Initial Explorations in Optimizing Flownet

Kevin Cotellesso
Rose-Hulman Institute of Technology, ME497

5/25/2025

1 Introduction

Flownet is a neural network that predicts how the shape of an extruded fluid changes over some distance. This paper describes an exploration of using various optimization algorithms to generate a nozzle shape that will result in a desired extrusion shape. It will first overview the input to Flownet and how inlet nozzle shape was parameterized, then discuss how an objective function was constructed. Finally, it will present the implementation details and results of running Nelder-Mead, Particle Swarm Optimization, and a steepest descent line search algorithm on several test problems of varying difficulty.

2 Problem Formulation

To shoehorn Flownet into the role of an objective function that can be used with numerical methods, it needs to have numeric parameters as input and a scalar output to be minimized or maximized.

2.1 Input Parameterization

Flownet essentially has two inputs: a flow rate ratio that specifies the ratio between shaped and surrounding flows, and an image that describes the nozzle shape. For these explorations, I opted to fix flow rate ratio at 0.5 and encode nozzle shape as a list of connected points in polar coordinates.

Since a nozzle is roughly circular, the easiest method of encoding a nozzle shape is a list of polar coordinates (r, θ). If we let θ be evenly spaced, a nozzle can be encoded by a list of radii $\overline{r}$ or "spoke lengths". This method can encode any nozzle shape that passes the "radial line test" (the polar-coordinate manifestation of the vertical line test).

Examples of a nozzle image can be seen in the left pane of Figures 1 and 2, which show nozzle encodings

$$\overline{r}_{0,Easy1} = [550, 650, 650, 750, 450, 550, 750, 750, 650, 450]$$

and

$$\overline{r}_{0,Easy2} = [500, 500, 500, 500, 500, 500, 500, 500, 500, 500]$$

respectively.

2.2 Objective Function Value

The output of Flownet is a boolean image with low pixels for the outer flow and high pixels for the inner (shaped) flow. The objective is to make this output image look as similar as possible to some target image with the same structure, so a suitable objective function is the similarity between the two images. We can use $1 - MSE(Im_1, Im_2)$, where $MSE(Im_1, Im_2)$ is the mean squared error between Im_1 and Im_2.

The final optimization problem is the following unconstrained problem:

$$max : f(\overline{r}) = 1 - MSE(Im_{output}(\overline{r}), Im_{target})$$

where $\overline{r}$ is an ordered list of radii to points at evenly spaced angles comprising the inlet nozzle shape.

3 Methods

As this is an exploration, several optimization techniques were tried. These were chosen due to features of this specific problem, as detailed in the next few subsections.

3.1 Nelder-Mead

Nelder-Mead was attempted first because it inherently estimates the gradient and is a simple algorithm. The gradient estimate is important because the design space is continuous; arbitrarily small changes in $\overline{r}$ should result in small changes in the output image. However, it is difficult to calculate an analytic gradient for the space, as the objective function contains a convolutional neural network.

One potential issue is that Nelder-Mead can break down for high-dimensional problems, and a nozzle encoding $\overline{r}$ with n parameters is an n-D problem.

The implementation of Nelder-Mead used in this paper is Scipy's implementation.

3.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) was chosen because it is an exploratory method that works well in high dimensional continuous spaces such as the one in this problem. An gradient-free exploratory method such as PSO may prove superior to gradient based methods if the design space turns out to have discontinuities or many local minima.

The downside to exploratory methods in general is that they must rigorously enforce feasibility constraints, as it may be possible for them to find nozzle encodings that break Flownet. Gradient based methods, on the other hand, are likely to stay in feasible space if feasible initial nozzle encoding and target image are used.

3.3 Steepest Descent Line Search

Steepest Descent Line Search is a simple gradient-based method that works well for high-dimensional continuous problems. If this problem's design space is continuous, then steepest descent could find an optimum value more quickly than PSO.

The major downside to steepest descent is that it must perform an explicit gradient estimation using a finite difference method, which requires $O(n)$ function evaluations for an n-D problem. If step size is not tuned properly, steepest descent will be very slow and require many function evaluations.

I had to tweak a default implementation of steepest descent and difference estimation to work with this problem. Due to the range of $\bar{r}$ (pixel distances within $[100, 900]$) and the range of similarity values (scalar similarity values within $[0.0, 1.0]$), the gradient is very small throuought the space, so step sizes must be proportionally larger. Armijo backtracking line search with an initial step size of over 100,000 worked well.

Since the design space is based on pixel distances, the smallest meaningful change in a parameter is 1.0, so finite difference estimation had to use a step $h \geq 1$ which is much much larger than default. I found that $h = 5$ worked well for these tests.

4 Results

To evaluate the performance of these optimization algorithms, I formulated several problems for them to solve. Each problem consists of an initial nozzle encoding $\overline{r}_0$ and a target outlet shape. Four problems were formulated: Easy 1, Easy 2, Hard 1 and Hard 2. All are 10D problems; nozzle encodings $\overline{r}$ have 10 elements.

4.1 Specific Problems

Easy 1 was designed to be the easiest of the four for my gradient based methods. It was created by perturbing a semi-random initial nozzle encoding by 50 pixels for the first 5 elements and -50 pixels for the remaining 5 elements to get a solution nozzle. Easy 2 was designed as a more real-world example, with the same target as Easy 1 but with all values for $\overline{r}_0$ set to 500px — a generic circular nozzle.

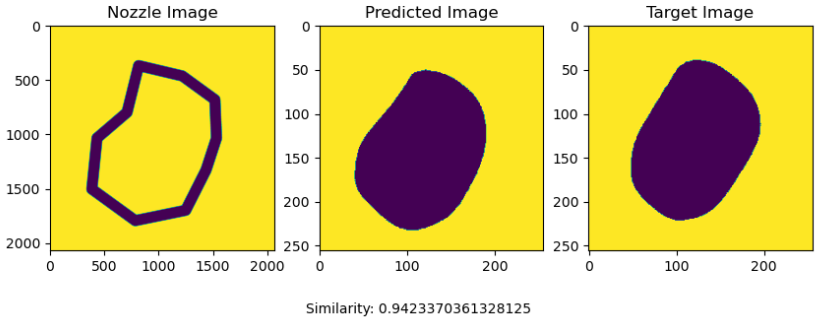

Figure 1: Easy 1 "fix the bean"

Hard 1 and Hard 2 are more difficult partly because of a more complex triangular shape and partly because they were generated using a 20-parameter nozzle encoding while the optimization algorithm only has access to a 10-parameter nozzle encoding. The initial nozzle encodings $\overline{r}_0$ for Hard 1 and Hard 2 are the same as for Easy 1 and Easy 2 respectively.

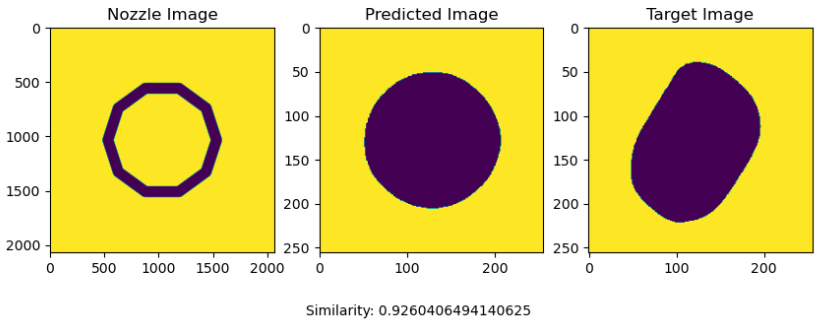

Figure 2: Easy 2 "fix the bean but you don't know it's a bean yet"

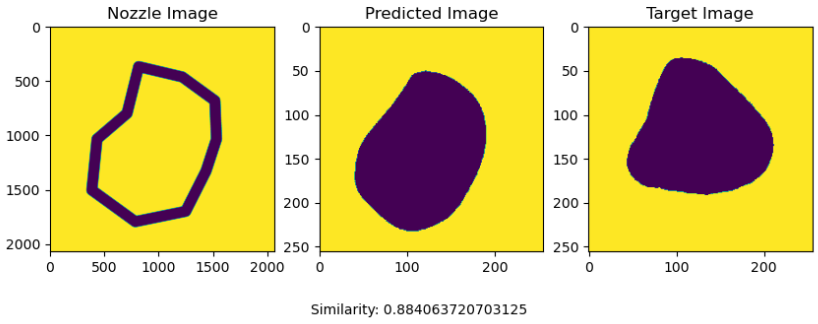

Figure 3: Hard 1 "The Strange Triangle"

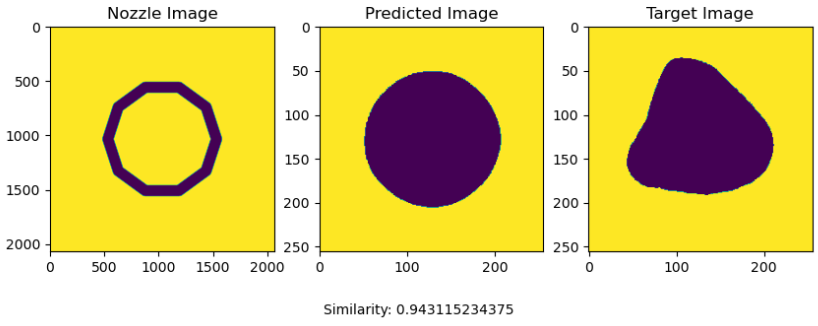

Figure 4: Hard 2 "The Shape Shifter"

4.2 Nelder-Mead

Nelder-Mead successfully solved Easy 1, Easy 2 and Hard 2. Table 1 shows a summary of the results. Figure 5 shows an example output of Nelder-Mead for task Hard 2.

Nelder Mead	Easy 1	Easy 2	Hard 1	Hard 2
Similarity	0.994	0.992	—	0.992
Iterations	167	228	—	183
Function Evals	429	508	—	452

Table 1: Nelder-Mead Performance on All Attempted Tasks

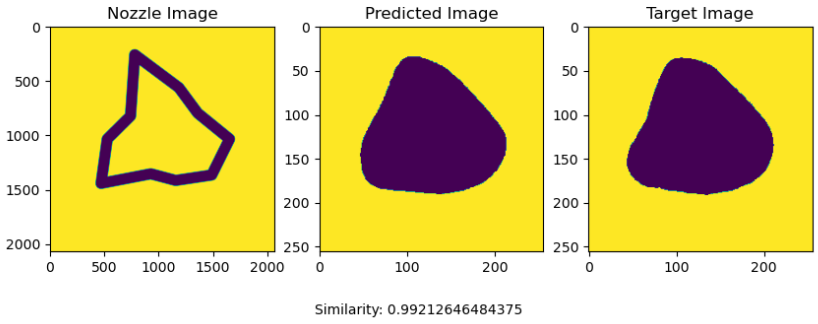

Figure 5: Nelder-Mead's Final Solution for Hard 2.

4.3 Steepest Descent

Steepest Descent successfully solved Easy 1 and Easy 2. Table 2 shows a summary of the results.

Steepest Descent	Easy 1	Easy 2	Hard 1	Hard 2
Similarity	0.9986	0.9945	—	—
Iterations	95	200	—	—
Function Evals	—	7074	—	—

Table 2: Steepest Descent Performance on All Attempted Tasks

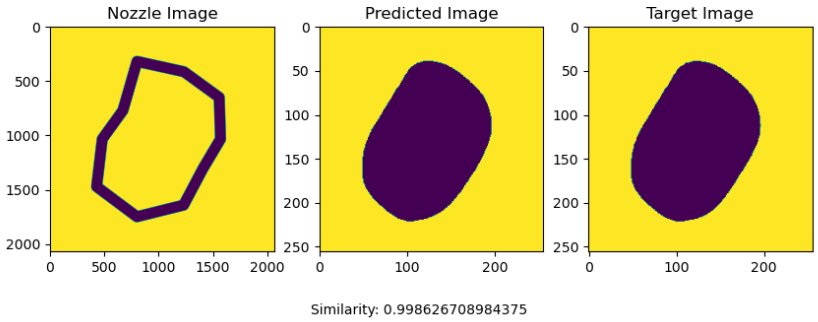

Figure 6: Steepest Descent, Easy 1. This was the best match found by any of the algorithms. Imagine what it could have done if it hadn't OOM'ed my laptop...

4.4 Particle Swarm Optimization

Particle Swarm Optimization does not require an initial chromosome, but was able to successfully solve the Easy target image. Since it is a stochastic algorithm, it was run 5 times to ensure its repeatability. The results from these runs are tabulated in Table 3. Visual results from PSO's best and worst runs are shown in figures 7 and 8 respectively.

PSO	Run 1	Run 2	Run 3	Run 4	Run 5
Similarity	0.981	0.995	0.994	0.985	0.991
Iterations	91	78	79	83	97
Function Evals	3614	3072	3101	3256	3996

Table 3: PSO Performance on Easy 1/2 Task

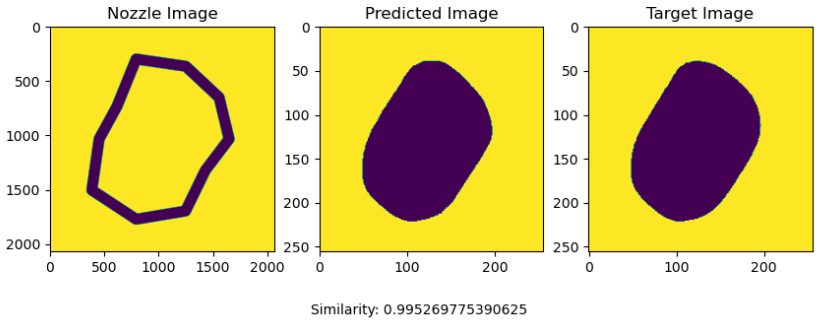

Figure 7: PSO's best run — Run 2

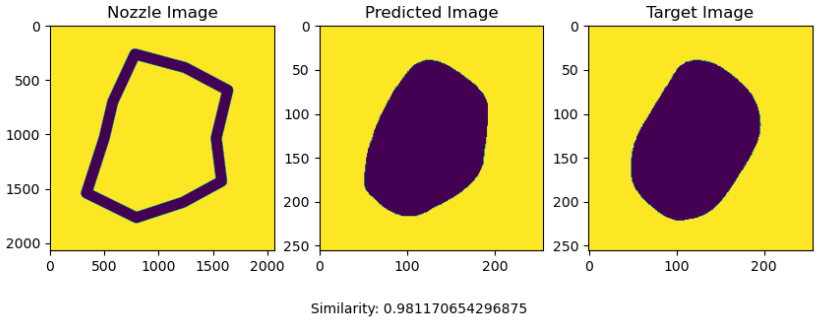

Figure 8: PSO's worst run — Run 1

5 Discussion

In these trials, all three optimization algorithms were able to solve the backward pass of Flownet. Nelder-Mead performed the best, finding satisfactory visual matches and similarity values with roughly 500 function evaluations for each attempted problem. This is possibly due to a more mature implementation courtesy of Scipy; the other two algorithms were implemented by yours truly.

PSO found highly competitive solutions for the Easy problems 3 out of the 5 runs, but for the remaining 2 found visually subpar solutions. This could be due to the termination conditions used; stagnation conditions could have been too lax, allowing small improvements to be misidentified as stagnation. It must be noted that PSO used a fairly large number of function evaluations (3000-4000).

Steepest descent was the most fincky to get working by far, and used a large number of function evaluations (about 7000 for the only run that finished). It

also achieved the highest similarity scores of 0.999 and 0.995 on Easy 1 and 2 respectively, outperforming Nelder-Mead and most PSO runs.