02456 Deep Learning
OT conditional flow matching

Technical University of Denmark - December 21, 2023

Eline Dorothea Sigumfeldt – s183540, Christina Jensen – s194123
Aksel Beltoft – s194126, Just Peter Taudorf Lorensen – s194140

***Abstract*—Flow matching is a method for training continuous normalizing flow models. It is a method that circumvents the limitations of diffusion models by adopting a simulation-free approach. This allows for training in a broader range of possibility paths and expand on diffusion. We will be studying two methods of probability density path, diffusion (variance preserving) and optimal transport.**

I. Introduction

Think of flow matching as giving a child a box of Lego blocks (representing, e.g., Gaussian noise) which are initially scattered randomly.
Flow matching is like creating an instruction manual that teaches a child how to assemble these blocks step-by-step into a specific shape, like a car, a castle or any desired target. The method plots a clear course from the jumble of blocks to the finished structure. We're essentially creating a 'manual' that describes how to transform this starting pattern, whatever it may be, into another specific pattern or distribution.
The interesting thing here is it that doesn't have to start with a box of random Lego blocks. It could also start with an already built Lego structure of anything. Another example is Protein Folding. Beginning with a sequence of amino acids (simple pattern) and predicting the complex three-dimensional structure of a protein (complex pattern).

II. Project Goal

The objective of this project is to compare the performance between Diffusion (Variance preserving) and Optimal Transport (OT) when using flow matching for generating images using the same hyperparameters. This comparison will mainly be done by comparing the Fréchet inception distance (FID). Furthermore, images generated from the models will be presented and a discussion will also be made upon the GPU time used for training each model.
We draw inspiration and foundational knowledge from key works in the field:

1) Yaron Lipman et al.'s "Flow Matching for Generative Modeling," which introduces the concept of flow matching in generative modeling. [5]
2) Xingchao Liu et al.'s "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow," providing insights into data generation via rectified flow. [6]
3) Michael S. Albergo and Eric Vanden-Eijnden's "Building Normalizing Flows with Stochastic Interpolants," contributing to our understanding of stochastic interpolants in normalizing flows. [1]
4) Alexander Tong et al.'s "Improving and Generalizing Flow-Based Generative Models with Minibatch Optimal Transport," offering a perspective on optimizing flow-based models. [9]

These fundamental works have shaped our understanding of time-varying probability routes, vector fields, and the complexities of flow matching for data production. We sincerely thank these researchers for allowing us to further understand the subject of Conditional Flow Matching.

III. Theory

In this analogy, the Gaussian distribution is similar to the random assortment of Lego blocks. Flow matching effectively develops the 'instruction manual' or the algorithmic pathway that guides the transformation of this initial random state (Gaussian distribution) into a structured and complex final state (the target distribution).
The manual represents the flow - a time-dependent vector field, that shows how each piece moves and changes over time to form the final shape, thereby molding the initial noise distribution into the desired data distribution. This transformation journey, dictated by the flow, is the model's way of navigating through the probability density path – the route that the data takes as it transforms from its initial state to its final, complex form in the context of Continuous Normalizing Flows (CNFs).
The interesting thing here is it that doesn't have to start with a box of random Lego blocks. It could also start with an already built Lego structure of anything. Another example is Protein Folding. Beginning with a sequence of amino acids (simple pattern) and predicting the complex three-dimensional structure of a protein (complex pattern).

A. Continuous Normalizing Flows

Continuous Normalizing flows are a type of generative model that utilizes continuous transformations.

Time dependent probability paths p_t and vector fields v_t are used to describe the change from a simple distribution p_0 (in this report pure white noise) to a more complex arbitrary distribution p_1 which corresponds to a distribution similar to

our training data. Throughout this report, We refer to the training data distribution as q. The vector field v_t is used to construct a time-dependent diffeomorphic map called a flow using the ordinary differential equations (ODE's):

$$v_t(\phi_t(x)) = \frac{d}{dt}\phi_t(x) \tag{1}$$

where x denotes the data space $x \in \mathbb{R}^d$. The vector field v_t can be modelled using a neural network $v_t(x;\theta)$, where $\theta \in \mathbb{R}^d$ is the learnable parameters, which then is used to model the flow ϕ_t. This process is called a Continuous Normalizing Flow (CNF). The CNF then reshapes p_0 to p_1 using the change of variables theorem:

$$p_t = [\phi_t]_* p_0 \tag{2}$$

$$[\phi_t]_* p_0 = p_0\left(\phi_t^{-1}(x)\right) \det\left[\frac{\partial \phi_t^{-1}}{\partial x}(x)\right] \tag{3}$$

A vector field v_t generates a probability path p_t through its flow, conforming to eq. (2) as discussed in Lipman 2023, [5]. This process involves transformations $z(t)$ via a bijective function f, such that $z_1 = f(z_0(t))$. Utilizing neural ordinary differential equations (Neural ODEs), one can construct a reversible mapping from a Gaussian to a complex distribution, capturing the stochastic evolution of the diffusion process as it gradually converges on the target distribution.

B. Flow matching

Flow matching's objective is to match a target probability path, which allows flowing from the white noise distribution p_0 to the complex distribution p_1. We define the objective as minimizing the following loss function:

$$\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{t,p_t(x)} \left\| v_t(x) - u_t(x) \right\|^2 \tag{4}$$

where u_t is the vector field which generates p_t and v_t is the CNF vector field trained by a neural network, with θ as the parameters. With zero loss, the CNF model will generate p_t.

Before this loss function can be used to generate v_t, we need to have access to a closed form of u_t, which we don't have. We could construct many different probability paths that satisfy $p_1(x) \approx q(x)$. Instead, we use the conditional probability path and vector field.
Given a sample x_1, we construct a conditional probability path so $p_0(x|x_1) = p(x) = \mathcal{N}(x|0, I)$ and $p_1(x|x_1)$, where $p_1(x|x_1)$ is a normal distribution with $\mu = x_1$ and a sufficient small standard deviation $\sigma > 0$. Then we can construct the marginal probability path by marginalizing the condition probability path over q:

$$p_t(x) = \int p_t(x|x_1) q(x_1) dx_1 \tag{5}$$

$$p_1(x) = \int p_1(x|x_1) q(x_1) dx_1 \approx q(x) \tag{6}$$

Likewise, we can construct the marginal vector field:

$$u_t(x) = \int u_t(x|x_1) \frac{p_t(x|x_1) q(x_1)}{p_t(x)} dx_1 \tag{7}$$

where $u_t(x|x_1)$ is a conditional vector field that generates $p_t(x|x_1)$. These conditional vector fields $u_t(x|x_1)$ are much simpler because they only depend on a single data sample x_1. This give rise to Conditional Flow Matching, CFM. [5]

1) Conditional Flow matching: Conditional flow matching, CFM, will result in the same optima as flow matching because the loss functions have the same gradients wrt. θ. Instead of using the loss function given in eq. (4) the loss now becomes:

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t,q(x_1),p_t(x|x_1)} \left\| v_t(x) - u_t(x|x_1) \right\|^2 \tag{8}$$

By using $u_t(x|x_1)$ instead of $u_t(x)$ we can easily sample unbiased estimates from $p_t(x|x_1)$ to compute $u_t(x|x_1)$ because they are defined per sample x_1.
We define the conditional probability paths as a normal distribution:

$$p_t(x|x_1) = \mathcal{N}(x|\mu_t(x1), \sigma_t(x_1)^2 I) \tag{9}$$

where μ_t is a time dependent mean of the normal distribution and σ_t is a time dependent scalar standard deviation. At $t = 0$ we set $p_0(x|x_1) = \mathcal{N}(x|0, I)$ and at $t = 1$ we use a normal distribution for $p_1(x|x_1)$ with mean $\mu_1(x_1) = x_1$ and standard deviation $\sigma_1(x_1) = \sigma_{min}$ such that $p_1(x|x_1)$ is a normal distribution centered at x_1. The flow $\psi_t(x)$ can then be conditioned on x_1 so we get:

$$\psi_t(x) = \sigma_t(x_1)x + \mu_t(x_1) \tag{10}$$

When x is distributed as a normal distribution the flow $\psi_t(x)$ will map x to a normally-distributed random variable with mean $\mu_t(x_1)$ and std $\sigma_t(x_1)$. According to the change of variables theorem in eq. (2), $\psi_t(x)$ pushes the Gaussian noise $p_0(x|x_1) = p(x) = \mathcal{N}(x|0, I)$ to $p_t(x|x_1)$:

$$[\psi_t]_* p(x) = p_t(x|x_1) \tag{11}$$

Then this flow provides a vector field that generates the conditional probability path:

$$\frac{d}{dt}\psi_t(x) = u_t(\psi_t(x)|x_1) \tag{12}$$

Reparameterizing $p_t(x|x_1)$ in terms of just x_0 and using it in the CFM loss equation, we get:

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t,q(x_1),p_t(x_0)} \left\| v_t(\psi_t(x_0)) - \frac{d}{dt}\psi_t(x_0) \right\|^2 \tag{13}$$

We have defined the flow $\psi_t(x)$ to be a simple, invertible affine map, so we can use eq. (12) to find an expression for $u_t(x|x_1)$ in a closed form:

$$u_t(x|x_1) = \frac{\sigma_t'(x_1)}{\sigma_t(x_1)}(x - \mu_t(x_1)) + \mu_t'(x_1) \tag{14}$$

where we define the derivative with respect to time, i.e. $f' = \frac{d}{dt}f$. With this, we now know the unique vector field that defines ψ_t and generates $p_t(x|x_1)$. [5]

C. Optimal transport

Optimal transport can be explained with the help of an analogy of cheese production with import of milk. In this example, the optimal transport would be to take into each respective production capacities and supply needs. The goal of OT would then be to find a function, T, that assigns each farm to a cheese maker in an optimal way by taking into account both the distances between them and their respective demands. The way it achieves this is to generate a probability distribution for both the farmers and the cheese makers, we would then achieve two different histograms, one for the farmers and one for the cheese makers. The uses of OT would then be to change one of the histogram into the other one at the lowest possible effort, which would be the distances between the farmers and the cheese makers [8]. This distance would be calculated with the Wasserstein distance.

$$W\left(q_0, q_1\right)_2^2 = \inf_{\pi \in \Pi} \int_{\mathbb{R}^d \times \mathbb{R}^d} c(x,y)^2 d\pi(x,y) [9] \tag{15}$$

D. Diffusion

Forward model:

The idea with the forward version of diffusion is to start with data points and slowly add white noise until it approximates pure noise.

$$q\left(x_0, \ldots, x_T\right) = \prod_{t=1}^{T} q\left(x_t \mid x_{t-1}\right) \tag{16}$$

Backwards model: The main Idea, with diffusion, is to map a model of distribution (the outcome you want). However, since you do not know the ideal distribution, you would then start with a bad distribution. Then the idea of diffusion models is to turn the bad distribution slightly better and then with a large amount of iterations would we get the ideal outcome, that we were looking for.

Combination of forward and backwards:

The following step is to make a combination of the forward and backwards method. At this point, we want the generative distribution to match the observed distribution, when the iterations go to infinity. To achieve this, we make an assumption that at iteration 0, we actually have the desired distribution, which is the observed distribution (the data points).

Without a loss of generality, can we suggest that at a given time $t = T$ have added a sufficient amount of white noise to the observed distribution, so the following are achieved:

$$p(x, t = 0) = N(x|0, 1) \tag{17}$$

[3]

The objective is then to minimize the Kullback-Leibler (KL) divergence (KL divergence is a measure of how one probability distribution P diverges from the expected distribution Q. Meaning it is a measure of how different the two distributions are and, in some sense a "distance" between the two distributions), between the observed distribution and the generated distribution. This will result in an objective function which is denoted as:

$$\mathrm{L} = -\int \mathrm{p\_data}\left(\mathrm{x_0}\right) \ln \mathrm{p}\left(\mathrm{x_0}; \boldsymbol{\vartheta}\right) \mathrm{dx0} \tag{18}$$

[3]

Since the true distribution p_data is non-integrable. Then the integral can be approximated by a summation over random mini-batches from the observed dataset [3].

The challenge lies in dealing with the logarithmic term. To work around this, a fully expanded Markov form is used. Hence, the following can be written:

$$\begin{aligned} \ln p\left(x_0; \vartheta\right) &= \ln \left\{ \int dx_{1:T} p\left(x_0, \ldots, x_T\right) \right\} \\ &= \ln \left\{ \int dx_{1:T} p\left(x_T\right) \prod_{t=1}^{T} p\left(x_{t-1} \mid x_t; \theta\right) \right\} [3] \end{aligned} \tag{19}$$

The next step is to remove the integrals and simplify. To achieve this process, importance sampling is used. The idea with importance sampling is to approximate the integral as an expectation over another distribution $q(x)$, i.e., $\int f(x)dx \approx E_{x \sim q(x)}[f(x)/q(x)]$ [3].

This is where the diffusion part of diffusion models comes into play. A basic diffusion process is employed as $q\left(x_0, \ldots, x_T\right)$ to be analytically solved for use in importance sampling.

The two main distributions are defined in this context [3]:

1. Forward (destructive/diffusive) process: $q\left(x_0, \ldots, x_T\right)$. This is the diffusion that is solved analytically for importance sampling.

2. Backward (generative) process: $p\left(x_0, \ldots, x_T\right)$. This involves learnable parameters and transition kernels and is used to generate samples.

The loss can then be written as:

$$\ln p_0\left(x_0; \vartheta\right) = \ln \left\{ \int dx_{1:T} q\left(x_{1:T} \mid x_0\right) \left(p_T\left(x_T\right) \prod_{t=1}^{T} \frac{p\left(x_{t-1} \mid x_t; \theta\right)}{q\left(x_t \mid x_{t-1}\right)} \right) \right\} [3] \tag{20}$$

This is still pretty difficult to calculate, hence an approximation is done. This is achieved with the Evidence lower bounds (ELBO) method. The approximation can be written as the following:

$$\begin{aligned} \mathrm{ELBO} = \int dx_{0:T} q\left(x_{0:T}\right) \left\{ \ln \left(\frac{p_T\left(x_T\right)}{q\left(x_T \mid x_0\right)} \right) \right. \\ \left. + \ln \left(\prod_{t=2}^{T} \frac{p\left(x_{t-1} \mid x_t; \theta\right)}{q\left(x_{t-1} \mid x_t, x_0\right)} \right) + \ln p\left(x_0 \mid x_1\right) [3] \& [2] \right. \end{aligned} \tag{21}$$

Or with the Kullback-Leibler divergence term:

$$\begin{aligned} \mathrm{ELBO} = D_{\mathrm{KL}}\left(p_T\left(x_T\right) \| q\left(x_T \mid x_0\right)\right) + \\ \sum_{t=2}^{T} D_{\mathrm{KL}}\left(p\left(x_{t-1} \mid x_t; \theta\right) \| q\left(x_{t-1} \mid x_t, x_0\right)\right) + \\ \int dx_1 dx_0 q\left(x_1, x_0\right) \ln p\left(x_0 \mid x_1\right) [3] \& [2] \end{aligned} \tag{22}$$

E. Comparison between Diffusion and OT

Optimal transport focuses on efficiently transforming one probability distribution into another, often leading to more structured and interpretable models. It emphasizes direct, efficient paths in distribution transformation, often resulting in faster convergence and potentially more accurate models. On the other hand, diffusion methods involve a gradual process of adding and removing noise, simulating a Markov chain (A Markov chain process is a statistical model where the probability of each event depends only on the state attained in the previous event). This can lead to models that are more flexible and capable of handling complex, high-dimensional data, but they may require more computational resources and time.

The figure below shows conditional flow matching using diffusion and optimal transport, 1: In figure 1, we see the

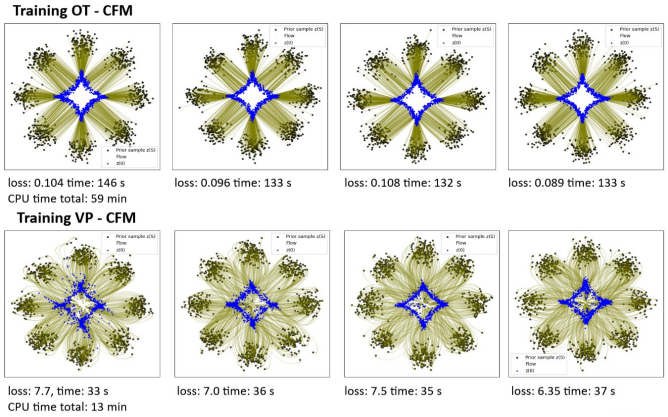

Fig. 1: Flow patterns for OT and Variance preserving diffusion

different training iterations of the models and the varying loss as the model converges from the Gaussian moons (the prior distribution) distribution, to a more complex distribution indicated with blue points. The OT-CFM appears to converge faster and with lower loss compared to the VP-CFM, which suggests a more efficient training process. The flow lines in the diagrams trace the path that points from the prior distribution take to become samples from the target distribution. With VP-CFM the initial stages show more randomness, taking exploratory paths as the model sift possibilities. Over time, it "learns" to identify and follow the more promising trajectories toward target distribution.

F. U-NET

U-Net model is a type of neural network architecture commonly used for image segmentation tasks. It is called U-Net, since it has an u shaped architecture. U-Net models consist of a contracting path and an expansive path. Where the contracting path is a convolution network that consist of a repeated application of convolution followed by a ReLu and a max pooling operation. During this step, the spatial information is reduced while increasing the feature information. The expansive path is responsible for restoring the spatial resolution of the image while combining it with features learned from the contracting path, this is achieved through a sequence of up-convolutions. To preserve the details during the upsampling process, U-Net uses skip connections. Which skip one or more layers in the contracting path and concatenate the feature maps with the corresponding layers in the expansive path to preserve the details.

The model used for this poster/paper is 4 steps deep with the following multiplication (1,2,4,8)

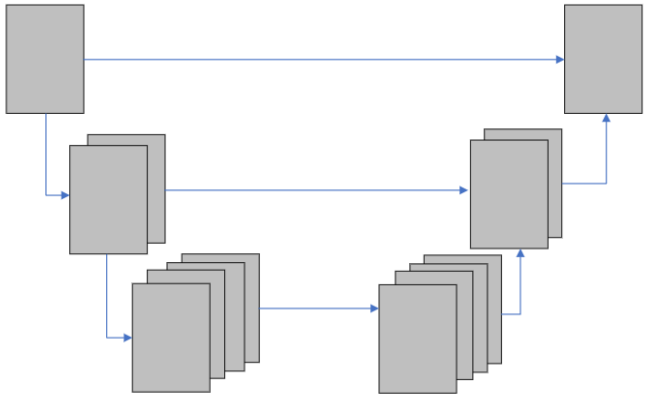

Fig. 2: U-NET architecture structure

G. FID Score

The Fréchet Inception distance (FID) score measures the similarity between the distribution of real images (from a dataset) and the distribution of generated images produced by a model.

$$\text{FID} = |\mu_1 - \mu_2| + \text{Tr}\left(\sigma_1 + \sigma_2 - 2\sqrt{\sigma_1 * \sigma_2}\right) \tag{23}$$

[7] A lower FID score indicates that the generated images are closer in terms of distribution to the real images, suggesting higher quality and realism in the generated samples.

IV. Method

Our objective is to generate images which have the same distributions as the CIFAR-10[4] data set, which consists of 60,000 32x32 pixel images in 10 classes. The OT and diffusion models are trained on CIFAR-10 with tools made by Tong, A. et. al. from their paper [9]. Tong provides a Github repository with a command line program for training both OT and diffusion models, along with a function to evaluate the FID-score of the resulting models. The number of iterations was chosen based on our available storage which was 30 GB on an HPC which corresponds to approximately 40,000 iterations for each model. The hyperparameters were selected based on [9] and are shown in Table I. The learning rate in the table is the target learning rate lr_t which follows the learning rate schedule, which can be described as:

$$lr(n) = lr_t \cdot \frac{\min(n, 5000)}{5000} \tag{24}$$

where n is the iteration. In this manner the learning rate grows the first 5000 iterations which prevents overshooting the optimal values and explore the solution space more carefully.

lr_t	Batch size	Iterations
10^{-4}	128	40,000

TABLE I: Hyperparameters for both the OT and diffusion model.

Slight modifications of the code are made, which is available at Github.

To compute the FID-score, 50,000 images from the CIFAR-10 data set is used to create the true distribution. The distribution of sampled images from the trained model is then compared with the FID-score as described in subsection III-G.

V. Results

In the following chapter, the results are shown for the two different methods. The chapter will start with the results from the variance preserving diffusion model, Optimal transport model and end with the FID score and GPU time used to train the models.

A. Variance preserving

We generate 64 samples every 20,000 training iteration of the two models. The samples from the diffusion model can be seen in fig. (3).

After 0 iterations, we are at $p_0(x)$ distribution, so all the samples are generated from white noise. After 20,000 we begin to see actual shapes and after 40,000 we begin to see the classes Cifar10 contains, i.e. cars, ships, frogs, and horses. The execution time of training the model was 4 hours and 19 minutes.

B. Optimal Transport

The samples from the diffusion model can be seen in fig. (4).

Just like the diffusion model, we start with pure noise after 0 iterations. After 20,000 we begin to see actual shapes and after 40,000 we begin to see the classes CIFAR-10 contains. With the naked eye, there isn't a big difference between the samples generated by the OT and the diffusion model. The execution time of training the OT model was 4 hours and 6 minutes.

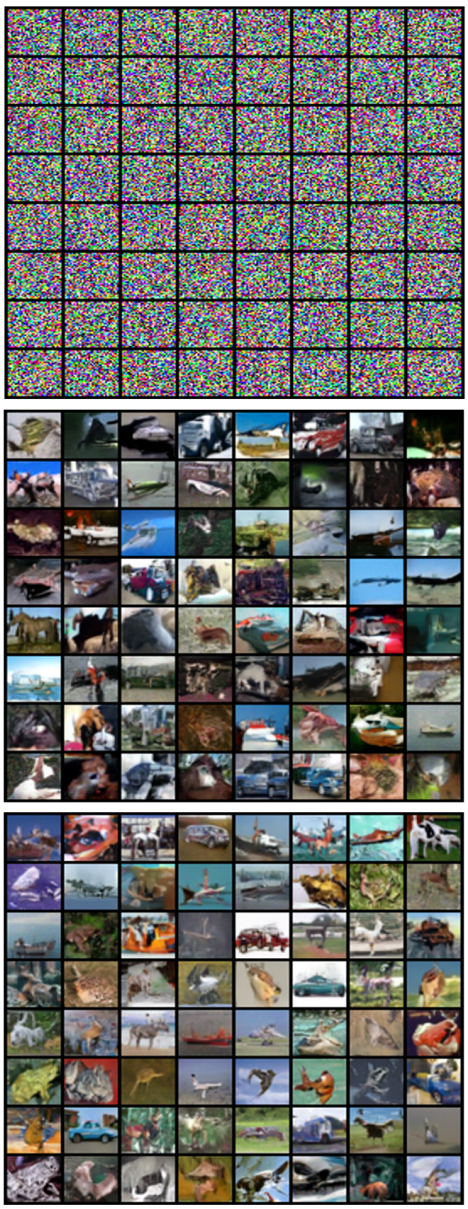

Fig. 3: Samples from the diffusion model. From the top: after 0, 20,000 and 40,000 iterations

C. FID-score

The measure used for evaluating the performance of the two models is the FID score, as mentioned in subsection III-G. After training our model with 40,000 iterations, our FID scores are as shown in Table II.

Model	OT	Diffusion
FID	47.2	85.1

TABLE II: FID score for the two models after 40,000 iterations.

The FID score shows a clear difference between the OT and diffusion model, as contrary to the generated images seen in fig. (4) and (3). The score for OT is $\frac{85.1-47.2}{85.1} = 45\%$ smaller than the score for diffusion. If we compare these scores with the execution time for training the models, which was 4 hours and 6 minutes for the OT model and 4 hours and 19 minutes for the diffusion model, OT both has the best execution time and FID score.

VI. Discussion

Comparing our FID-score with the result of Tong, A. produced in [9] where $FID = 3.5$ there is a huge difference. There might be different reasons for the difference. One major explanation is the number of iterations used. Tong uses 500,000 iterations, whereas we were only able to iterate 40,000 times due to storage limitations.

Fig. 4: Samples from the OT model. From the top: after 0, 20,000 and 40,000 iterations

Since our model had fewer iterations, it could be argued that the learning rate and learning rate schedule should be different from Tong's due to the model not converging to a stable solution fast enough. By increasing the learning rate, the optimal solution might be reached earlier. In future work, a new learning rate schedule could be introduced where the learning rate starts small to avoid overshooting, then increases to reach the solution faster and lastly decreases to slowly converge to the optimal solution.

Another option for future work could be to set a condition for the termination of the training. In our study, we used the condition that the number of iterations should be equal to 40,000. Another possibility is to keep score of the validation loss. A termination condition could be to stop when the validation loss has converged.

VII. Conclusion

From this report, it can be concluded that using Optimal Transport (OT) in CFM is superior to diffusion when evaluating on GPU time and FID score. The FID score for OT was 45% smaller than for diffusion, with the score being 47.2 for OT and 85.1 for diffusion. The GPU execution time was almost the same for training the two models, but OT was slightly better with a GPU time of 4 hours and 6 minutes compared to 4 hours and 19 minutes for the diffusion model. Compared to the FID score found by Tong, A. in [9] where $FID = 3.5$ for OT, our model's FID score was much larger. As discussed in section VI, one of the reasons for this may be a shorter training.

References

[1] Michael S. Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants, 2023.

[2] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020.

[3] Jonathan Kernes. Diffusion models, Dec. 13, 2022.

[4] Alex Krizhevsky. Learning multiple layers of features from tiny images, 2009.

[5] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling, 2023.

[6] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow, 2022.

[7] Pytorch.org. Fid command, Dec. 4, 2023.

[8] Ievgen Redko. Optimal transport: a hidden gem that empowers today's machine learning, Jun. 15, 2020.

[9] Alexander Tong, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrid Rector-Brooks, Kilian Fatras, Guy Wolf, and Yoshua Bengio. Improving and generalizing flow-based generative models with minibatch optimal transport, 2023.