Norwegian University of Science and Technology

DT8122 **Project Assignment — Flow Matching**

# PROBABILISTIC AI

dt8122@idi.ntnu.no

Summer 2025

## 1 Introduction

The project assignment is designed to give you hands-on experience with Flow Matching. Results from training and evaluating models on MNIST should be compiled into a report. The report should also cover discussion of results and methods as specified in the specific tasks.

## 2 Theory Questions

First we want you to dive deeper into Flow Matching [4, 1] and related methods/approaches. In addition to the great summer school lecture notes, there are a lot of resources online: articles, blogposts, video lectures etc.

1. Give a short, high-level description of Flow Matching and its *optimization objective.*
2. Normalizing Flow typically train one network for each layer/transformation, explain briefly how Flow Matching and Contineous Normalizing Flows (CNF) [2, 3] use the same network for every continuous $t$ value.
3. Compare Flow Matching and Contineous Normalizing Flows? Similarities/differences, pros/cons etc. You may structure this discussion as lists or bullet points.
4. Do a similar comparison of Flow Matching and Diffusion.

Note: we do not expect you to be experts on CNF and diffusion, only that you can explain some key differences. Many resources on Flow Matching also cover comparisons with the respective methods.

# 3 Flow Matching Experiments

Rules for this task:

- While we encourage you to implement Flow Matching by yourself, you are allowed to use existing implementations. Remember to properly cite and credit any external code you use. If you chose to implement most parts yourself, we will be a bit more lenient should you achieve subpar results. Still, your model should at the very least be able to generate some recognizable digits and correctly remove the noise from some digits in configuration 3 etc.
- Note that some existing implementations may use extremely big networks that can't fit on some laptops and do not train in feasible time. We have ensured that it is possible to achieve acceptable results on MNIST on common laptop hardware (terminate within 1.5 hours while running on CPU and using 3GB memory during training). So, if you experience any hardware limitations, you should try to reduce the network size or optimize slower parts of the code.

The second task is to implement and train Flow Matching with three different *configurations* on the MNIST dataset. We define each configuration by the distributions you will consider your *source* and *target* distributions (see [5] for our usage of these concepts), and how you should sample pairs from these. For all configurations, the MNIST dataset represents the target distribution and each configuration is further specified below:

1. Gaussian source distribution.
2. MNIST digit with *crop noise* (see Figure 1) matched with *random* target digit.
3. MNIST digit with *crop noise*, matched with its unaltered version.

I.e., for both configuration 2 and 3 we create one source distribution sample for each data sample by setting a randomly positioned, fixed-size square of pixels to white (see Figure 1). In configuration 3, we match the source samples with the samples they were created from, but in configuration 2 any source sample can be paired with any data samples. We have provided you with a PyTorch-based dataset wrapper for the MNIST dataset found in the *torchvision* python package that can be used for all three configurations. If you use other frameworks or programming languages, you must reimplement this code yourself.

## 3.1 Experiment Plots

For *each* configuration, provide the following plots:

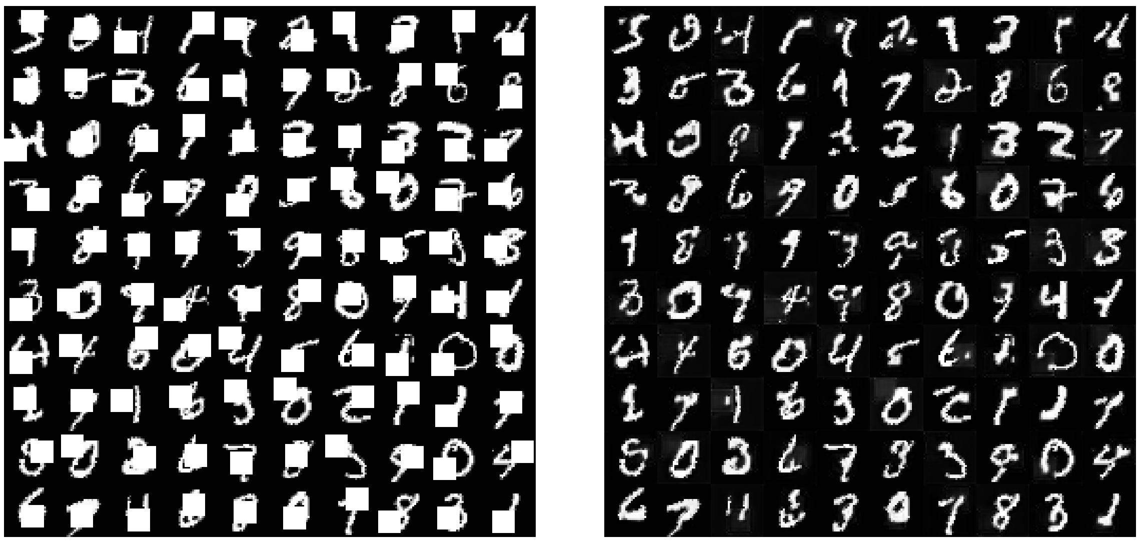

Figure 1: 100 examples of generated digits at $t = 0$ (left) and $t = 1$ (right) plotted for configuration 3. The left figure also demonstrates the *crop noise.*

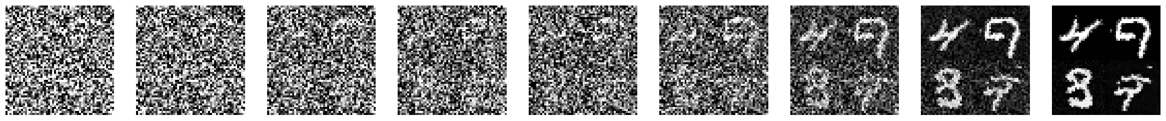

Figure 2: 4 examples of generated digits at increasing $t$ values plotted for configuration 1.

1. Generate at least 100 digits given source distribution samples (gaussian samples or MNIST samples in the test set with added crop noise). Show samples at $t = 0$ (source) and $t = 1$ (fully processed). An example is found in Figure 1.

2. Do the same for 4 digits for at least 8 different $t$ values, as seen in Figure 2.

**Note** that your model should process one digit at a time, but we want you to combine your results into bigger groups like we have done in the plots in Figures 1 and 2.

## 3.2 Experiment Discussion

1. Using the notation of [5] (Table 1), explain how $q(z)$, $\mu_t(z)$ and $\sigma_t$ are defined in your implementation. Secondly, briefly explain the architecture of the trained neural network.

2. Compare the three configurations by number of training epochs and generation quality. Does one configuration perform worse/require more training? If so, why do you think that is? Relate this discussion to [5]'s exploration of (minibatch) *Optimal Transport* in flow matching and their findings. (Hint: do you think their use of optimal transport

would pair samples like in Configuration 3/does it make sense to consider these pairings *optimal* given the task of removing crop noise?)

## 4 Submission Requirements

We expect you to submit the following:

- **Code**
  - Your code should be readable and include any necessary comments.
  - You should ensure the code is not dependent on a system-specific configuration. To ensure we can execute your code, please provide a list of packages (such as a requirements.txt file). It should be detailed in a README.md file how to install the packages and reproduce the results in the report.
  - All or part of the code may be a Jupyter notebook.
- **Report** that should include the plots and discussions asked for in tasks 1 and 2. Max. 8 pages (including figures).
  - The report should be a cohesive text with discussion about methods and results. The report should not be a collection of plots and bullet points.
  - If you do any preprocessing or changes to the methods, discuss those in the report.

Both the code and report should be in a private git repository on GitHub. The repository should have a README.md file detailing how to install necessary packages and reproduce the results in the report. Before the deadline, add the following user as a collaborator: DT8122. Do not make any changes after the deadline. Making changes after the deadline will result in an automatic fail. After adding DT8122 as a collaborator, send an email with a link to the repository to dt8122@idi.ntnu.no. Please title the email “DT8122 submission <YOUR NAME>”. **The deadline is September 14. 23:59 AoE (Anywhere on Earth)**

## References

[1] *Albergo Michael S., Vanden-Eijnden Eric.* Building Normalizing Flows with Stochastic Interpolants // The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. 2023. `https://openreview.net/forum?id=li7qeBbCR1t`.

[2] *Chen Ricky T. Q., Rubanova Yulia, Bettencourt Jesse, Duvenaud David K.* Neural Ordinary Differential Equations // Advances in Neural Information Processing Systems. 31. 2018. `https://openreview.net/forum?id=li7qeBbCR1t`.

[3] *Grathwohl Will, Chen Ricky T. Q., Bettencourt Jesse, Sutskever Ilya, Duvenaud David.* FFJORD: Free-Form Continuous Dynamics for Scalable Reversible Generative Models. // ICLR. 2019. `https://openreview.net/forum?id=rJxgknCcK7`.

[4] *Lipman Yaron, Chen Ricky T. Q., Ben-Hamu Heli, Nickel Maximilian, Le Matthew.* Flow Matching for Generative Modeling // The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. 2023. `https://openreview.net/forum?id=PqvMRDCJT9t`.

[5] *Tong Alexander, Fatras Kilian, Malkin Nikolay, Huguet Guillaume, Zhang Yanlei, Rector-Brooks Jarrid, Wolf Guy, Bengio Yoshua.* Improving and generalizing flow-based generative models with minibatch optimal transport // Trans. Mach. Learn. Res. 2024. 2024. `https://openreview.net/forum?id=CD9Snc73AW`.