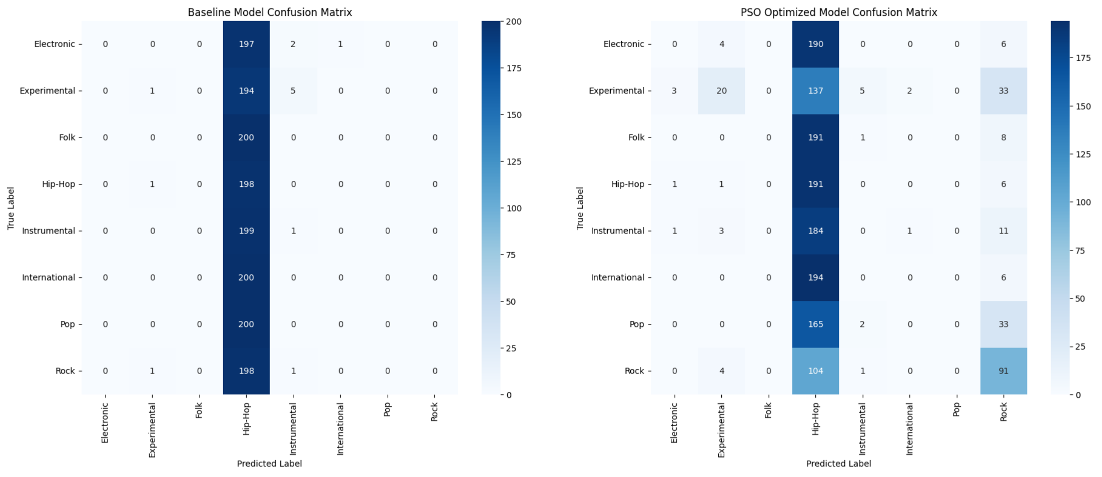


Figure 10. Confusion Matrices for FMA Genre Classification

4.6 CIFAR-100 Image Classification

4.6.1 Model Architectures

A deeper CNN, featuring three convolutional blocks with increasing filter depth and batch normalization, was used for this standard computer vision benchmark.

4.6.2 PSO Configuration

The PSO search (5 particles, 5 iterations) optimized the number of filters in each of the three blocks, the size of the final dense layer, and the learning rate.

4.6.3 Comparative Results

PSO provided a consistent, modest improvement over an already strong baseline.

Metric	Baseline Model	PSO Optimized Model
Accuracy	0.5897	0.5975
Balanced Accuracy	0.5897	0.5975
Macro F1-score	0.5868	0.5954
MCC	0.5857	0.5935
Cohen's Kappa	0.5856	0.5934

Table 6. Results for CIFAR-100 Image Classification

5 Conclusion

This project set out to investigate the application of Particle Swarm Optimization (PSO) for evolving neural networks. By testing this methodology across six diverse datasets, we gathered a rich set of results that paint a nuanced picture of PSO's capabilities, strengths, and limitations. This chapter synthesizes the individual findings from each experiment into a cohesive overall conclusion.

5.1 Synthesis of Results: The Versatile Roles of PSO

The primary finding of this project is that PSO is not a monolithic "improver" but a versatile tool whose role changes dramatically based on the nature of the problem it is applied to. We observed three distinct roles across our experiments.

5.1.1 Role 1: The Problem-Solver on Difficult Tasks

The most compelling demonstration of PSO's power was on the "FMA audio genre classification" task. The baseline model, when presented with challenging, information-limited 10-second audio spectrograms (reduced from the original 30-second duration for this experiment), completely failed to learn, achieving an accuracy of only 12.5%, equivalent to random chance. This failure was attributed to suboptimal default hyperparameters, particularly the learning rate, being unable to navigate the complex loss landscape.

In this scenario, PSO acted as a critical problem-solver. Even with a highly constrained search (20 total evaluations to manage CPU time and other time constraints), the algorithm successfully identified a much smaller learning rate and a different network capacity. This new configuration enabled the model to converge correctly, **doubling the accuracy to 25.5%**. This result powerfully illustrates that for difficult or unconventional problems, hyperparameter optimization is not merely beneficial; it can be the determining factor between a non-functional model and a working one.

5.1.2 Role 2: The Refiner on Well-Posed Problems

In contrast, on the **NASA Exoplanets** and **CIFAR-100** image classification tasks, the baseline models were already highly effective, achieving strong initial accuracies of 92.6% and 59.0%, respectively. These datasets were well-structured and the chosen baseline architectures were appropriate.

Here, PSO demonstrated its value as a fine-tuner. It did not discover a radically different solution but rather explored the local hyperparameter space to find minor adjustments that polished the model's performance. For both datasets, this resulted in a modest but consistent improvement across all evaluation metrics. This shows that even when a model is already performing well, metaheuristic search can still provide a valuable, measurable edge.

5.1.3 Role 3: The Revealer of Trade-offs on Complex Data

The experiments on the custom-scraped tabular datasets highlighted how PSO's behavior is influenced by underlying data characteristics.

On the **Autolist** car price regression task, the data contained significant price outliers. The PSO-optimized model achieved a lower typical-case error (better MAE and MedAE) but performed worse on outlier-sensitive metrics (RMSE, R^2). It essentially learned to be more accurate for the majority of cars at the cost of making larger mistakes on a few extreme cases.

This result reveals that PSO can be guided to optimize for different definitions of ”best”, and that a single ”best” model may not exist. Additionally, deducing from the results, we can say that outlier handling is of high and vital importance.

Similarly, on the **Yu-Gi-Oh!** card classification task, the severe class imbalance proved to be the dominant factor. While PSO offered marginal gains in some metrics, its impact was heavily constrained by the lack of data for rare classes. This underscores a fundamental principle: an optimization algorithm can only work with the information present in the data. No amount of hyperparameter tuning can fully compensate for a fundamentally imbalanced or low-signal dataset.

5.2 Future Work and Potential Improvements

Based on the insights gained during this project, several avenues for future work could be explored:

- **Advanced Outlier Handling:** For the regression tasks, explicitly handling outliers before training—for instance, by applying a logarithmic transformation to the target variable—would lead to more stable and robust models with higher R^2 values.
- **Expanded PSO Search:** The PSO runs in this project were constrained by available computational time. A more exhaustive search with more particles and iterations, particularly on a GPU, could potentially uncover even better hyperparameter configurations.
- **Data Augmentation:** For the CIFAR-100 task, implementing data augmentation techniques (random flips, rotations, crops) would artificially expand the training set and almost certainly lead to a significant boost in final accuracy for both the baseline and optimized models.
- **Comparing Optimization Algorithms:** A follow-up project could compare the performance and efficiency of PSO against other optimization methods like Bayesian Optimization or Genetic Algorithms on these same datasets.

5.3 Final Conclusion

This project successfully validated that Particle Swarm Optimization is a powerful and flexible method for evolving neural networks. Its true value lies not in a uniform ability to improve any model, but in its adaptability to different challenges. It can function as a critical problem-solver for difficult tasks, a precise fine-tuner for well-posed problems, and a tool for navigating complex performance trade-offs. The results clearly demonstrate that the success of any machine learning endeavor is a holistic interplay between the model architecture, the optimization of its hyperparameters, and the inherent characteristics of the data itself.

References

[1] Konami Digital Entertainment, *Yu-Gi-Oh! TRADING CARD GAME - CARD DATABASE*. Available: `https://www.db.yugioh-card.com/yugiohdb/`

[2] NASA Science, *NASA Exoplanet Archive*. Available: `https://science.nasa.gov/exoplanets/exoplanet-catalog/`

[3] Yahoo! Finance, *S&P 500 (^GSPC) Historical Data*. Available: `https://finance.yahoo.com/quote/%5EGSPC/history/`

[4] Autolist, *Used and New Car Search*. Available: `https://www.autolist.com/`

[5] M. Defferrard, K. Benzi, P. Vandergheynst, and X. Bresson, *FMA: A Dataset for Music Analysis*, 18th International Society for Music Information Retrieval Conference (ISMIR), 2017. Available: `https://github.com/mdeff/fma`

[6] A. Krizhevsky, *Learning Multiple Layers of Features from Tiny Images*, Tech Report, 2009. Available: `https://www.cs.toronto.edu/~kriz/cifar.html`

[7] M. Abadi et al., *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, 2015. Software available from `tensorflow.org`.

[8] L. J. N. M. B. E. G. C. M. L. P. Miranda, *PySwarms: a research toolkit for Particle Swarm Optimization in Python*, Journal of Open Source Software, 3(21), 433, 2018. Available: `https://github.com/ljvmiranda921/pyswarms`

[9] F. Pedregosa et al., *Scikit-learn: Machine Learning in Python*, Journal of Machine Learning Research, 12, pp. 2825-2830, 2011.

[10] B. McFee et al., *librosa: Audio and Music Signal Analysis in Python*, Proceedings of the 14th Python in Science Conference, 2015.

[11] Semantix, *Evolving Neural Networks with Particle Swarm Optimization.* Medium, 2019. Available: `https://medium.com/semantixbr/evolving-neural-networks-with-particle-swam-optimization-26a261f49d9f`

Appendices

The complete source code for all experiments, including data preprocessing scripts and Jupyter Notebooks for each dataset, is available in the public GitHub repository for this project. The repository provides the necessary code to replicate the findings presented in this document.

`https://github.com/bbandic1/Evolving-Neural-Networks-with-PSO`