Assignment 2 for Artificial Intelligence Question 1 Report

Name: Aishwarya Mahindru, RollNo.: g25ait1012

Github Link: https://github.com/AishwaryaMahindru/nas-ga-basic

---

1. **Q1A – Roulette-Wheel Selection**
   In the original assignment template, parent selection was expected via tournament selection (commented out in your file). This was replaced with **Roulette-Wheel Selection**, implemented directly inside the `selection()` method.

   - **Code Behaviour:**
     - Computes fitness values for all individuals:
       fitnesses = [c.fitness for c in self.population]
     - Normalizes them to obtain probabilities.
     - Builds a cumulative probability list.
     - Draws two random values to select two indices:
       selected.append(idx)
     - These indices are later converted to actual parents inside evolve():
       parent1 = self.population[idx1]
       parent2 = self.population[idx2]

   - **How this is valid:**
     - Higher-fitness architectures get proportionally higher chances.
     - Lower-fitness ones still retain some probability → preserves genetic diversity.
     - Matches biologically-inspired evolutionary selection better than tournament selection.

   - **Code changes:**
     - Removed tournament selection (commented out).
     - Added roulette-wheel logic inside selection().
     - Updated evolve() to correctly convert selected indices into
       parent1 = self.population[idx1]
       parent2 = self.population[idx2]
     - Ensured crossover and mutation receive full Architecture objects.

   - **NAS Run Log Github link:** https://github.com/AishwaryaMahindru/nas-ga-basic/blob/main/outputs/run_1/nas_run.log

2. **Q1B – Modified Fitness Function (Weighted Conv/FC Penalty)**
   I revised the fitness computation inside evaluate_fitness() by introducing a weighted parameter penalty based on architecture complexity.
   - **What the Code Computes**
     - After training and obtaining best_acc, it computes two parameter counts:

- conv_params = 0
- fc_params = 0

```
for name, param in model.named_parameters():
    if 'feature_extractor' in name:
        conv_params += param.numel()
    elif 'classifier' in name or 'head' in name:
        fc_params += param.numel()
```

- Then apply weights:
  Conv weight: 0.7
  FC weight: 0.3

- Penalty:
  *penalty = (w_conv * conv_params + w_fc * fc_params) * 1e-6*

- Final fitness:
  architecture.fitness = best_acc - penalty

- **Justification of Weights:**
  - Convolution parameters dominate computational cost (most FLOPs).
  - Conv weights are reused across spatial locations → very expensive.
  - FC layers contribute fewer multiply-accumulate ops → lower cost.
  - Penalizing conv parameters more avoids evolution of overly large conv blocks.
  - Lower FC penalty preserves classifier capacity.
  - (0.7, 0.3) resembles real CNN compute distribution.

3. Results:

Generation 1 Results

Arch	Conv Params	FC Params	Old Penalty	New Penalty	Diff	Fitness	Acc
1	1,216	10,51,402	1.05265	0.31627	-0.73638	0.23873	0.56
2	27,536	21,02,794	2.13043	0.65011	-1.48031	-0.04411	0.61
3	2,41,312	10,49,994	1.29188	0.48392	-0.80797	0.15208	0.64
4	1,792	41,97,130	4.19905	1.26039	-2.93866	-0.73039	0.53
5	19,888	1,31,786	0.15183	0.05346	-0.09838	**0.55254**	0.61
6	2,03,264	41,95,722	4.39931	1.401	-2.9983	-0.882	0.52
7	28,032	10,49,290	1.07748	0.33441	-0.74307	0.18459	0.52
8	15,248	21,02,794	2.11814	0.64151	-1.47663	-0.05251	0.59
9	2,432	41,99,946	4.20244	1.26169	-2.94076	-0.73069	0.53
10	2,432	20,99,978	2.10247	0.6317	-1.47078	-0.0527	0.58

Generation 2 Results

Arch	Conv Params	FC Params	Old Penalty	New Penalty	Diff	Fitness	Acc
1	19,888	1,31,786	0.15183	0.05346	-0.09838	**0.52454**	0.58
2	1,216	10,51,402	1.05265	0.31627	-0.73638	0.21373	0.53
3	2,432	41,99,946	4.20244	1.26169	-2.94076	-0.70469	0.56

4	1,792	41,97,130	4.19905	1.26039	-2.93866	-0.73339	0.53
5	448	10,51,402	1.05188	0.31573	-0.73615	0.22627	0.54
6	2,432	41,99,946	4.20244	1.26169	-2.94076	-0.72469	0.54
7	1,792	41,97,130	4.19905	1.26039	-2.93866	-0.74139	0.52
8	1,792	41,97,130	4.19905	1.26039	-2.93866	-0.74339	0.52
9	1,216	21,02,794	2.10404	0.63169	-1.47235	-0.09969	0.53
10	2,432	41,99,946	4.20244	1.26169	-2.94076	-0.72169	0.54

Generation 3 Results

Arch	Conv Params	FC Params	Old Penalty	New Penalty	Diff	Fitness	Acc
1	19,888	1,31,786	0.15183	0.05346	-0.09838	**0.55854**	0.61
2	448	10,51,402	1.05188	0.31573	-0.73615	0.20627	0.52
3	1,216	21,02,794	2.10404	0.63169	-1.47235	-0.10269	0.53
4	1,216	21,02,794	2.10404	0.63169	-1.47235	-0.11269	0.52
5	1,216	21,02,794	2.10404	0.63169	-1.47235	-0.08369	0.55
6	1,792	41,97,130	4.19905	1.26039	-2.93866	-0.70239	0.56
7	45,968	20,99,978	2.14636	0.66217	-1.48419	-0.04517	0.62
8	4,05,520	2,64,970	0.67116	0.36336	-0.30781	0.30765	0.67
9	1,792	41,97,130	4.19905	1.26039	-2.93866	-0.72739	0.53
10	9,728	83,91,434	8.40142	2.52424	-5.87718	-2.03624	0.49

Generation 4 Results

Arch	Conv Params	FC Params	Old Penalty	New Penalty	Diff	Fitness	Acc
1	19,888	1,31,786	0.15183	0.05346	-0.09838	**0.55854**	0.61
2	4,05,520	2,64,970	0.67116	0.36336	-0.30781	0.30465	0.67
3	2,432	20,99,978	2.10247	0.6317	-1.47078	-0.0487	0.58
4	1,216	21,02,794	2.10404	0.63169	-1.47235	-0.10269	0.53
5	1,792	41,97,130	4.19905	1.26039	-2.93866	-0.71039	0.55
6	1,792	41,97,130	4.19905	1.26039	-2.93866	-0.71839	0.54
7	1,792	41,97,130	4.19905	1.26039	-2.93866	-0.69539	0.57
8	1,792	41,97,130	4.19905	1.26039	-2.93866	-0.72239	0.54
9	69,920	10,54,218	1.12436	0.36521	-0.75915	0.25479	0.62
10	1,216	21,02,794	2.10404	0.63169	-1.47235	-0.09969	0.53

Generation 5 Results

Arch	Conv Params	FC Params	Old Penalty	New Penalty	Diff	Fitness	Acc
1	19,888	1,31,786	0.15183	0.05346	-0.09838	**0.56254**	0.62
2	4,05,520	2,64,970	0.67116	0.36336	-0.30781	0.31865	0.68
3	4,05,520	2,64,970	0.67116	0.36336	-0.30781	0.31465	0.68
4	4,05,520	2,64,970	0.67116	0.36336	-0.30781	0.29265	0.66
5	4,05,520	2,64,970	0.67116	0.36336	-0.30781	0.30865	0.67
6	1,216	21,02,794	2.10404	0.63169	-1.47235	-0.08469	0.55
7	4,05,520	2,64,970	0.67116	0.36336	-0.30781	0.32065	0.68

8	4,05,520	2,64,970	0.67116	0.36336	-0.30781	0.30765	0.67
9	1,216	21,02,794	2.10404	0.63169	-1.47235	-0.09569	0.54
10	1,216	21,02,794	2.10404	0.63169	-1.47235	-0.08169	0.55

Summary of Best Architecture per Generation

Generation	Conv Layers	Best Fitness	Accuracy
Gen-1	3	0.55254	0.606
Gen-2	3	0.52454	0.578
Gen-3	3	0.55854	0.612
Gen-4	3	0.55854	0.612
Gen-5	3	**0.56254**	**0.616**

Final Best Architecture

Feature	Value
Conv layers	**3**
Conv configs	32-5, 16-3, 32-5
Pooling	Max
Activation	ReLU
FC Units	64
Final Accuracy	**0.616**
Final Fitness	**0.5625**
Total Params	1,51,834

Roulette Wheel Selection Details:

Index	Fitness	Probability	Cumulative
0	0.56254	0.260034	0.260034
1	0.32065	0.148217	0.408251
2	0.31865	0.147293	0.555544
3	0.31465	0.145444	0.700988
4	0.30865	0.14267	0.843658
5	0.30765	0.142208	0.985866
6	0.29265	0.135274	1.12114
7	-0.08169	-0.037761	1.08338
8	-0.08469	-0.039147	1.044232
9	-0.09569	-0.044232	1

High-Level Evolution Summary Table

Generation	Dominant Conv Depth	Common Filters	Common Activation	Pool	Notes
0	1 conv	16, 32, 64	ReLU + Leaky	avg	Very simple networks
1	1 + some 3/4 conv	16, 32, 64	ReLU	avg/max	More variety
2	Mixed 1, 3, 4 conv	32, 64, 128	ReLU	max+avg	Complex filters emerge
3	**4 conv dominant**	64, 128	ReLU	avg	Convergence begins
4	**4 conv strongly dominant**	64, 128	ReLU	avg	Population converged

Interpretation:

- The GA started with mostly 1-conv shallow models in Generation 0.
- As evolution progressed, deeper 3-conv and 4-conv architectures appeared and gained higher fitness.
- By Generations 3–4, the population strongly converged to 4-conv CNNs, indicating they performed best.
- ReLU became the dominant activation since it consistently gave higher accuracy.
- Avg pooling emerged as the stable pooling choice in later generations.
- FC units converged to 256, showing the GA found this configuration most efficient and reliable.

4. Learnings & Reflections

- Understanding Genetic Algorithms in NAS
  - I learned how Genetic Algorithms can guide neural architecture search by evolving CNN structures generation after generation.
  - I saw firsthand how even a small change in selection, mutation, or fitness calculation affects the entire search trajectory.
- Debugging & Mistakes Identified
  - I initially assumed convolution layers would contain "conv" in their parameter names, which caused conv parameters to always be zero.
    - This was fixed by switching to module-type checks (nn.Conv2d, nn.Linear) instead of name-based checks.
  - Another mistake was selecting parent indices and directly passing them into crossover/mutation.
    - This caused attribute errors until I correctly mapped indices back to architecture objects.
  - Throughout the assignment, I made multiple Git commits fixing mistakes step-by-step, which helped me track changes clearly and understand the evolution of my implementation.
- Importance of Logging & Iterative Reruns
  - Adding logging for fitness probabilities, parameter counts, and penalty differences made debugging much easier.
  - Multiple reruns taught me that evolutionary algorithms are extremely sensitive — even

small bugs drastically change the best architecture found.

- Training Efficiency & Hardware Impact
  - I realized how costly CNN evaluation inside a GA is, since every architecture must be trained, even if briefly.
  - Using an **NVIDIA A100 GPU** made a huge difference:
    - Runtime that took hours on CPU completed in minutes.
    - Faster feedback loops helped me debug and iterate much more effectively.

5. System Used and Runtime

- GPU – A100 (via Google Colab)
- Runtime – 47 minutes
- Snapshot of GPU Resource Utilization: https://github.com/AishwaryaMahindru/nas-ga-basic/blob/main/outputs/GPU_Run_Resources.png