PyNA-tta-S: a framework for Neural Architecture Search powered by metaheuristic optimization

Andrea Mazzeo[1], Roberto Del Prete[1,2], Maria Daniela Graziano[1]

[1] Università degli Studi di Napoli Federico II, Dipartimento di Ingegneria Industriale, Piazzale Tecchio 80, 80125 Napoli, Italy
[2] ESA Φ-lab, ESA-ESRIN, Via Galileo Galilei 1, 00044 Frascati, Italy

What is NAS?

Neural Architecture Search (NAS) is a process within the field of AI that focuses on automating the design of neural network architectures. The objective is to obtain an architecture, or several candidate architectures, that are **optimized for the application of interest** and to work on the device of interest. NAS can be used to discover neural network architectures that are more efficient, more accurate, and better tailored to the task and data at hand than those that could be designed by human experts alone.

What is PyNA-tta-S?

DL-based models have great potential for use in spaceborne activities, however their use comes with a unique set of challenges. Embedded systems have **severe limitations** in terms of **electrical power**, **data volume** and **data rate**, and hardware challenges like **processing power** and the **deterioration of components**. Moreover, the specific mission requirements might further limit the usability of AI models onboard of a spacecraft.

PyNA-tta-S is a NAS framework developed using pytorch lightning that was designed to output problem-specific solutions based on input data and user requirements. **Genetic Optimization** is used to explore the search space towards solutions that fit the applications of interest, guided by the selection of a **problem-specific fitness** function, and exploiting the **modularity** of CNN architectures. The framework was tested on a binary classification task of Sentinel-2 imagery of ship wakes and sea clutter, and the results were compared against those of an EfficientNetB0. PyNA-tta-S is currently **under development**.

Genetic Optimization

Genetic Algorithm (GA) optimization is inspired by the biological process of natural genetic transfer between generations. Any implementation of the genetic optimization algorithm employs the following steps:

1. Starting population is initialized
2. Fitness of every individual is evaluated
3. Mating pool is generated, the worst individuals are excluded
4. Couples are generated
5. Crossover and mutation used to generate new population
6. Repeat from step 2.

Architectures are codified as Chromosomes. Example:

- Architecture Code = «**LRr3agn1EPaELco2k3s2p2agn1EPMEHCEE**»
- Chromosome (Genotype) = '**LRr3agn1**' ,'**Pa**','**Lco2k3s2p2agn1**','**PM**','**HC**'
- '**Lco2k3s2p2agn1**' → **ConvBnAct** with:

{
Out channels = 2*In_Channels,
Kernel size = 3,
Stride = 2,
Padding = 2,
Activation = GELU,
Times repeated = 1,
}

New generations are obtained by crossover and mutation. Example:

- Single-point crossover:

Crossover from this point

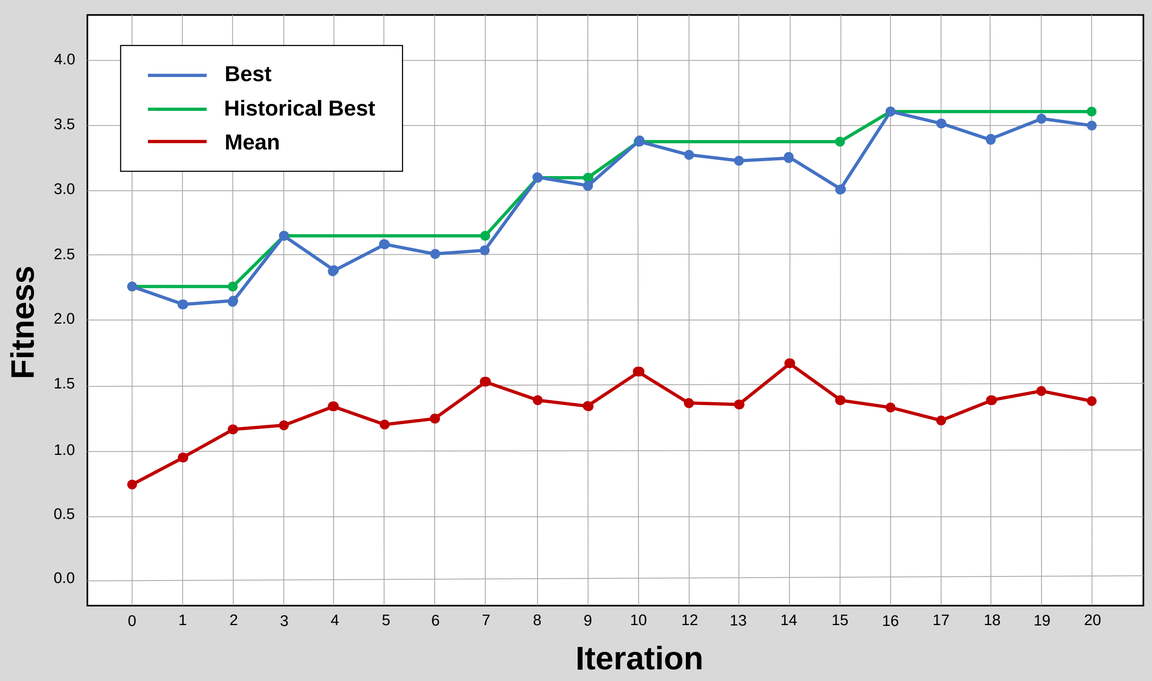
Child 1 = ['LRr3agn1' , 'Pa' , 'Lco2k3s2p2agn1' , 'PM' , 'HC']
Child 2 = ['Lbo3k5s1p1arn1' , 'PM', 'HC']

- Mutation:

Mutation

Child 1 = ['Leo3k5s2p0agn1' , 'PM' , 'HC']
Child 2 = ['Lbo3k5s1p1arn1' , 'PM', 'Lco2k3s2p2agn1' , 'PM' , 'HC']

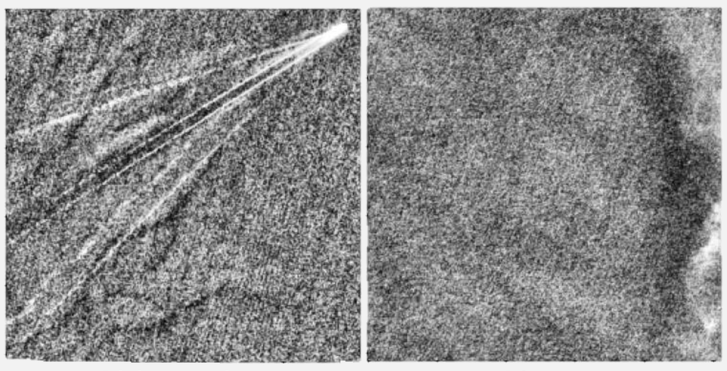
Genetic Algorithm optimization



$$fitness = \frac{w_{Acc} * Acc + w_{F1} * F1 + w_{MCC} * MCC}{w_{Acc} + w_{F1} + w_{MCC}} * \frac{1}{w_t * t_{train}}$$

- Results from testing on xS2Wakes - https://zenodo.org/records/10018939 - binary classification
- Reproducibility: seed = 7, mating pool cutoff = 0.8, mutation probability = 0.1, population size = 20
- For the fitness formula: w_{acc} = 3, w_{F1} = 2, w_{MCC} = 1, w_t = 10
- Notes: results will depend on testing hardware, on the dataset, on the application. The fitness formula should reflect the required performance

Data and Method



Ship Wake Clutter

The dataset:

- In ship wake detection, sea clutter often generates false positives. xS2Wakes is a small dataset with 2 labeled classes: **ship wakes** and **sea clutter**.
- 269 four-band S2 images 4x256x256 px
- Bands B2 (blue), B3 (green), B4 (red), B8 (near infrared)
- Dataset available at https://zenodo.org/records/10018939

The test was conducted on PyNA-tta-S and EfficientNetB0. For PyNA-tta-S, the following fitness function was used:

$$fitness = w_{Acc} * Acc - w_{np} * num\_param$$

- Where w_{acc} = 20, w_{np} = 1e-6
- seed = 7, population size = 20, GA_iterations = 10
- mating pool cutoff = 0.8, mutation probability = 0.1
- EfficientNetB0 initialized with IMAGENET1K_V1 weights

Results

	EfficientNetB0	PyNA-tta-S
Weight (MB)	46.4	7.97
Accuracy (%)	85.45	96.22

The test shows that PyNA-tta-S, when guided by a **fitness function that rewards lighter solutions**, successfully proposes an architecture that is significantly lighter than EfficientNetB0.

The performance of the architectures proposed by PyNA-tta-S can be improved through hyperparameter tuning and case-specific considerations.

Future Developments

PyNA-tta-S is still in an immature state, but the road towards improving it is relavely clear:

- The current generation of architectures is completely random, as are mutations and crossover. A logic based on experience could be implemented.
- The vocabulary will be expanded, and the parameter search space will be better evaluated.
- Detection and segmentation heads will be implemented to allow the framework to cover a plethora of other applications
- NAS operations will expanded to Neck architectures