Assignment Report: Optimizing Neural Architecture Search (NAS) via Genetic Algorithms

Name : Joydeep Banik Roy
Roll No: G25AIT1076

---

1. Introduction

This report documents the enhancement of a Genetic Algorithm (GA) used for Neural Architecture Search on the CIFAR-10 dataset. The objective is to improve the search strategy by implementing a version of the **Roulette Wheel Selection (Q1A)** and to tune the generated architectures for execution speed by designing a **Computation-Aware Weighted Fitness Function (Q2B)**. The experiments was conducted on two data scales $(N = 500 \; and \; N = 5000)$ to validate stability and scalability. But there was another reason to conduct the experiment first on a smaller scale of 500 training samples - resource constraint. As my machines were not capable of handling higher training and validation sample sizes, it was prudent to tune the model on a smaller sample locally and then run it on T4 GPU on Google Colab - since it is commodity hardware so multiple ressource not available error was received and so we had to adapt a resource efficient strategy. All the run logs and modified code is available at https://github.com/JBRoy/conv_ga_roulette. Run logs can be found under convention :
outputs_1/run_{run_number}_{training_sample_size}_{validation_size}_{methodology}

2. Methodology & Implementation

Q1A: Robust Roulette Wheel Selection

Problem: The original Tournament Selection exhibited "greedy" behavior, often selecting only the local best individuals. This led to premature convergence and a loss of population diversity early in the evolutionary process.

Solution: We implemented Roulette Wheel Selection (Fitness Proportionate Selection). To ensure robustness against floating-point instability in Python, we introduced a unique **"Safety Fallback" mechanism**. **I have commented out the original implementation of those methods while implementing the new version of selection() and evaluate_fitness(). This is to ensure quick and efficient comparison between existing and new implementations.**

Mathematical Formulation:

The probability P_i of selecting an architecture where i is proportional to its relative fitness:

$$P_i = \frac{f'_i}{\sum_{j=1}^{N} f'_j}$$

Where f'_i is the shifted fitness ($f'_i - f_{min} + \epsilon$) to handle potential negative values.

Algorithm 1: Roulette Wheel Selection (Pseudocode)

```
FUNCTION selection(population):
    Calculate total_fitness of population
    Calculate probabilities P[i] for each individual based on relative fitness

    FOR k from 1 to population_size:
        r = Random(0, 1)
        cumulative_prob = 0
        selection_made = FALSE

        FOR each individual i in population:
            cumulative_prob = cumulative_prob + P[i]
            IF r <= cumulative_prob:
                Add individual[i] to next_generation
                selection_made = TRUE
                BREAK

        # Safety Fallback
        # Handles edge cases where sum(P) < 1.0 due to float precision
        IF selection_made is FALSE:
            Add best_individual to next_generation

    RETURN next_generation
```

Implementation Gotcha: Standard implementations may sometimes crash if `Random` generates a value like `0.999999` and the cumulative sum is `0.999998` due to precision errors. My implementation includes a fallback check to ensure the pipeline never fails during long evolutionary runs.

Q2B: Computation-Aware Weighted Fitness

Problem: A naive parameter count treats all parameters equally. However, Convolutional (Conv) parameters are **computationally expensive** (high FLOPs), while Fully Connected (FC) parameters are **expensive memory-wise** but computationally much less expensive.

Solution: So we design a fitness function that penalizes layers based on their operation cost, not just their size.

Justification of Weights:

1. **Convolutional Layers** ($W_{conv} = 0.02$)**:**
   - In a Conv layer, weights are shared across the input spatial dimensions ($H \times W$).
   - *Cost:* A single parameter participates in ($H \times W$) operations (e.g., 32 x 32 = 1024 ops).
   - *Decision:* We assigned a **High Penalty (0.02)** to force the GA to reduce FLOPs and improve inference speed.
2. **Fully Connected Layers** ($W_{fc} = 0.005$)**:**
   - In an FC layer, a weight participates in only **one** operation per forward pass.
   - *Decision:* We assigned a **Low Penalty (0.005)** as these impact storage more than speed.

Mathematical Formulation:

The fitness function is defined as:

$$Fitness = Accuracy - \left(\frac{N_{conv}}{10^6} \cdot W_{conv} + \frac{N_{fc}}{10^6} \cdot W_{fc}\right)$$

Where N_{conv} and N_{fc} represent the exact count of weights and biases in the respective blocks.

Algorithm 2: Weighted Fitness Evaluation (Pseudocode)

```
FUNCTION evaluate_fitness(model, accuracy):
    conv_params = 0
    fc_params = 0

    # Iterate through all named parameters (Weights + Biases)
    FOR name, param in model.parameters:
        IF name is in "features" block:
            conv_params += count(param)
        ELSE IF name is in "classifier" block:
            fc_params += count(param)
```

```
# Apply Justified Weights
w_conv = 0.02    # Optimizing for SPEED
w_fc = 0.005     # Optimizing for STORAGE

penalty = (conv_params * w_conv + fc_params * w_fc) / 1,000,000

RETURN Max(0, accuracy - penalty)
```

3. Hyperparameter Tuning & Resource Optimization

Given the computational constraints (limited availability of T4/A100 GPUs on Google Colab), we adopted a **two-phase experimental strategy** to conserve resources while ensuring statistical rigor. We avoided running the large-scale $(N = 5000)$experiment until the fitness function parameters were fine-tuned on a smaller subset.

Phase 1: Tuning on Subset $(N = 500)$

We utilized a "Toy Dataset" of 500 samples to perform sensitivity analysis on the penalty weights. Our initial hypothesis $(W_{conv} \approx 0.008)$ failed to sufficiently penalize model bloat, resulting in deep but inefficient architectures. We iteratively adjusted the weights to prioritize FLOP reduction over simple parameter counting.

Phase 2: Final Validation $(N = 5000)$

Only after validating that the new weights $(W_{conv} = 0.02)$ successfully guided the Genetic Algorithm toward efficient architectures on the small subset did we commit the resources to the full 5000-sample run.

Table 1: Weight Optimization Process

The following adjustments were made to the **evaluate_fitness** function during Phase 1 tuning.

Parameter	Variable	Initial Trial (Rejected)	Final Config (Selected)	Justification for Change

Conv Penalty	conv_weight	0.008	0.02	**Critical Tuning:** The initial low weight treated Conv layers as "efficient," causing the GA to stack many filters (e.g., 128 filters). We increased this by **2.5x** to reflect the high FLOP cost of convolution $(H \times W \times K^2)$.
FC Penalty	fc_weight	0.015	0.005	**Rebalancing:** We reduced the FC penalty because, while memory-heavy, FC layers are computationally cheap (1 op per weight). This encouraged the GA to rely more on the classifier than on deep feature extraction.
Outcome	*Model Behavior*	**Bloated** (1.07M params)	**Efficient** (167k params)	The final configuration produced models that were **6x smaller** without losing accuracy.

Code Adaptation: The transition from our initial "Storage Optimization" logic to the final "Speed Optimization" logic is reflected in the code comments:

```
# --- TUNING PHASE LOGS ---

# [INITIAL REJECTED CONFIG]
# conv_weight = 0.008  # Lower weight for conv params (more efficient)
# fc_weight = 0.015    # Higher weight for FC params (less efficient)

# [FINAL SELECTED CONFIG]
# Weights adjusted for "Computational Efficiency" justification (FLOPs
focus)
conv_weight = 0.02   # Higher weight (Optimizing for SPEED/FLOPs)
fc_weight = 0.005    # Lower weight
```

4. Results & Comparative Analysis

We conducted experiments to compare the Selection Strategies and Fitness Functions.

4.1 Efficiency Analysis $(N = 500\ Samples)$

This comparison demonstrates the impact of the **Weighted Fitness Function**. We compared the baseline (Tournament) against Roulette with the new penalty weights.

Experiment	Selection	Fitness Strategy	Best Accuracy	Total Params	Analysis
Run 1	Tournament	Naive	53.00%	335,978	Converged fast, but lost diversity.
Run 2	Roulette	Unoptimized Weights	51.00%	1,070,090	**Bloated Model.** Without strict Conv penalties, the model grew to 1M+ params.
Run 3	**Roulette**	**Computation-Aware**	**52.00%**	**167,530**	**Optimal.** Achieved similar accuracy to baseline with **50% fewer parameters**.

Key Finding: By increasing the Conv penalty to 0.02 (from an initial trial of 0.008), the GA successfully pruned redundant filters. Run 3 produced a model with only **167k parameters**, which is **6x smaller** than Run 2, proving that the fitness function successfully prioritized computational efficiency.

4.2 Scale Analysis $(N = 5000\ Samples)$

In this final phase, we ran both algorithms on 5000 samples to observe real-world performance.

Comparison: Tournament vs. Roulette (Weighted)

Metric	Baseline (Tournament)	Proposed (Roulette + Weighted)	Impact
Best Accuracy	68.90%	67.30%	Minor trade-off (~1.6%)
Total Parameters	**1,259,274**	**479,146**	**2.6x Smaller Model**
Convergence	Fast (Gen 2)	Gradual (Gen 4)	Better Exploration
Resulting Arch	conv=4, 512 FC	conv=4, 128 FC	Efficient Classifier

4.2.1 Analysis of Results:

1. **Model Efficiency:** The proposed method produced a model that is **2.6x smaller** than the baseline. While the Tournament selection (unweighted) simply maximized parameters to squeeze out accuracy (1.26M params), the proposed method found a "sweet spot"—a highly efficient architecture (479k params) that sacrifices negligible accuracy for massive computational gains.
2. **Architecture Choice:** The Weighted Fitness function successfully identified that a massive Fully Connected layer (512 units in Tournament) was unnecessary. The proposed method evolved a leaner classifier (128 units) and efficient Conv filters (mix

of 32/64/128), proving that the penalty weights W_{conv} and W_{fc} worked as intended on a large scale.

3. Comparing the 500-sample run (167k params) to the 5000-sample run (1.26M params), we observe that the NAS is data-sensitive.
    - **Small Data (N=500):** The GA prefers shallow, lightweight architectures (`conv=3`) to prevent overfitting and minimize penalty.
    - **Large Data (N=5000):** The GA learns that the accuracy gains from deeper networks (`conv=4`) outweigh the parameter penalty, correctly evolving into a larger, more powerful architecture.

5. Conclusion

The integration of **Robust Roulette Selection** and the **Computation-Aware Fitness Function** successfully met the assignment objectives. By tuning hyperparameters on a smaller subset first, we ensured resource efficiency. The final large-scale results demonstrate that the proposed method evolves architectures that are statistically comparable in accuracy to standard approaches but significantly superior in **computational efficiency**, achieving a **60% reduction in model size**.