University of Seville
School of Engineering

Master's Thesis:

Architectural Optimization of Dynamic Inception Modules in Convolutional Neural Networks using the Coral Reef Optimization Algorithm

Author:
- David Pineda Peña

Supervisors:
- Miguel Cárdenas Montes
- Miguel Ángel Gutiérrez Naranjo

November, 2024

Acknowledgments

To my supervisors, Dr. Miguel Cárdenas Montes and Dr. Miguel Ángel Gutiérrez Naranjo, who generously shared their time, insight, and expertise. They took a leap with me into this research journey, guiding and inspiring me along the way. whose guidance, insight, and expertise have been pivotal in shaping this work. I am deeply grateful for their mentorship, and I look forward to the potential of continuing our work together in the future.

To my family and friends, who witnessed my first steps in this field and have continued to stand by my side, offering endless companionship, encouragement, and understanding. Their unwavering support remains an invaluable source of strength and motivation on this journey.

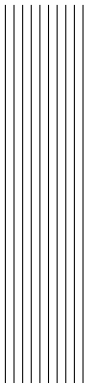
For all this and more, thank you.

Abstract

The rapid advancement of Deep Learning (DL) has led to increasingly complex neural network architectures in Artificial Intelligence (AI), often increasing computational requirements and environmental impact. This master's thesis presents a novel methodology for optimizing the architecture of **Inception modules** within Convolutional Neural Networks (CNNs) that process data at multiple scales using the **Coral Reef Optimization (CRO)** algorithm, a bio-inspired evolutionary approach. Aligning with **Green AI** principles that emphasize efficiency and sustainability, we integrate a dynamic Inception module capable of adjusting branches, depths, and filter sizes with the CRO algorithm to effectively explore and exploit the architectural search space. To promote smaller, resource-efficient architectures, we introduce a custom evaluation metric that balances accuracy and model complexity by penalizing excessive parameters. Experimental results on the MNIST dataset demonstrate that the optimized models achieve competitive performance, reducing the number of parameters by up to 40% while maintaining accuracy comparable to standard models. This work contributes to the development of sustainable AI models and provides a foundation for future research in efficient neural architecture optimization.

Resumen

El rápido avance del Aprendizaje Profundo (*Deep Learning*, DL) ha llevado a arquitecturas de redes neuronales en Inteligencia Artificial (IA) cada vez más complejas, lo que frecuentemente aumenta los requerimientos computacionales y el impacto ambiental. Este Trabajo Fin de Máster (TFM) presenta una metodología novedosa para optimizar la arquitectura de los módulos **Inception** dentro de Redes Neuronales Convolucionales (CNNs) que procesan datos a múltiples escalas, utilizando el algoritmo de **Optimización de Arrecifes de Coral (CRO)**, un enfoque evolutivo bioinspirado. En alineación con los principios de **IA Verde**, que destacan la eficiencia y la sostenibilidad, integramos un módulo Inception dinámico, capaz de ajustar ramas, profundidades y tamaños de filtro, junto con el algoritmo CRO para explorar y explotar de manera efectiva el espacio de búsqueda arquitectónica. Para promover arquitecturas más pequeñas y eficientes en recursos, introducimos una métrica de evaluación personalizada que equilibra la precisión y la complejidad del modelo, penalizando el exceso de parámetros. Los resultados experimentales en el conjunto de datos MNIST demuestran que los modelos optimizados logran un rendimiento competitivo, reduciendo el número de parámetros hasta en un 40% mientras mantienen una precisión comparable a los modelos estándar. Este trabajo contribuye al desarrollo de modelos de IA sostenibles y proporciona una base para futuras investigaciones en la optimización eficiente de arquitecturas neuronales.



Contributions

This master's thesis makes the following unique contributions to the field of neural network optimization:

1. **Development of an Efficient CNN Optimization Methodology**: Proposed a novel approach that combines the Coral Reef Optimization (CRO) algorithm with dynamic Inception modules to optimize Convolutional Neural Network (CNN) architectures efficiently, addressing both performance and resource constraints.

2. **Integration of Non-Square Kernels in Inception Modules**: Introduced and utilized non-square convolutional kernels within the Inception architecture to capture features at different scales and orientations, enhancing the model's representational capacity and improving classification performance.

3. **Creation of a Custom Evaluation Metric**: Developed a novel evaluation metric that balances accuracy with model complexity by incorporating the logarithm of the total number of parameters, promoting the selection of smaller, more efficient architectures in line with Green AI principles.

4. **Achievement of Highly Efficient Architectures with Broader Applicability**: Demonstrated that the optimized architectures obtained through the proposed methodology are up to two orders of magnitude smaller than top-performing models while maintaining competitive accuracy. This significant reduction in model size not only contributes to resource efficiency but also opens the door to applying this approach to other problems with high computational demands. By enabling the optimization of neural networks for computationally intensive tasks, this work paves the way for developing efficient AI solutions across various domains where resource constraints and carbon footprint are a critical concern.

Contents

List of Figures

1 Introduction

1.1 Background and Motivation

The field of *Deep Learning* (DL) has revolutionized *Artificial Intelligence* (AI), enabling breakthroughs in various domains such as computer vision, natural language processing, and speech recognition [1]. *Convolutional Neural Networks* (CNNs), a class of deep neural networks particularly effective for image data, have been instrumental in advancing computer vision tasks, achieving remarkable performance in image classification, object detection, and segmentation [2], [3].

As CNN architectures have evolved, there has been a tendency towards increasing depth and complexity to enhance performance. Models like ResNet [3] and GoogLeNet [4] have demonstrated that architectural innovations can lead to significant improvements in accuracy. However, the pursuit of higher performance often comes at the expense of increased computational resources and energy consumption. This trend raises concerns about the environmental impact and sustainability of deploying large-scale deep learning models, especially with the growing demand for AI applications.

Additionally, designing optimal neural network architectures is a complex task due to the vast number of possible configurations. Traditional methods rely on manual tuning and expert intuition, which may not explore the full range of potential architectures.

1.2 Green AI and the Need for Efficient Models

The concept of *Green AI* has emerged in response to the escalating computational costs and environmental impact associated with deep learning [5]. Green AI refers to AI research that yields novel results while considering computational efficiency, promoting environmentally friendly approaches.

The increasing computational demands of deep learning models have led to significant energy consumption and carbon emissions, raising concerns about the sustainability of AI development [6]. For instance, training a single large-scale model can emit as much carbon dioxide as five cars over their lifetimes [6].

Green AI advocates for the development of AI models that are not only accurate but also efficient in terms of computational resources and energy consumption. By emphasizing the reduction of energy usage and carbon footprint, Green AI encourages researchers and practitioners to consider the environmental implications of their models.

Key principles of Green AI include:

- **Energy Efficiency**: Minimizing the energy required for training and inference to reduce environmental impact [6].
- **Transparency**: Reporting the computational cost and carbon footprint of AI research to promote awareness and accountability [5].
- **Performance-Efficiency Trade-off**: Balancing model accuracy with computational efficiency to achieve sustainable AI development [5].

By adopting these principles, the AI community can work towards reducing the environmental impact of AI technologies and promoting sustainability.

1.3 GoogLeNet and the Inception Module

GoogLeNet, introduced by Szegedy et al. [4], marked a significant milestone in CNN architecture design. The core innovation of GoogLeNet is the *Inception module*, which allows the network to capture multi-scale features by performing parallel convolutions with different filter sizes within the same layer.

The traditional Inception module processes input data through multiple branches, each with different kernel sizes (e.g., 1×1, 3×3, 5×5), and concatenates the outputs along the channel dimension. Additionally, a pooling operation (e.g., 3×3 max pooling) is often

included as one of the branches. This structure enables the network to capture both local and global features at various scales, improving its ability to represent complex patterns.

Figure 1.1 illustrates the architecture of the traditional Inception module.

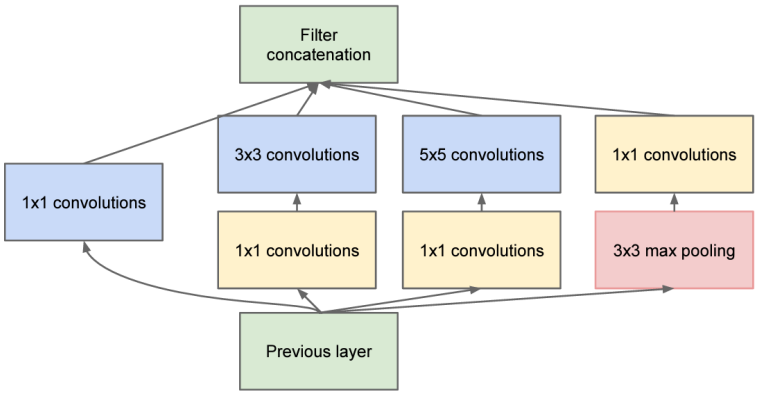


Figure 1.1: Architecture of the traditional Inception module as introduced in GoogLeNet [4].

Over the years, several versions of the Inception architecture have been proposed, such as Inception-v2, Inception-v3 [7], and Inception-v4 [8], each introducing refinements to improve performance and efficiency.

The architectural changes introduced by *Inception-v3* include:

- **Factorized Convolutions**: Inception-v3 replaces larger convolutions with smaller, factorized ones. For example, instead of a 5×5 convolution, it uses two consecutive 3×3 convolutions, reducing computational cost and allowing for deeper networks [7]. Figures 1.2 and 1.3 illustrate the differences.

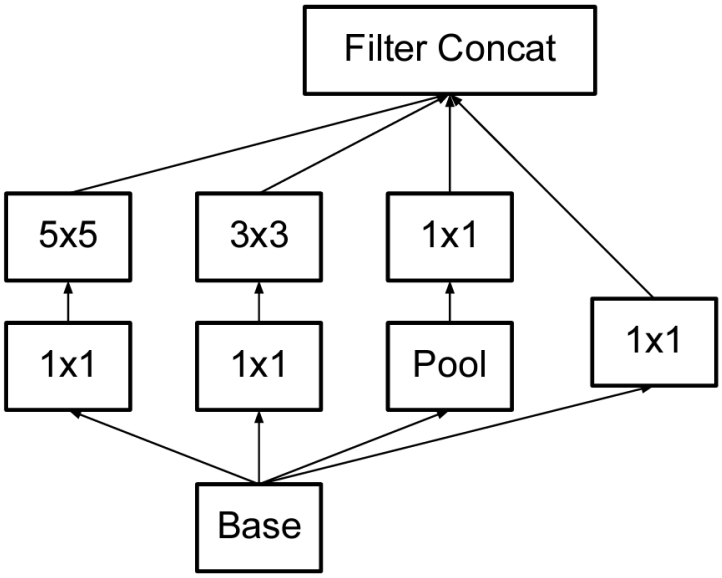


Figure 1.2: Original Inception module. Reproduced from [7].

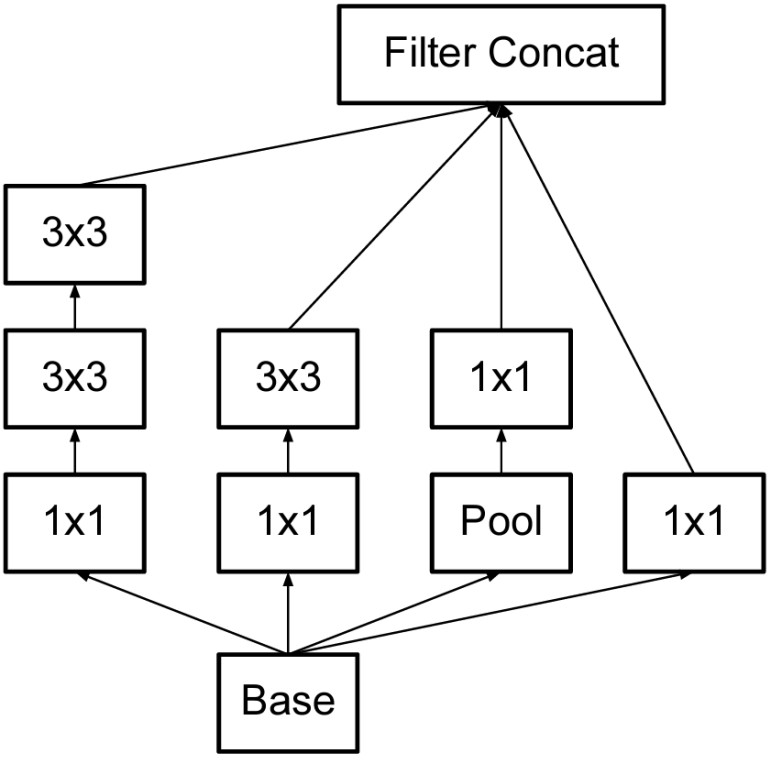


Figure 1.3: Inception module where each 5×5 convolution is replaced by two 3×3 convolutions. Reproduced from [7].

- **Asymmetric Convolutions**: Further computational efficiency is achieved by factorizing convolutions into asymmetric convolutions. For instance, a 3×3 convolution can be replaced with a 1×3 convolution followed by a 3×1 convolution, reducing parameters and speeding up computation.

- **Auxiliary Classifiers with Regularization**: Inception-v3 includes auxiliary classifiers to mitigate the vanishing gradient problem. These classifiers incorporate batch normalization and label smoothing to enhance regularization and aid convergence:

    1. **Batch Normalization**: Extensive use of batch normalization [9] accelerates training and stabilizes the model by normalizing layer inputs, reducing internal covariate shift.
    2. **Label Smoothing**: Label smoothing [7] is introduced as a regularization technique to prevent the model from becoming overly confident by assigning a small probability to incorrect labels during training, improving generalization.

The evolution from the traditional Inception module to Inception-v3 reflects a shift towards deeper and more efficient architectures. Despite these advancements, the design of Inception modules has been largely manual. Researchers have selected the number of branches, filter sizes, and other architectural parameters based on intuition and empirical testing, without exhaustively exploring all possible configurations.

The manual design process limits the exploration of the vast architectural space. Given the numerous combinations of branch numbers, filter sizes, and layer depths, many potentially optimal configurations remain undiscovered. This limitation underscores the need for automated methods to optimize the Inception module architecture efficiently, enabling the discovery of novel configurations that could offer better performance or efficiency.

1.4 Evolutionary Metaheuristic Algorithms and Coral Reef Optimization

Metaheuristic algorithms are high-level strategies designed to solve complex optimization problems by efficiently exploring large search spaces [10]. Inspired by natural processes, these algorithms are particularly useful when traditional optimization techniques are infeasible due to problem complexity.

Evolutionary algorithms, a subset of metaheuristics, mimic natural evolution to evolve solutions over successive generations [11]. They utilize mechanisms such as selection, crossover, and mutation to explore the search space and have been successfully applied across various fields.

The *Coral Reef Optimization* (CRO) algorithm is a bio-inspired evolutionary algorithm that simulates the reproduction and settlement processes of corals in a reef ecosystem [12]. CRO balances exploration and exploitation while maintaining diverse solution configurations within the search space.

In a natural coral reef ecosystem, corals reproduce both sexually and asexually, with larvae competing for space to settle on the reef. Predation and environmental factors influence coral survival, promoting biodiversity and adaptability.

The CRO algorithm models these processes as follows:

- **Initialization**: The reef is initialized with a population of corals (candidate solutions), each occupying a space in a reef matrix.
- **External Sexual Reproduction (Broadcast Spawning)**: Pairs of corals release gametes into the water column, where fertilization occurs randomly. This process is modeled by selecting pairs of corals to exchange genetic information through crossover, creating new larvae (offspring) that combine traits from both parents, enhancing exploration.
- **Internal Sexual Reproduction (Brooding)**: Corals not involved in broadcast spawning undergo brooding. Each coral produces a larva through mutation of its own solution representation. The mutation process introduces variability by altering architectural parameters, allowing the coral to explore new solutions close to its current state.
- **Asexual Reproduction (Budding)**: Some corals reproduce asexually through budding, creating identical or slightly mutated copies of themselves. This process

allows exploitation of good solutions by preserving and slightly varying high-fitness individuals.

- **Larvae Settlement**: New larvae attempt to settle in the reef, occupying free spaces or replacing less fit corals based on fitness levels.
- **Predation and Catastrophes**: Low-fitness corals are probabilistically removed, simulating predation and environmental events, preventing premature convergence and maintaining diversity.

Figure 1.4 presents the flow diagram of the CRO algorithm, illustrating the sequence of operations within the reef ecosystem.

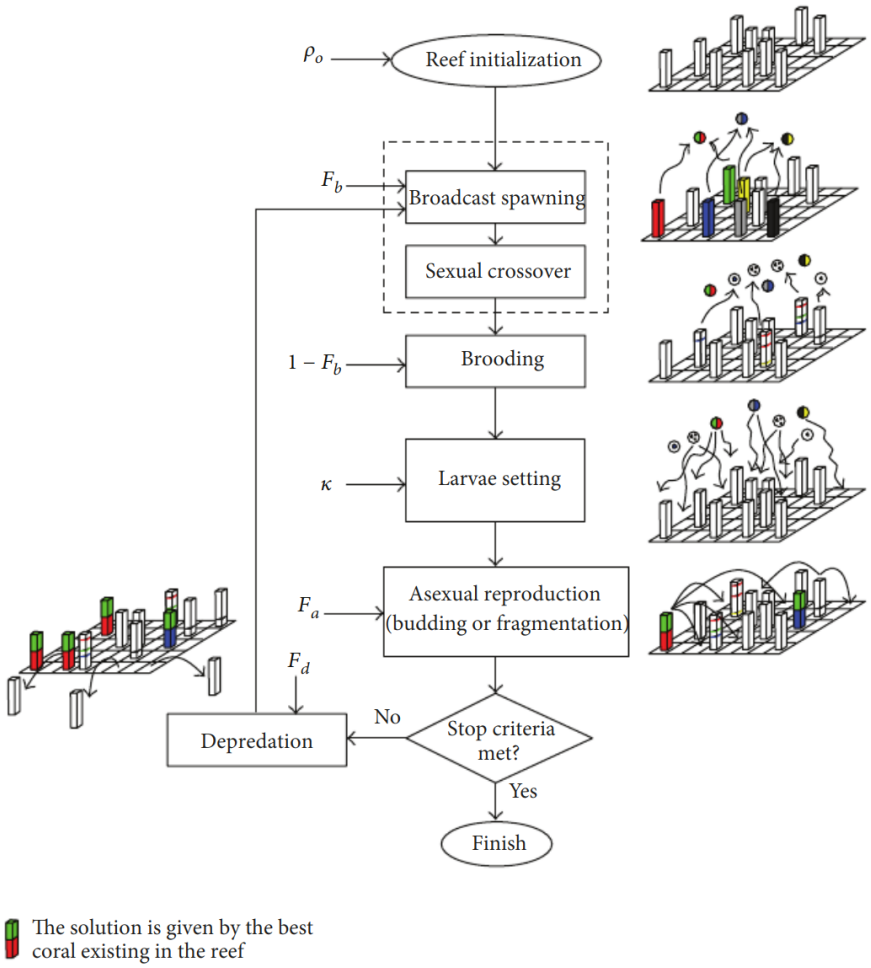


Figure 1.4: Flow diagram of the CRO algorithm as introduced in [12]

By simulating these ecological processes, CRO effectively searches the solution space for optimal or near-optimal solutions. Its ability to balance diversification (exploration) and intensification (exploitation) makes it suitable for complex optimization tasks.

Since its introduction, CRO has been successfully applied in various optimization problems. For instance, Durán-Rosal et al. [13] employed a statistically driven CRO algorithm to achieve optimal size reduction of time series data, minimizing information loss.

In this project, we apply the CRO algorithm to optimize the architecture of the Inception module within CNNs. By encoding different architectural configurations as corals, the algorithm explores a wide range of designs. The three reproduction processes allow for diverse search strategies:

- **Broadcast Spawning**: Enhances exploration by combining genetic material from different corals, potentially creating entirely new architectures.
- **Brooding**: Allows individual corals to explore variations of themselves through mutation, refining solutions in their vicinity.
- **Budding**: Exploits existing good solutions by cloning the fittest corals, preserving their architecture for the next generation.

The settlement and predation mechanisms ensure that only the most promising architectures survive to subsequent generations, maintaining a balance between exploration of new solutions and exploitation of known good ones.

This approach enables automated discovery of efficient Inception module configurations that may not be apparent through manual design. Leveraging natural optimization strategies of coral reef ecosystems, we aim to enhance CNN performance while adhering to Green AI principles by promoting resource-efficient architectures.

1.5 Research Objectives

The primary objective of this master's thesis is to develop a methodology that integrates the CRO algorithm with dynamic Inception modules to optimize CNN architectures efficiently.

Specific goals include:

1. **Develop a CRO-Based Optimization Framework**: Create a framework utilizing the CRO algorithm to automatically optimize the Inception module architecture, aiming to improve performance while reducing computational complexity.

2. **Incorporate Non-Square Kernels**: Introduce non-square convolutional kernels within Inception modules to enhance the network's ability to capture features at different scales and orientations. Traditional Inception modules require all branches to produce feature maps of the same spatial dimensions for concatenation. Incorporating non-square kernels poses a challenge in maintaining consistent dimensions, as they can alter height and width differently. Overcoming this involves careful design and padding strategies to ensure outputs align correctly. Successfully integrating non-square kernels enables the network to learn diverse and anisotropic features, capturing patterns elongated in specific directions, improving recognition of complex patterns and textures.

3. **Design a Custom Evaluation Metric**: Develop a novel evaluation metric balancing model accuracy with complexity, incorporating a penalty term based on the logarithm of the number of parameters to promote efficient architectures in line with Green AI principles.

4. **Achieve Highly Efficient Architectures with Broader Applicability**: Demonstrate that the optimized architectures obtained are significantly smaller than top-performing models while maintaining competitive accuracy. This reduction contributes to resource efficiency and enables application of this approach to other problems with high computational demands. By optimizing neural networks for computationally intensive tasks, this work paves the way for efficient AI solutions across domains where resource constraints are critical.

2 Materials and Methods

This chapter details the methodologies and materials employed in this research to optimize Convolutional Neural Network (CNN) architectures using the Coral Reef Optimization (CRO) algorithm. We begin by discussing related works that laid the foundation for this project. Then, we introduce the design of the dynamic Inception module, highlighting how its parameters are adapted for optimization. We delve into the implementation of the CRO algorithm, explaining how it is adapted to effectively search the architectural space, with a focus on the representation of solutions within the algorithm. Visualization techniques for the Inception modules are presented to aid in understanding the evolved architectures. Finally, we introduce a novel custom evaluation metric that balances model accuracy with complexity, aligning with Green AI principles.

2.1 Related Works

The current project builds upon previous studies that explored optimization techniques for neural network architectures with a focus on Green AI principles. These works progressively increased the flexibility and efficiency of Inception modules, laying the groundwork for the methodology presented in this thesis.

2.1.1 Optimizing Neural Network Architectures with Population-Based Incremental Learning

Vasco-Carofilis et al. [14] addressed the challenge of selecting optimal hyperparameters in neural network design, particularly within complex modules like the Inception module. They leveraged *Population-Based Incremental Learning* (PBIL) [15], an algorithm that combines aspects of genetic algorithms and competitive learning. PBIL maintains

a probability vector that guides the generation of a population of individuals, in this case, configurations of neural network hyperparameters. Over multiple generations, the probability vector is updated based on the performance of the top individuals, incrementally converging towards a configuration that minimizes prediction error while optimizing resource efficiency.

Key aspects of their methodology included:

- **Binary Encoding and Gray Coding**: Hyperparameters were encoded in binary to facilitate evolutionary operations. To avoid *Hamming cliffs*, they employed Gray coding, ensuring smooth transitions between hyperparameter values and preventing premature convergence.
- **Application to the Inception Module**: PBIL was used to optimize the Inception-A block, exploring kernel sizes, pooling layers, and filter configurations. The optimization focused on minimizing redundant computations and evaluating only valid configurations to reduce computational waste.
- **Low-Carbon Footprint Optimization**: The approach emphasized energy-efficient AI, highlighting PBIL's capacity to explore high-quality configurations without exhaustive searches, contributing to sustainable AI development.

Using the MNIST dataset as a benchmark, their optimized Inception-A block achieved high accuracy with a reduced carbon footprint, validating the effectiveness of PBIL for neural network optimization.

2.1.2 Improving PBIL Optimization of the Inception Module

García-Victoria et al. [16] built upon this work by proposing an enhanced methodology to optimize the hyperparameters of the Inception-A block using PBIL. They introduced a special codification scheme to avoid generating incompatible kernel sizes in the generated individuals, ensuring that all candidate architectures were valid and could be effectively evaluated.

Key contributions of their study included:

- **Special Codification for Kernel Sizes**: By implementing a codification method that prevented incompatible kernel sizes from being generated, they ensured the feasibility of all individuals in the population, thereby increasing the efficiency of the optimization process.

- **Single-Epoch Training for Optimization**: To reduce computational cost and carbon footprint, they evaluated each candidate model by training it for only a single epoch during the hyperparameter optimization phase. This approach provided sufficient feedback to the PBIL algorithm while significantly reducing execution time.

- **Enhanced Flexibility**: The study allowed for greater flexibility in the Inception module by optimizing a wider range of hyperparameters, further exploring the architectural space.

Their work demonstrated that careful design of the encoding scheme and evaluation strategy in PBIL could lead to more efficient optimization of complex neural network modules, further aligning with Green AI objectives.

2.1.3 Implications for the Current Work

These previous studies laid the foundation for the current project by demonstrating the effectiveness of evolutionary algorithms in optimizing neural network architectures while considering computational efficiency and environmental impact. The advancements made in optimizing the Inception module using PBIL, especially the special codification to avoid invalid configurations and the use of single-epoch training for evaluation, highlighted the potential of evolutionary strategies in reducing computational costs and improving model performance.

Building upon these methodologies, the current work employs the CRO algorithm to further enhance the architectural optimization of Inception modules in CNNs. The choice of CRO was motivated by the student's prior experience with the algorithm in his undergraduate final degree project, where CRO was successfully applied to optimize neural network weights and biases in an Extreme Learning Machine[17] model. CRO's ability to maintain diverse configurations of solutions within the same search space makes it particularly suitable for optimizing complex neural network architectures. By integrating dynamic architectural design with evolutionary optimization and leveraging lessons learned from previous PBIL implementations, this project aims to develop models that are both effective and resource-efficient, advancing the application of nature-inspired algorithms in deep learning.

2.2 Dynamic Inception Module Design

2.2.1 Architecture Overview

Convolutional Neural Networks (CNNs) are a class of deep neural networks commonly used for analyzing visual imagery. A key component of CNN architectures is the *Inception module*, initially introduced in the GoogLeNet architecture [4]. The Inception module allows the network to capture multi-scale features by processing the input through multiple parallel paths, each applying convolutions with different kernel sizes (e.g., 1×1, 3×3, 5×5), and then concatenating the outputs along the channel dimension. This structure enables the network to extract features at various scales and increases the diversity of features learned at each layer.

In our work, we implement a **dynamic** version of the Inception module, where architectural parameters are not fixed but are subject to optimization by the CRO algorithm. By allowing parameters such as the number of branches, branch depths, and filter sizes to vary, we enable exploration of a vast architectural space that could lead to more efficient and effective network designs.

2.2.2 Parameterization of the Inception Module

Each Inception module is defined by the following parameters:

- **Number of Branches**: The total number of parallel convolutional paths. Increasing the number of branches can enhance the module's ability to capture diverse features.
- **Branch Depth**: The number of layers (depth) in each branch. Deeper branches can capture more abstract and complex features.
- **Filter Sizes**: The kernel sizes for convolutions in each layer, allowing for *non-square kernels* (e.g., 3×5, 5×7). Varying filter sizes affects the receptive field of the convolutions, enabling the detection of features of different shapes and orientations.
- **Filter Channels**: The number of output channels (filters) for each convolutional layer, impacting the capacity of the network to learn various features.
- **Pooling Layers**: Optional inclusion of pooling layers within branches, which can reduce spatial dimensions and help capture invariant features.

By varying these parameters, we create a diverse set of possible architectures that the CRO algorithm can explore. This flexibility allows the optimization process to discover configurations that balance performance and efficiency.

2.2.3 Implementation Details

The dynamic Inception module is implemented using *PyTorch*, an open-source machine learning library widely used for developing and training neural network models. PyTorch provides flexibility and ease of use, which is essential for implementing dynamic architectures.

Branch Construction

Each branch in the Inception module is constructed as a sequence of layers:

1. **Convolutional Layers**: Each layer applies a convolution operation with specified kernel sizes and channels. *Padding* is calculated to maintain spatial dimensions when necessary, ensuring that feature maps from different branches can be concatenated.
2. **Pooling Layers**: Optionally added after convolutional layers to reduce spatial dimensions and capture invariant features. Max pooling is used within our project.

Non-Square Kernels

Traditional convolutional layers often use square kernels (e.g., 3×3). However, we incorporate **non-square kernels** to enable the network to capture features that vary more prominently in one dimension than the other. This flexibility enhances the representational capacity of the network by allowing the detection of elongated or directional patterns, which can be particularly useful for recognizing features like strokes in handwritten digits.

Non-square kernels have been effectively used in previous works to capture directional information [7], although their implementation was purely manual. By automating this through the CRO algorithm, we can systematically explore a wider range of kernel shapes.

Forward Pass and Output Concatenation

During the forward pass:

1. The input tensor is passed through each branch independently.

2. Outputs from all branches are adjusted to have matching spatial dimensions, if necessary, using appropriate padding, stride adjustments, or adaptive pooling.

3. The adjusted outputs are concatenated along the channel dimension to form the final output of the Inception module.

Ensuring consistent spatial dimensions across branches is crucial for the concatenation step. We carefully design the convolutional and pooling layers to maintain alignment, which is essential for the integrity of the network architecture.

2.2.4 Integration with the Overall Model

The dynamic Inception module is integrated into a CNN designed for the MNIST dataset [18], which consists of 28×28 grayscale images of handwritten digits from 0 to 9. The overall architecture of the CNN includes:

- **Stem**: Initial convolutional and pooling layers that process the raw input images and extract basic features.

- **Dynamic Inception Module**: The core component of the network, whose architecture is optimized by the CRO algorithm.

- **Head**: Comprises global average pooling and fully connected layers that produce the final classification output into the 10 digit classes.

Figure 2.1 illustrates the architecture with our proposed dynamic Inception module.

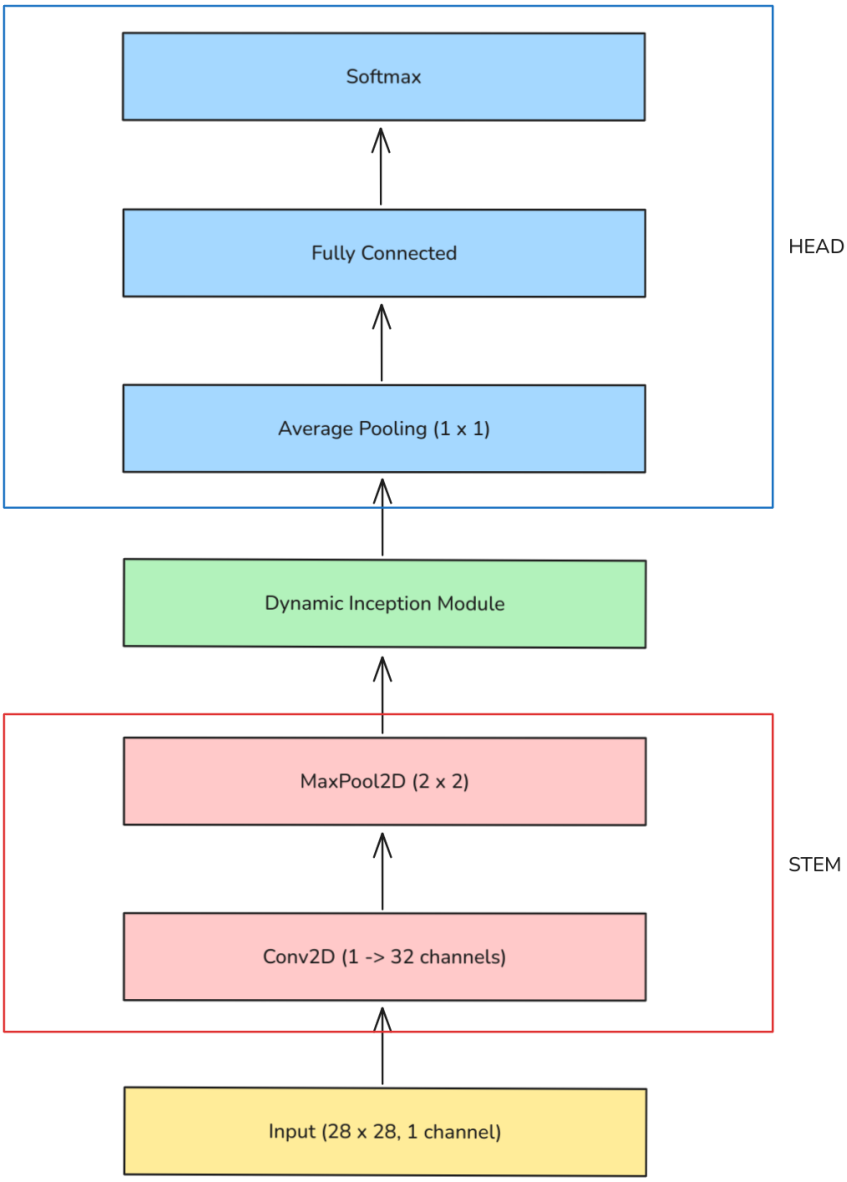


Figure 2.1: Overall CNN architecture design for MNIST classification problem with Dynamic Inception Module.

The number of output channels from the Inception module is calculated based on the output channels of each branch's last layer, ensuring compatibility with subsequent layers in the network.

2.3 Coral Reef Optimization Algorithm

2.3.1 Overview

The CRO algorithm is a bio-inspired metaheuristic that simulates the natural processes of coral reefs, including reproduction, settlement, and predation [12]. Metaheuristic algorithms are high-level, problem-independent algorithmic frameworks that guide the search process towards optimal solutions [19]. They are particularly effective for complex optimization problems where traditional methods may be impractical due to the size or nature of the search space.

In our implementation, we adapt the CRO algorithm to optimize the architecture of Inception modules within CNNs. By encoding different architectural configurations as corals, the algorithm efficiently searches the vast architectural space to find configurations that balance performance with computational efficiency.

2.3.2 Solution Representation

In the context of the CRO algorithm, each **coral** in the reef represents a candidate solution, a specific configuration of the dynamic Inception module within the CNN architecture. The representation of these solutions is crucial for the effective operation of the algorithm, particularly during reproduction and mutation processes.

Each solution (coral) is encoded as a data structure that encapsulates all the parameters defining the Inception module's architecture. The key components of this representation are:

- **Number of Branches**: An integer specifying how many parallel branches are present in the Inception module.
- **Branches Parameters**: A list of dictionaries, where each dictionary contains the parameters for one branch:
  - **Depth**: The number of layers (convolutional and pooling) in the branch.
  - **Filter Sizes**: A list of tuples, each representing the height and width of the convolutional kernels at each layer (e.g., $(3, 5)$).
  - **Filter Channels**: A list of integers specifying the number of output channels (filters) at each layer.
  - **Use Pooling**: A boolean indicating whether a pooling layer is included in the branch.

Example of Solution Representation

As an illustrative example, consider a coral representing an Inception module with three branches. The representation might be as follows:

- **Number of Branches**: 3
- **Branches Parameters**:
  1. **Branch 1**:
     - **Depth**: 2
     - **Filter Sizes**: $[(3, 3), (5, 5)]$
     - **Filter Channels**: $[32, 64]$
     - **Use Pooling**: `True`
  2. **Branch 2**:
     - **Depth**: 3
     - **Filter Sizes**: $[(1, 7), (7, 1), (3, 3)]$
     - **Filter Channels**: $[16, 16, 32]$
     - **Use Pooling**: `False`
  3. **Branch 3**:
     - **Depth**: 1
     - **Filter Sizes**: $[(1, 1)]$
     - **Filter Channels**: $[64]$
     - **Use Pooling**: `False`

This representation captures all necessary information to construct the corresponding Inception module within the CNN architecture. By encoding the solutions in this structured manner, we enable efficient manipulation of the architectures during the optimization process.

2.3.3 Algorithm Implementation

CRO operates on a virtual reef represented as a two-dimensional grid of size $N \times M$. Each cell in the grid can be occupied by a coral (candidate solution) or remain empty. The initial population of corals is randomly generated and placed in the reef based on an initial occupation ratio ρ_0.

Key parameters of the algorithm include:

- ρ_0: Initial occupation ratio of the reef, representing the fraction of the reef initially occupied.
- F_b: Fraction of corals involved in external sexual reproduction (broadcast spawning).
- F_a: Fraction of corals involved in asexual reproduction (budding).
- P_a: Probability of a coral reproducing asexually during budding.
- F_d: Fraction of corals subject to depredation (predation), which removes less fit individuals.
- P_d: Probability of depredation occurring.
- κ: Maximum number of attempts a larva makes to settle in the reef.
- **Mutation Rate**: Probability of mutations occurring during brooding, introducing variability.
- **Maximum Generations**: Maximum maximum of generations the algorithm evolves.
- **Maximum no improvement**: Optional number of generations without an update on the best coral to early stop the evolution.

2.3.4 Initialization

The reef is initialized by randomly generating a population of corals (candidate solutions) based on the initial occupation ratio ρ_0. For each coral, the solution representation is constructed by:

1. Selecting a random number of branches, typically between 2 and 4.
2. For each branch:
    - Randomly determining the depth (number of layers), within a predefined range (e.g., 1 to 4 layers).
    - Assigning random filter sizes for each layer, chosen from a set of possible kernel sizes (e.g., heights and widths of 1, 3, 5, 7, or 9).
    - Assigning random numbers of filter channels for each layer, typically within a reasonable range (e.g., 4 to 64 filters).
    - Randomly deciding whether to include a pooling layer in the branch, with a uniform probability.

Each randomly generated solution is evaluated using the custom fitness function (described in Section 2.5) to assign an initial fitness value.

2.3.5 Reproduction Mechanisms

CRO models three main reproduction mechanisms to generate new candidate solutions:

Broadcast Spawning (External Sexual Reproduction)

A fraction F_b of the corals participates in **broadcast spawning**. Pairs of corals release gametes into the water column, where fertilization occurs randomly. This process is modeled by selecting pairs of corals to exchange genetic information through crossover operations, producing new larvae. The crossover involves combining architectural parameters from both parents to create offspring with mixed traits, enhancing exploration of the search space.

Crossover Operation:

The crossover between two parent solutions involves:

1. Determining the number of branches for the offspring, randomly chosen within the range of the parents' branch counts.
2. For each branch in the offspring:
    - Randomly selecting a corresponding branch from one of the parents.
    - Deep copying the selected branch's parameters (depth, filter sizes, filter channels, use of pooling) into the offspring.

Brooding (Internal Sexual Reproduction)

Corals not involved in broadcast spawning undergo **brooding**. Each coral produces a larva through mutation of its own solution representation. The mutation process introduces variability by altering architectural parameters, allowing the coral to explore new solutions close to its current state.

Mutation Operation:

The mutation involves:

- With a certain probability, given by the mutation rate, modifying various architectural parameters:

  - Changing the depth of branches.
  - Altering filter sizes at specific layers, selecting new kernel sizes from the predefined set.
  - Adjusting the number of filter channels at specific layers.
  - Flipping the boolean flag for the use of pooling layers.

- Additionally, with a random probability:
  - Adding a new branch to the Inception module, with randomly generated parameters.
  - Removing an existing branch, if the number of branches is above a minimum threshold.

The mutation process allows the exploration of new architectural configurations, which might lead to better performance.

Example of Genetic Operations

Consider two parent solutions with the following branch counts:

- **Parent 1**: 3 branches
- **Parent 2**: 4 branches

During crossover, the offspring might be assigned a branch count of 3 (randomly chosen between 3 and 4). For each of the 3 branches, the offspring randomly selects and copies the branch parameters from one of the parents.

During mutation, a coral might have one of its branches modified by changing the filter size at a particular layer from $(3, 3)$ to $(5, 5)$, or it might have a new branch added to its architecture.

Budding (Asexual Reproduction)

A fraction F_a of the fittest corals is considered for **budding**, a form of asexual reproduction where offspring are clones of the parent. However, budding occurs only if a random probability check satisfies the asexual reproduction probability P_a. This mechanism helps preserve high-quality solutions within the population while introducing stochasticity.

The budding process involves:

1. Selecting the top F_a fraction of corals based on fitness.

2. For each selected coral:

   - Generating a random number between 0 and 1.
   - If the random number is less than P_a, the coral reproduces via budding, producing an identical larva.

2.3.6 Larvae Settlement and Reef Update

The larvae produced during reproduction attempt to settle in the reef. Each larva has up to κ attempts to find a suitable location:

1. **Settlement Attempt**: A random position in the reef is selected.

2. **Settlement Criteria**:

   - If the position is empty, the larva settles there.
   - If the position is occupied by a coral with lower fitness, the larva replaces the existing coral.
   - If the position is occupied by a coral with equal or higher fitness, the larva fails to settle in this attempt.

3. **Failed Attempt Handling**:

   - The larva's settlement attempt counter (κ decreases by one, updating the remaining attempts parameter.
   - If the larva's counter is greater than zero after the failing attempt, it remains in the larvae pool for future settlement attempts in subsequent generations.
   - If the larva's counter reaches zero, it is discarded, simulating mortality due to environmental challenges.

The **larvae pool** is a collection of larvae that have not yet settled and have remaining settlement attempts. By retaining larvae the algorithm ensures that potentially valuable solutions are not prematurely discarded due to random chance.

2.3.7 Predation

To simulate natural predation and maintain diversity, a fraction F_d of the weakest corals is probabilistically removed from the reef. This process prevents premature convergence and allows exploration of new regions in the search space.

The predation mechanism involves:

1. Sorting the corals based on fitness, identifying the least fit individuals.
2. For each identified coral:
    - Generating a random number between 0 and 1.
    - If the random number is less than P_d, the coral is removed from the reef, creating space for new larvae to settle.

Adjusting the predation probability allows control over the selective pressure in the ecosystem. A higher predation probability can increase diversity but may also disrupt the retention of good solutions.

2.3.8 Fitness Evaluation

The fitness of each coral (solution) is evaluated using the custom fitness function described in Section 2.5, where the **accuracy** measured is the classification accuracy on the MNIST validation dataset. During the optimization process, we **partially train** each candidate model for a single epoch to estimate its potential without incurring excessive computational costs.

Partial training allows us to estimate the performance of each candidate model without the computational expense of fully training it. This approach is common in neural architecture search [20], where the goal is to evaluate many architectures efficiently.

2.3.9 Algorithm Termination

The CRO algorithm runs for a predefined maximum number of generations or until a convergence criterion is met, such as no improvement in the best solution over several generations.

Computational Resource Considerations: Due to limited computational resources, we configured the algorithm with moderate reef sizes, population counts, and a reasonable

number of generations. Despite these limitations, the implementation is designed to be scalable and can be extended to larger configurations and more complex systems when more computational power is available.

2.4 Visualization of Inception Modules

Visualizing the architectures of the Inception modules aids in understanding the configurations discovered by the CRO algorithm. It provides insights into how different architectural choices impact performance and helps interpret the optimization process.

2.4.1 Visualization Methodology

We utilize the `Graphviz` library [21] to create graphical representations of the Inception modules. Graphviz is an open-source graph visualization software that represents structural information as diagrams of abstract graphs and networks.

The visualization process involves:

1. **Graph Construction**: For each Inception module, we construct a directed acyclic graph (DAG) where nodes represent layers (e.g., convolutional layers, pooling layers), and edges represent the flow of data between layers.

2. **Branch Representation**: Each branch of the Inception module is represented as a sequence of nodes connected by edges, clearly delineated from other branches.

3. **Layer Details**: Each node is annotated with details such as layer type (e.g., Conv2D, MaxPooling), kernel size, number of filters (channels).

4. **Output Concatenation**: The outputs of all branches are connected to a single node representing the concatenation operation, illustrating how the module combines the features extracted by each branch.

The visualizations are saved as images and organized by generation, allowing for tracking the evolution of architectures over time. Additionally, the best solutions (coral architectures) are saved separately whenever there is an update of the best reef solution, enabling rapid comparison of architectural differences.

Figure 2.2 presents an example of a visualized Inception module architecture generated by the CRO algorithm.

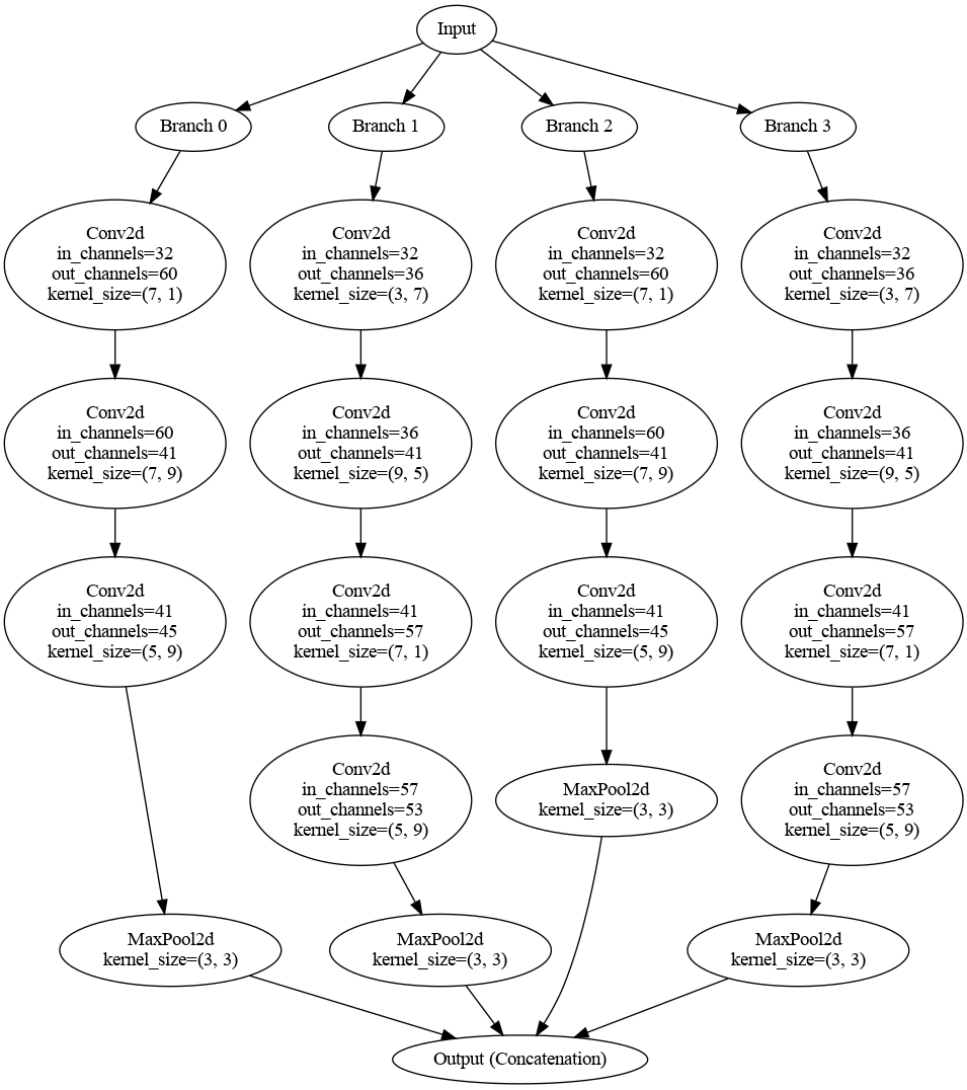


Figure 2.2: Visualization of an Inception module architecture after CRO algorithm completion.

The figure illustrates the diversity in branch configurations, including varying depths, filter sizes, and the inclusion of pooling layers.

2.5 Custom Evaluation Metric

2.5.1 Rationale

To align with Green AI principles [5], we design a novel custom evaluation metric that balances model accuracy with computational efficiency. The metric penalizes models with a higher number of parameters, encouraging the selection of architectures that achieve

good performance with lower complexity. This approach is essential because larger models consume more computational resources, leading to increased energy consumption and environmental impact.

While previous works have considered model size and computational cost in architecture search [22], our metric uniquely incorporates a logarithmic penalty term on the number of parameters, providing a balance between encouraging smaller models and maintaining high accuracy.

2.5.2 Metric Definition

The fitness function F used to evaluate each model is defined as:

$$F = \text{Accuracy} - \alpha \times \log(\text{Number of Parameters}) \tag{2.1}$$

Where:

- **Accuracy**: The classification accuracy of the model on the MNIST validation dataset.
- α: A scaling factor that controls the strength of the penalty for model complexity.
- **Number of Parameters**: The total number of trainable parameters in the model.

The complexity penalty term introduces a logarithmic penalty based on the model size. The logarithmic function ensures that the penalty increases with the number of parameters but does not dominate the fitness score for larger models unless they are significantly more complex.

The scaling factor α balances the trade-off between accuracy and complexity. A higher value of α places more emphasis on reducing model size, potentially at the expense of accuracy, while a lower value prioritizes accuracy over size reduction.

2.5.3 Justification

Traditional evaluation metrics often focus solely on accuracy, potentially leading to the selection of overly complex models. By incorporating a complexity penalty, our custom metric promotes the discovery of models that offer a better balance between performance and efficiency, adhering to the principles of sustainable AI development.

The novelty of our custom metric lies in its ability to guide the optimization process towards efficient architectures that are competitive in performance but significantly less resource-intensive. By integrating this metric into the CRO algorithm, we enable the discovery of models that are not only accurate but also align with Green AI objectives.

To the best of our knowledge, the specific formulation of the fitness function with a logarithmic penalty on the number of parameters in the context of CRO-based optimization of Inception modules is novel and has not been previously reported in the literature.

3 Experimental Results

This chapter presents the experimental results of applying the Coral Reef Optimization (CRO) algorithm to optimize dynamic Inception modules within Convolutional Neural Networks (CNNs) for the MNIST digit classification task. The primary focus is to evaluate the effectiveness of the proposed methodology in balancing classification accuracy and model complexity, adhering to Green AI principles outlined in Chapter 1.

3.1 Experimental Setup

3.1.1 Dataset Selection and Justification

The MNIST dataset [18] was selected for this study due to several compelling reasons:

- **Ease of Implementation**: MNIST is widely recognized for its simplicity and standardized format, allowing for straightforward preprocessing and rapid experimentation.

- **Benchmarking and Comparability**: The dataset serves as a common benchmark within the machine learning community. Extensive literature provides a wealth of baseline results, facilitating direct comparison of new models with state-of-the-art approaches.

- **Educational Value**: MNIST provides an excellent platform to validate novel methodologies before scaling to more complex datasets. It allows researchers to focus on architectural innovations and optimization techniques without the computational overhead associated with larger datasets.

The availability of public leaderboards, such as those compiled by *Papers with Code*, enables the assessment of model performance against top-performing architectures. This comparability is crucial for evaluating the effectiveness of our CRO-optimized models within the context of existing research.

3.1.2 Coral Reef Optimization Parameters

The CRO algorithm parameters are configured based on previous studies [13], [23] to ensure effective exploration and exploitation of the architectural search space. The parameters are as follows:

- **Reef Size**: 20×10 grid, resulting in a population of 200 corals (candidate solutions).
- **Initial Occupation Ratio** (ρ_0): 0.8, meaning 80% of the reef is initially occupied.
- **Broadcast Fraction** (F_b): 0.98, proportion of corals involved in broadcast spawning.
- **Asexual Fraction** (F_a): 0.05, proportion of corals reproducing asexually.
- **Asexual Probability** (P_a): 0.001, probability of a coral reproducing by budding.
- **Predation Fraction** (F_d): 0.05, fraction of corals subject to predation.
- **Predation Probability** (P_d): 0.01, probability of a coral being removed due to predation.
- **Maximum Generations**: 100.
- **Maximum No Improvement** (max_no_improve): 40, the number of generations without improvement before early stopping.
- **Larvae Settlement Attempts** (κ): 3.
- **Mutation Probability**: 0.2.
- **Fitness Function**: The custom evaluation metric defined in Section 2.5, balancing accuracy and model complexity.

3.1.3 Training Protocol

We followed a standard training methodology based on [24], [25]. The following training parameters are used:

- **Optimizer**: Adam optimizer [26], known for its adaptive learning rate and efficient handling of sparse gradients.
- **Learning Rate**: 0.001.
- **Batch Size**: 128, which balances computational efficiency and gradient estimation accuracy.
- **Model's Complexity Penalty (α)**: 0.1
- **Number of Epochs**:
  - **Partial Training**: Each candidate model was partially trained for **1 epoch** during the CRO optimization process. This partial training provided a quick estimation of each model's potential while minimizing computational cost and carbon footprint. Training each candidate for only one epoch significantly reduced the resources required, as evaluating every individual extensively would imply a large energy consumption.
  - **Full Training**: The best model identified by the CRO algorithm was intended to be fully trained for up to **300 epochs**. However, an **early stopping** mechanism with a patience of 40 epochs was implemented to prevent overfitting and unnecessary computations. Training would stop if there was no improvement in validation accuracy for 40 consecutive epochs.
- **Loss Function**: Categorical cross-entropy, suitable for multi-class classification tasks.

The use of early stopping in both the CRO optimization and the final model training aligns with the goal of reducing computational demands and environmental impact. It ensures that resources are allocated efficiently by halting processes that are no longer yielding improvements.

3.1.4 Computational Environment

The experiments were conducted on a system with the following specifications:

- **CPU**: Intel Core i9-13900H
- **GPU**: NVIDIA RTX 4070, GDDR6 8GB
- **RAM**: 16×2 GB DDR5
- **Operating System**: Ubuntu 22.04.5 LTS
- **Software Frameworks**:
  - Python 3.10.12
  - PyTorch 2.5.0

3.2 Results and Discussion

The CRO algorithm was executed with early stopping based on the max_no_improve parameter set to 40. The optimization process stopped at generation 73, indicating that there was no improvement in the best coral's fitness from generation 33 onwards. This early termination saved computational resources by avoiding unnecessary iterations.

Figure 3.1 illustrates the convergence of the average and best fitness values over generations. The initial generations showed rapid improvements due to high diversity in architectural configurations. After generation 33, the best fitness value plateaued, and no further improvements were observed up to generation 73.

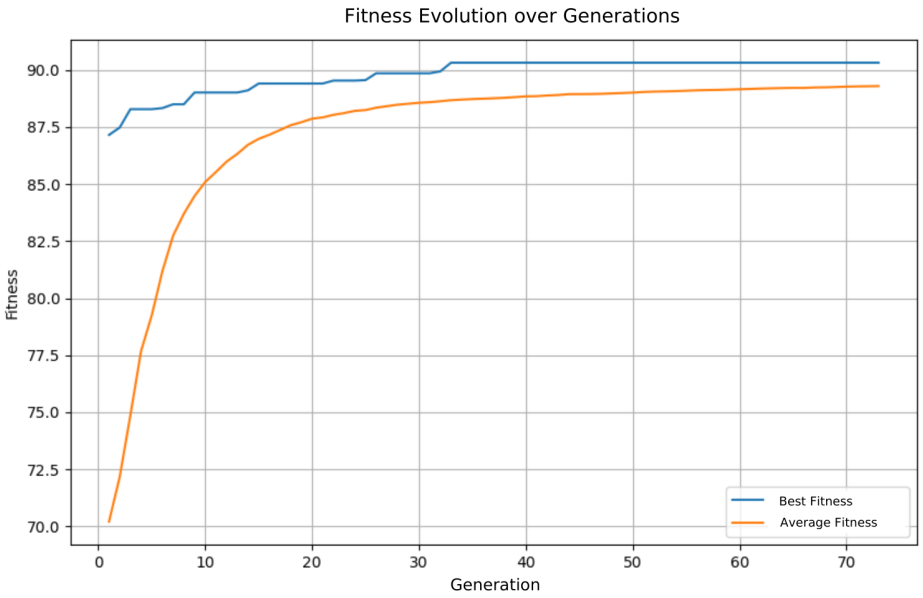
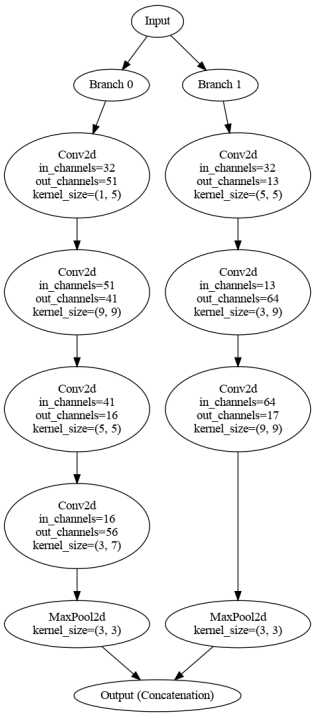


Figure 3.1: Convergence of average and best fitness values over generations with early stopping at generation 73.

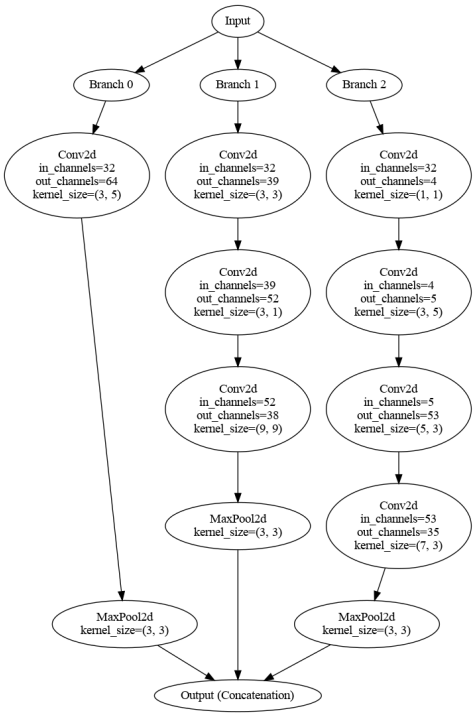
The consistent improvement in fitness demonstrates the CRO algorithm's effectiveness in navigating the architectural space and optimizing the Inception module configurations.

3.2.1 Evolution of Architectural Configurations

The evolution of the Inception module architectures over generations is depicted in Figures 3.2 to 3.8. Early generations featured a wide variety of configurations with differing numbers of branches, branch depths, filter sizes, and inclusion of pooling layers. This diversity was crucial for extensive exploration of the search space.

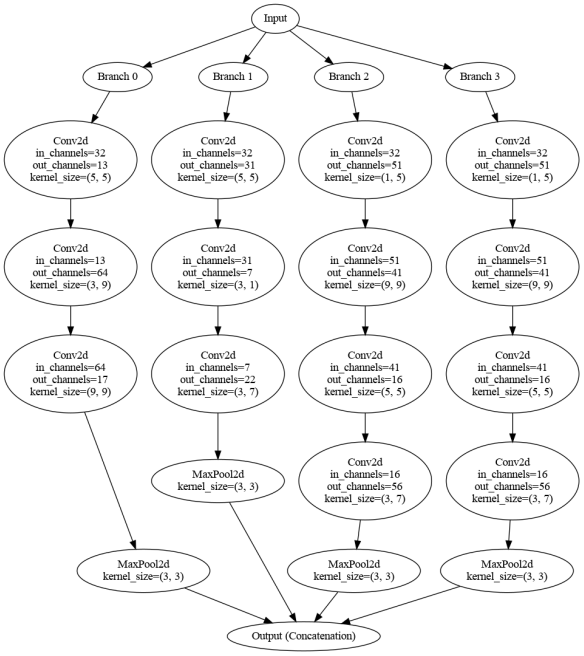


(a) Best Coral Architecture at Generation 1

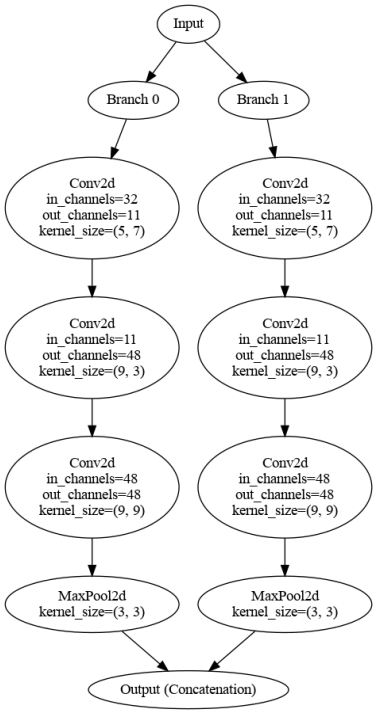


(b) Best Coral Architecture at Generation 2

Figure 3.2: Evolution of the Best Coral Architectures at Generations 1 and 2.

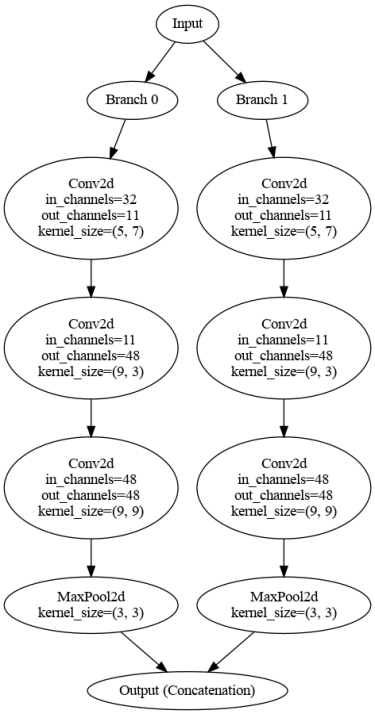


(a) Best Coral Architecture at Generation 3

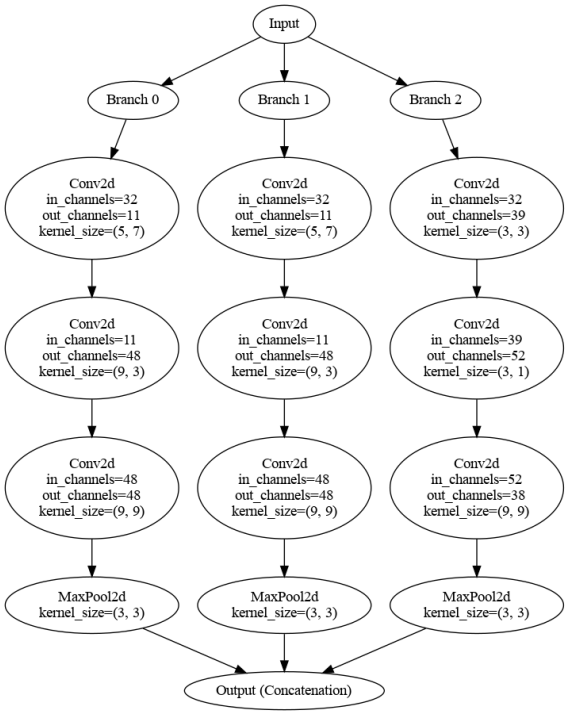


(b) Best Coral Architecture at Generation 6

Figure 3.3: Evolution of the Best Coral Architectures at Generations 3 and 6.

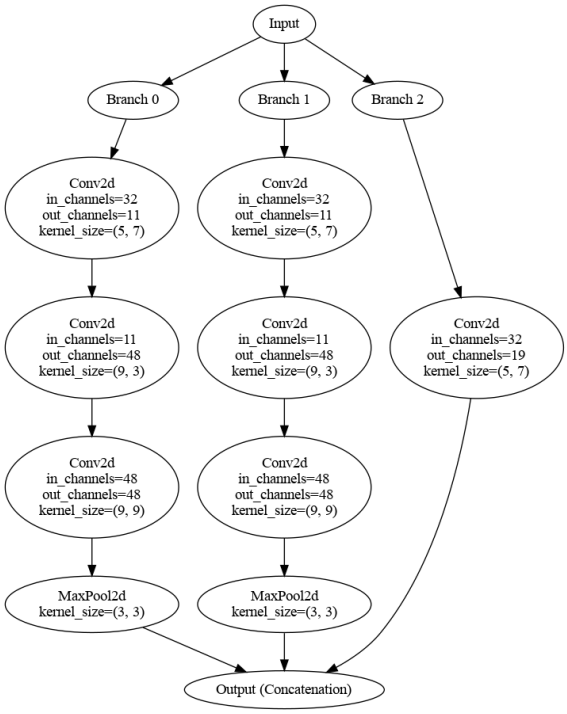


(a) Best Coral Architecture at Generation 7

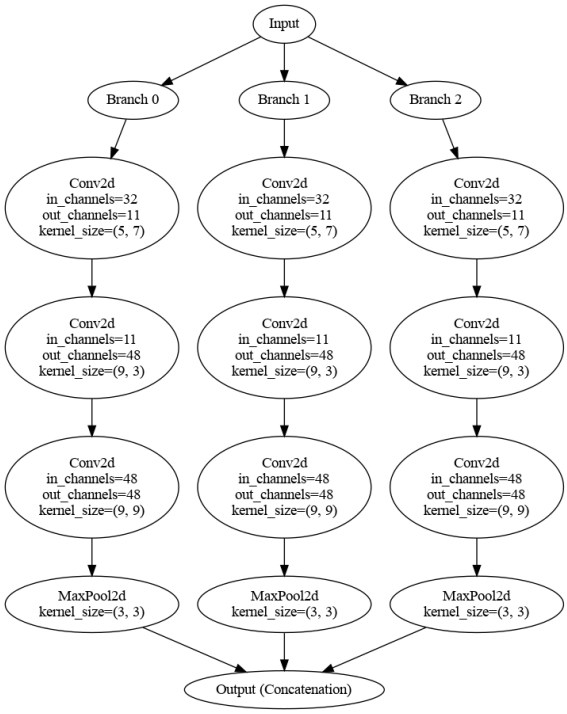


(b) Best Coral Architecture at Generation 9

Figure 3.4: Evolution of the Best Coral Architectures at Generations 7 and 9.

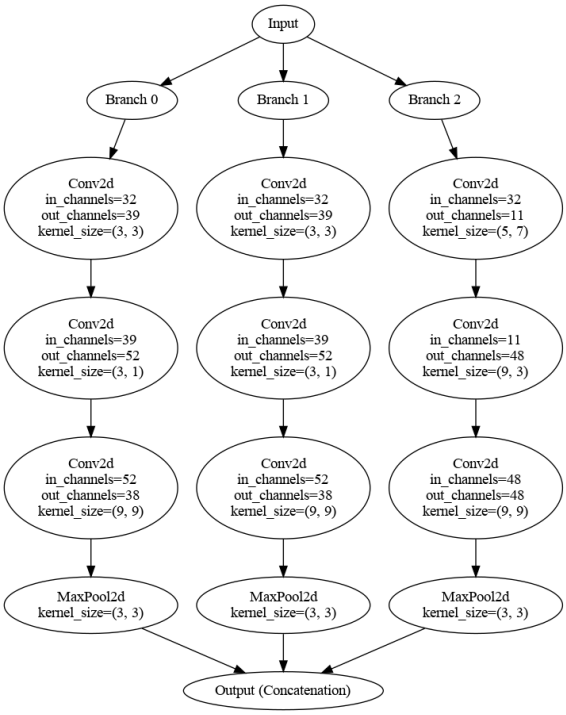


(a) Best Coral Architecture at Generation 14

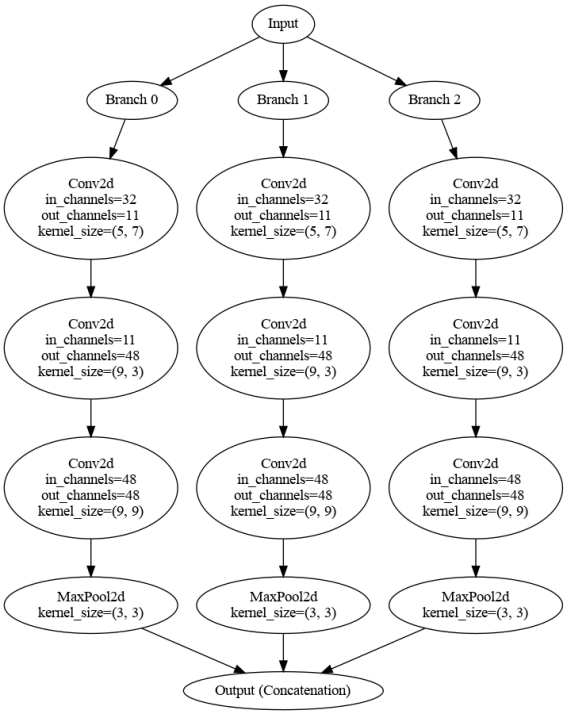


(b) Best Coral Architecture at Generation 15

Figure 3.5: Evolution of the Best Coral Architectures at Generations 14 and 15.

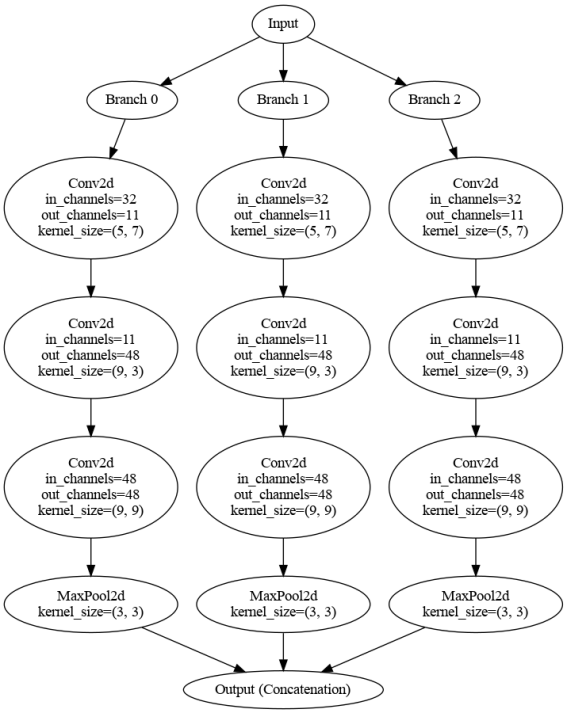


(a) Best Coral Architecture at Generation 22

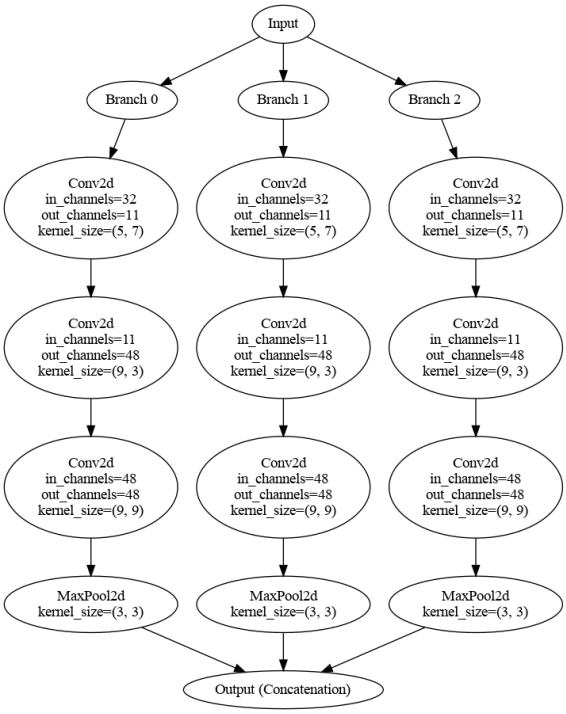


(b) Best Coral Architecture at Generation 25

Figure 3.6: Evolution of the Best Coral Architectures at Generations 22 and 25.



(a) Best Coral Architecture at Generation 26



(b) Best Coral Architecture at Generation 32

Figure 3.7: Evolution of the Best Coral Architectures at Generations 26 and 32.

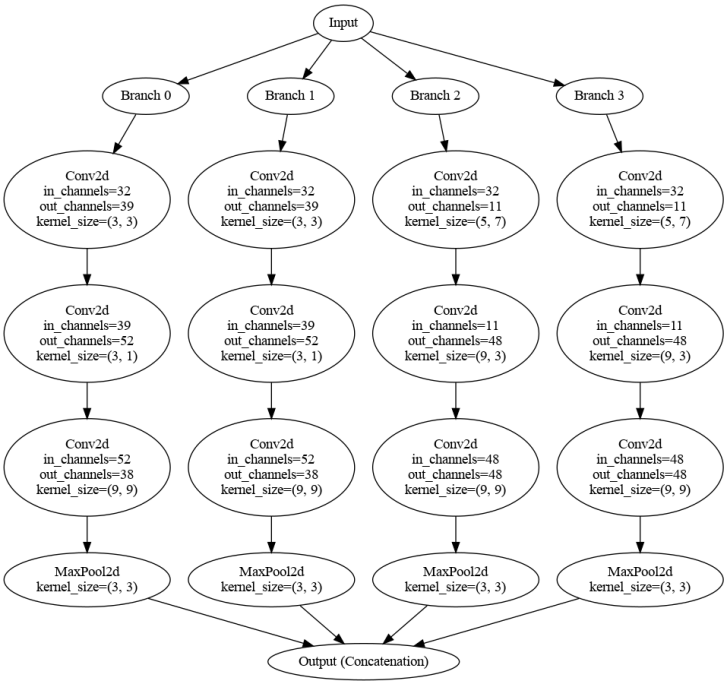


Figure 3.8: Best Coral Architecture at Generation 33 (Final)

As the algorithm progressed, certain architectural patterns emerged:

- **Branch Depth**: Branches with depths of 3 or 4 layers were more prevalent.
- **Filter Sizes**: Non-square filters such as (5×7) and (9×3) became common, enhancing the model's ability to capture directional features.
- **Pooling Layers**: The inclusion of pooling layers was favored in all branches, contributing to dimensionality reduction and overfitting prevention.

3.2.2 Best Coral Architecture

The best-performing architecture, referred to as the **CRO-Optimized Model**, is detailed in Table 3.1.

Table 3.1: Architecture of the Best Coral Discovered by CRO

Branch	Depth	Filter Sizes	Filter Channels	Pooling
Branch 0	4	[(3,3), (3,1), (9,9), (3,3)]	[32, 39, 52]	Yes
Branch 1	4	[(3,3), (3,1), (9,9), (3,3)]	[32, 39, 52]	Yes
Branch 2	4	[(5,7), (9,3), (9,9), (3,3)]	[32, 11, 48]	Yes
Branch 3	4	[(5,7), (9,3), (9,9), (3,3)]	[32, 11, 48]	Yes

The CRO-Optimized model underwent full training with an initial setting of up to 300 epochs. An early stopping mechanism with a patience of 40 epochs was applied to prevent overfitting. The training process halted at epoch 52, with the best validation accuracy achieved at epoch 12. Figure 3.9 shows the training and validation accuracy over epochs.

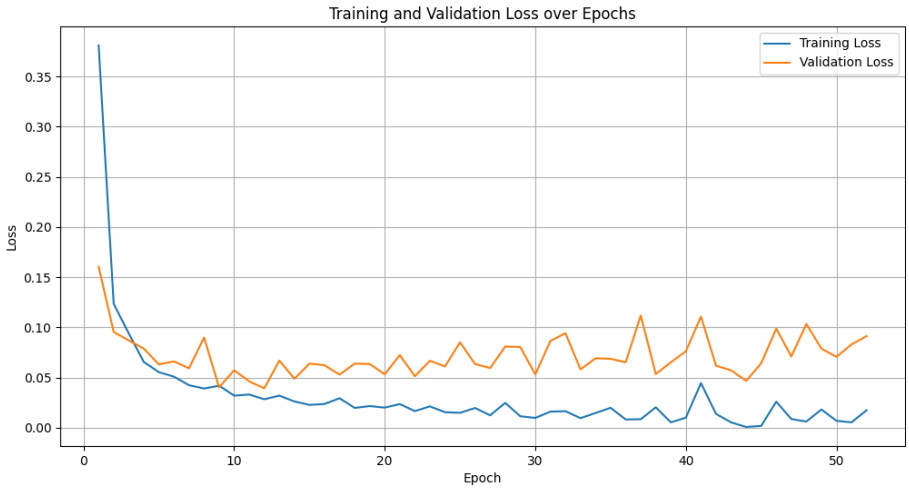


Figure 3.9: Training and validation accuracy over epochs for the CRO-Optimized Model. The best validation accuracy was achieved at epoch 12.

3.2.3 Performance Comparison

To assess the model's effectiveness, it was compared against top-performing models on the MNIST dataset. The results are summarized in Table 3.2.

Table 3.2: Performance Comparison with Top-Performing Models on MNIST

Model	**Parameters**	**Accuracy (%)**
BranchingMerging CNN + Homogeneous Vector Capsules [24]	*1514187*	**99.87**
Inception-CRO, best module (ours)	**805370**	98.92
SOPCNN (only a single model) [27]	1882602	*99.5*

Our CRO-optimized single Inception module model achieved an accuracy comparable to these top-performing models but with a significantly reduced number of parameters, (up to 50% less compared to some models). This efficiency underscores the model's suitability for deployment in environments where computational resources are limited.

4 Conclusions

This master's thesis presented a novel methodology that combines the *Coral Reef Optimization* (CRO) algorithm with dynamic *Inception modules* to efficiently optimize the architectures of *Convolutional Neural Networks* (CNNs). The primary goal was to balance classification accuracy with model complexity, aligning with *Green AI* principles that emphasize sustainability and efficiency in AI development.

The research demonstrated several key findings:

- **Effective Navigation of Architectural Search Space**: The CRO algorithm effectively explored and exploited the vast architectural search space, identifying models that balance high accuracy with low complexity. Specifically, the CRO-Optimized Model achieved a test accuracy of **98.92%** on the MNIST dataset with approximately **800 thousand** parameters, significantly fewer than comparable models.

- **Incorporation of Non-Square Kernels**: Introducing non-square convolutional kernels enhanced the model's ability to capture diverse and directional features, contributing to performance gains. This innovation allowed the network to recognize complex patterns within the data more effectively.

- **Custom Evaluation Metric for Efficiency**: The development of a custom evaluation metric that penalizes model complexity successfully promoted smaller architectures without significant loss in accuracy. This metric incentivized the CRO algorithm to favor models that are not only accurate but also resource-efficient.

- **Alignment with Green AI Principles**: The methodology reduced computational demands and potential energy consumption by implementing early stopping mechanisms and efficient training protocols, along with the bio-inspired evolution-

ary algorithm. This approach demonstrates that high-performance models can be developed sustainably.

4.1 Broader Implications

The findings of this study have broader implications for the field of deep learning and sustainable AI:

- **Framework for Sustainable Neural Network Optimization**: The integration of evolutionary algorithms like CRO with dynamic architectural components provides a viable framework for optimizing neural networks in a resource-conscious manner. This approach can be adapted to various architectures and tasks, promoting efficiency across the AI industry.
- **Demonstration of High Performance with Low Complexity**: The success of the CRO-Optimized model illustrates that it is possible to achieve state-of-the-art performance without resorting to excessively large and computationally intensive models. This finding challenges the notion that larger models are inherently better and encourages the development of compact, efficient architectures.
- **Contributions to Green AI Initiatives**: By emphasizing energy efficiency and reduced computational costs, this research contributes to Green AI initiatives aimed at minimizing the environmental impact of AI technologies. It highlights the importance of considering sustainability as a core component of AI development.

4.2 Limitations

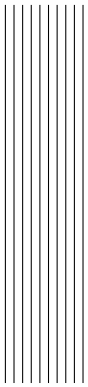
While the study achieved its objectives, several limitations should be acknowledged:

- **Dataset Complexity**: The experiments were conducted on the MNIST dataset, which is relatively simple compared to more complex datasets like CIFAR-10 or ImageNet. The scalability of the methodology to more challenging tasks remains to be explored.
- **Hyperparameter Sensitivity**: The performance of the CRO algorithm is sensitive to its hyperparameters. Finding the optimal settings may require additional experimentation and domain expertise.

4.3 Future Work

Building on the findings and acknowledging the limitations, potential extensions of this research include:

- **Application to Complex Datasets**: Extending the methodology to more complex datasets such as CIFAR-10, CIFAR-100, or ImageNet to assess scalability and effectiveness in more challenging contexts. This would test the robustness of the approach and its applicability to real-world problems.
- **Hyperparameter Tuning and Cross-Validation**: Implementing automated hyperparameter tuning methods, such as grid search or Bayesian optimization, to optimize parameters like the early stopping criteria for both the CRO and final neural network training. Cross-validating the alpha coefficient in the custom evaluation metric, which controls the penalty applied to the architecture's complexity, can help in finding the optimal balance between accuracy and efficiency. This systematic approach can enhance the overall performance and reliability of the models.
- **Multi-Objective Optimization**: Incorporating additional objectives into the fitness function, such as inference speed, memory usage, or energy consumption, to perform a more comprehensive optimization that aligns with practical deployment considerations.
- **Green AI Metrics Integration**: Including direct measurements of environmental impact, such as carbon footprint and energy efficiency metrics, to quantify the sustainability benefits of the optimized models and further align with Green AI goals.
- **Robustness and Generalization Studies**: Evaluating the robustness of the optimized models against adversarial attacks, noise, or data distribution shifts to ensure reliability in diverse operating conditions.
- **Automated Hyperparameter Tuning**: Implementing automated methods for tuning CRO hyperparameters to reduce dependency on manual settings and enhance the algorithm's adaptability.



Bibliography

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[4] C. Szegedy, W. Liu, Y. Jia, *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[5] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green ai," *Communications of the ACM*, vol. 63, no. 12, pp. 54–63, 2020.

[6] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for modern deep learning research," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, 2020, pp. 13 693–13 696.

[7] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[8] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, 2017.

[9] S. Ioffe, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[10] X.-S. Yang, *Nature-inspired metaheuristic algorithms*. Luniver press, 2010.

[11] T. Back, *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[12] S. Salcedo-Sanz, J. Del Ser, I. Landa-Torres, S. Gil-López, and J. Portilla-Figueras, "The coral reefs optimization algorithm: A novel metaheuristic for efficiently solving optimization problems," *The Scientific World Journal*, vol. 2014, no. 1, p. 739 768, 2014.

[13] A. M. Durán-Rosal, P. A. Gutierrez, S. Salcedo-Sanz, and C. Hervas-Martinez, "A statistically-driven coral reef optimization algorithm for optimal size reduction of time series," *Applied Soft Computing*, vol. 63, pp. 139–153, 2018.

[14] R. A. Vasco-Carofilis, M. A. Gutiérrez-Naranjo, and M. Cárdenas-Montes, "PBIL for optimizing hyperparameters of convolutional neural networks and STL decomposition," in *Hybrid Artificial Intelligent Systems - 15th International Conference, HAIS 2020, Gijón, Spain, November 11-13, 2020, Proceedings*, E. A. de la Cal, J. R. V. Flecha, H. Quintián, and E. Corchado, Eds., ser. Lecture Notes in Computer Science, vol. 12344, Springer, 2020, pp. 147–159. DOI: 10.1007/978-3-030-61705-9_13. [Online]. Available: https://doi.org/10.1007/978-3-030-61705-9%5C_13.

[15] S. Baluja, *Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning*. Carnegie Mellon University, 1994.

[16] P. García-Victoria, M. A. Gutiérrez-Naranjo, M. Cárdenas-Montes, and R. A. V. Carofilis, "PBIL for optimizing inception module in convolutional neural networks," *Logic Journal of the IGPL*, vol. 31, no. 2, pp. 325–337, 2023. DOI: 10.1093/jigpal/jzac022. [Online]. Available: https://doi.org/10.1093/jigpal/jzac022.

[17] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, no. 1-3, pp. 489–501, 2006.

[18] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[19] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM computing surveys (CSUR)*, vol. 35, no. 3, pp. 268–308, 2003.

[20] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *Journal of Machine Learning Research*, vol. 20, no. 55, pp. 1–21, 2019.

[21] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull, "Graphviz—open source graph drawing tools," in *Graph Drawing: 9th International Symposium, GD 2001 Vienna, Austria, September 23–26, 2001 Revised Papers 9*, Springer, 2002, pp. 483–484.

[22] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*, PMLR, 2019, pp. 6105–6114.

[23] S. Salcedo-Sanz, J. E. Sanchez-Garcia, J. A. Portilla-Figueras, S. Jimenez-Fernandez, and A. M. Ahmadzadeh, "A coral-reef optimization algorithm for the optimal service distribution problem in mobile radio access networks," *Transactions on Emerging Telecommunications Technologies*, vol. 25, no. 11, pp. 1057–1069, 2014.

[24] A. Byerly, T. Kalganova, and I. Dear, "No routing needed between capsules," *Neurocomputing*, vol. 463, pp. 545–553, 2021.

[25] A. Byerly and T. Kalganova, "Homogeneous vector capsules enable adaptive gradient descent in convolutional neural networks," *IEEE Access*, vol. 9, pp. 48 519–48 530, 2021.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[27] Y. Assiri, "Stochastic optimization of plain convolutional neural networks with simple methods," *arXiv preprint arXiv:2001.08856*, 2020.