Exercise 8: Anything goes

Due date: *December 12th, 2025 (push to GitHub repo before the lecture)*
General Requests: *Same as before - last chance to form proper gitflow habits T.T ...*

8.1 Anything goes

You've implemented and analyzed a ton of algorithms now, each with their own strengths and weaknesses. For this last exercise, 'anything goes'; we'll use everything that was discussed in the course, and all the knowledge you've acquired so far, to solve a Network Architecture Search (NAS) problem.

- pick an **algorithm** (HC, GAs, SA, ACO, PSO, DE) or an **algorithm combination** (i.e., combined global and local search strategies, etc.) of your liking, but be prepared to justify your choice
- use a **reasonable subsample** of the fashion mnist dataset. it consists of 70.000 grayscale images (60.000 for training, 10.000 for testing) of fashion articles (from Zalando, funnily enough) in 10 categories. the readme in the repo explains how to import the data with python, the labels available, some benchmarks, etc[1].
- you want to optimize a CNN architecture to optimize classification performance on the fashion mnist dataset. for a short **recap** on CNN design/use with pytorch, check out this link. some **guidelines** for your search space:
  - convolutional layers: at most 3
  - filters per layer: at most 32
  - kernel size per layer: 3x3 or 5x5
  - pooling layers after each convolutional layer: max pooling or average pooling
  - dropout after each convolutional layer: yes or no
  - dropout rate: at most 0.5
  - fully connected layer: at most 128 neurons

  discretize the parameters into however many steps you see fit (again, do whatever you want, but be prepared to justify your choices)[2]. if you want, you can of course expand the search space further (depthwise convolution, activation function, batch size, loss function, optimizer, etc.), but, again, note the compute time
- **important**: if these outlines blow up your compute, change them however you see fit[3]
- **important**: only train for a small number of epochs (e.g., 3-5), and use a fixed learning rate of, e.g., 1e-3
- design the **(guiding) objective function** however you see fit, but implement **at least one penalty**. use whichever metric(s) you find suitable. again, be prepared to defend your choices
- ideally, use your knowledge on performance testing (slides, chapter 2) to **validate** your results

8.2 Analysis and Reporting

Add a streamlit final page on NAS to the documentation you created for the last exercises.
Your final documentation webpage should contain the following sections:

1. Introduction - Overview of the problem: the main issues, the main goals and challenges, ...
2. Methods - Describe and defend your approach; which objective function did you implement and why? what does your search space look like (i.e., which values did you allow for the different parameters)? which metrics did you use, which penalties did you enforce? tell us all about your setup
3. Results - Display your found optimum architecture and its performance and visualize the search process (performance over time, etc.)
4. Discussion - Analyze your findings: do your results seem sound/could you confidently defend your results and choices to your boss? did you have issues with finding a suitable objective function or search space? did you easily decide on an algorithm/algorithm combination, or did you have to test multiple ones? what would you improve if you had more time/resources? what were your takeaways from this last exercise?

[1] The fashion MNIST dataset was designed to replace the classical MNIST handwritten digit dataset, for which results of over 95% accuracy can be achieved with out-of-the-box models and which thus doesn't pose as much of a challenge to model architectures.
[2] It might, e.g., make sense not to discretize the possible filter numbers in steps of 1, but rather as {4,8,16, ... }; it's your call, though.
[3] Again, it's about properly using all of the knowledge you've gained so far, not about achieving state-of-the-art results.