Notes on Diffusion Models

Erik Paul

November 26, 2023

1 Introduction

Given data points $x_1, \ldots, x_n$, our aim is to sample new data points which are in some way similar to $x_1, \ldots, x_n$. For instance, the given data points may be images of cats and our goal is to sample new images of cats.

Our assumption is that $x_1, \ldots, x_n$ are samples of a random variable $\mathbf{x}$ with an unknown underlying probability distribution $p^*(\mathbf{x})$ over a known space of possible data points, e.g., the space of all images of a certain size. We attempt to approximate $\mathbf{x}$ by defining a family of probability distributions p_θ, where θ are the model parameters, to sample from and determine θ such that p_θ approximates p^* as good as possible. Our known data points are used to assess how well p_θ approximates p^* for a given set of parameters θ.

For denoising diffusion probabilistic models (DDPMs) [18, 52], we define a Markov chain by gradually adding noise to our training samples until they are (more or less) normally distributed noise and then attempt to model the reversal of this diffusion process. Sampling is achieved by generating random noise and applying the reverse process.

Scope In the following, we cover the original paper on denoising diffusion models [18] which is based on [52], several papers on architecture improvements including guidance mechanisms [8, 17, 35], the deterministic sampling scheme DDIM [53], cascaded diffusion models [19], the conditional diffusion models GLIDE [36], LDM [44], unCLIP (DALL-E 2) [41], and Imagen [47], and the fine-tuning techniques LoRA [20], Textual Inversion [13], and DreamBooth [46].

Related surveys Apart from the cited papers, the contents of the following notes draw major inspirations from Lilian Weng's blog post [60], the survey by Calvin Luo [33], and the blog post by Sergios Karagiannakos and Nikolaos Adaloglou [23] on the topic. It is also worth to mention the blog post by Niels Rogge and Kashif Rasul [43] which provides detailed insights into the official implementation of the DDPM model. For introductions to the closely related score based models and their connection to DDPMs, we recommend the introductions by Yang Song [54] and by Ayan Das [5, 6]. For an extensive overview of variants and applications of diffusion models, see the survey by Cao et al. [4].

Contents

2 Preliminaries

In the following, we assume that the reader is familiar with basic concepts of probability theory, including the normal distribution $\mathcal{N}(\mu, \sigma^2)$ with density

$$\mathcal{N}(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

We recall that for $X \sim \mathcal{N}(\mu, \sigma^2)$ and $a, b \in \mathbb{R}$, we have $aX + b \sim \mathcal{N}(a\mu + b, a^2\sigma^2)$. Moreover, for $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$, we have $X_1 + X_2 \sim \mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$.

A real random vector $\mathbf{X} = (X_1, \dots, X_k)^{\mathrm{T}}$ is called a *normal random vector* if there exists a vector $\boldsymbol{\mu} \in \mathbb{R}^k$ and a matrix $\mathbf{A} \in \mathbb{R}^{k \times l}$ such that $\mathbf{X} = \mathbf{A}\mathbf{Z} + \boldsymbol{\mu}$, where $\mathbf{Z} = (Z_1, \dots, Z_l)$ is a vector of independent normally distributed variables, i.e., $Z_i \sim \mathcal{N}(0, 1)$ for all $i = 1, \dots, l$. In this case, we write $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with the *covariance matrix* $\boldsymbol{\Sigma} = \mathbf{A}\mathbf{A}^{\mathrm{T}}$. Note that $\operatorname{Cov}[X_i, X_j] = \mathbb{E}[(\sum_k a_{ik} Z_k) \cdot (\sum_k a_{jk} Z_k)] = \sum_k a_{ik} a_{jk} \mathbb{E}[Z_k^2] = \boldsymbol{\Sigma}_{ij}$, as the expected value is linear, $\mathbb{E}[Z_k^2] = \operatorname{Var}[Z_k] = 1$, and $\mathbb{E}[Z_k Z_l] = \mathbb{E}[Z_k]\mathbb{E}[Z_l] = 0$ for $k \neq l$ since Z_k and Z_l are independent.

The density for $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is given by

$$\mathcal{N}(\mathbf{X}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi \det \boldsymbol{\Sigma})^k}} \exp\left(-\frac{1}{2}(\mathbf{X} - \boldsymbol{\mu})^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} (\mathbf{X} - \boldsymbol{\mu})\right).$$

If $\mathbf{X} \sim \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2\end{bmatrix}, \begin{pmatrix}\boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22}\end{pmatrix}\right)$, then

$$\mathbf{X}_1 \mid \mathbf{X}_2 \sim \mathcal{N}(\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\mathbf{X}_2 - \boldsymbol{\mu}_2), \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21})$$

3 The Mathematical Model

We model p_θ as a marginal distribution $p_\theta(\mathbf{x}_0) = \int p_\theta(\mathbf{x}_{0:T}) d\mathbf{x}_{1:T}$, where $\mathbf{x}_1, \ldots, \mathbf{x}_T$ are latent variables of the same dimensionality as $\mathbf{x}$. We define the *forward diffusion process* by

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}), \qquad q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}),$$

where $\beta_1, \ldots, \beta_T$ is a variance schedule of small positive variances. The forward process is easy to sample from and is used to create the training set for learning the reverse process. The variances β_t are usually held constant as hyperparameters but can also be learned. For our model family p_θ, we choose $p_\theta(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; 0, \mathbf{I})$ and

$$p_\theta(\mathbf{x}_{0:T}) = p_\theta(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t), \quad p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)),$$

i.e., we assume the reverse process can also be modeled by Gaussians. Here, $\boldsymbol{\mu}_\theta$ and $\boldsymbol{\Sigma}_\theta$ are modeled using neural networks.

The forward process admits sampling $\mathbf{x}_t$ at an arbitrary timestep t by defining $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$, as then

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}).$$

This is clear by definition for $t = 1$ and if it is true for $t-1$, then with $\boldsymbol{\varepsilon}_{t-1}, \boldsymbol{\varepsilon}_t \sim \mathcal{N}(0, \mathbf{I})$ independently distributed, we have

$$\begin{aligned}
\mathbf{x}_t &\sim \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\boldsymbol{\varepsilon}_t \\
&\sim \sqrt{1-\beta_t}(\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}}\boldsymbol{\varepsilon}_{t-1}) + \sqrt{\beta_t}\boldsymbol{\varepsilon}_t \\
&\sim \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (\alpha_t - \bar{\alpha}_t + 1 - \alpha_t)\mathbf{I}).
\end{aligned}$$

We choose β_t such that $\bar{\alpha}_t \overset{t\to\infty}{\longrightarrow} 0$, so the distribution of $\mathbf{x}_t$ approaches $\mathcal{N}(0, \mathbf{I})$ for $t \to \infty$.

Properties for Fixed $\mathbf{x}_0$ The distribution $q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0)$ can be computed analytically, as we know that

$$\begin{bmatrix} \mathbf{x}_{t-1} \\ \mathbf{x}_t \end{bmatrix} \Bigg| \mathbf{x}_0 \sim \mathcal{N}\left(\begin{bmatrix} \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 \\ \sqrt{\bar{\alpha}_t}\mathbf{x}_0 \end{bmatrix}, \begin{bmatrix} (1-\bar{\alpha}_{t-1})\mathbf{I} & \sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})\mathbf{I} \\ \sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})\mathbf{I} & (1-\bar{\alpha}_t)\mathbf{I} \end{bmatrix} \right)$$

since

$$\begin{aligned}
\mathbf{x}_{t-1,i} &\sim \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_{0,i} + \sqrt{1-\bar{\alpha}_{t-1}}\boldsymbol{\varepsilon}_{t-1,i} \\
\mathbf{x}_{t,j} &\sim \sqrt{\bar{\alpha}_t}\mathbf{x}_{0,j} + \sqrt{\alpha_t}\sqrt{1-\bar{\alpha}_{t-1}}\boldsymbol{\varepsilon}_{t-1,j} + \sqrt{\beta_t}\boldsymbol{\varepsilon}_{t,j}
\end{aligned}$$

so

$$\begin{aligned}
\mathrm{Cov}(\mathbf{x}_{t-1,i}, \mathbf{x}_{t,j}) &= \mathbb{E}\left[\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})\boldsymbol{\varepsilon}_{t-1,i}\boldsymbol{\varepsilon}_{t-1,j} + \sqrt{1-\bar{\alpha}_{t-1}}\sqrt{\beta_t}\boldsymbol{\varepsilon}_{t-1,i}\boldsymbol{\varepsilon}_{t,j}\right] \\
&= \begin{cases} \sqrt{\alpha_t}(1-\bar{\alpha}_{t-1}) & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}
\end{aligned}$$

Thus, we can compute

$$
\begin{aligned}
&q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \\
&= \mathcal{N}\left(\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\mathbf{x}_2 - \boldsymbol{\mu}_2), \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}\right) \\
&= \mathcal{N}\left(\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{\alpha_t}\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0), 1 - \bar{\alpha}_{t-1} - \alpha_t\frac{(1-\bar{\alpha}_{t-1})^2}{1-\bar{\alpha}_t}\mathbf{I}\right) \\
&= \mathcal{N}\left(\underbrace{\sqrt{\alpha_t}\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0}_{\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t,\mathbf{x}_0)}, \underbrace{\beta_t\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}}_{\tilde{\beta}_t}\mathbf{I}\right).
\end{aligned}
$$

Note that if we sample $\mathbf{x}_t$ through $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}_t$, we can rewrite $\tilde{\boldsymbol{\mu}}_t$ as

$$
\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \boldsymbol{\varepsilon}_t) = \sqrt{\alpha_t}\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\left(\frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}_t}{\sqrt{\bar{\alpha}_t}}\right) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\varepsilon}_t\right)
$$

Properties for normally distributed $\mathbf{x}_0$ If we assume that $\mathbf{x}_0 \sim \mathcal{N}(0, \mathbf{I})$, we can explicitly compute

$$
\begin{aligned}
q(\mathbf{x}_t) &= \int q(\mathbf{x}_t|\mathbf{x}_0)q(\mathbf{x}_0)d\mathbf{x}_0 \\
&= \frac{1}{\sqrt{(2\pi(1-\bar{\alpha}_t))^k}}\frac{1}{\sqrt{(2\pi)^k}}\int \exp\left(-\frac{||\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0||_2^2}{2(1-\bar{\alpha}_t)}\right)\exp\left(-\frac{||\mathbf{x}_0||_2^2}{2}\right)d\mathbf{x}_0 \\
&= C\int \exp\left(-\frac{||\mathbf{x}_t||_2^2 - 2\sqrt{\bar{\alpha}_t}\mathbf{x}_t^{\mathrm{T}}\mathbf{x}_0 + \bar{\alpha}_t||\mathbf{x}_0||_2^2 + (1-\bar{\alpha}_t)||\mathbf{x}_0||_2^2}{2(1-\bar{\alpha}_t)}\right)d\mathbf{x}_0 \\
&= C\int \exp\left(-\frac{||\mathbf{x}_0 - \sqrt{\bar{\alpha}_t}\mathbf{x}_t||_2^2 + (1-\bar{\alpha}_t)||\mathbf{x}_t||_2^2}{2(1-\bar{\alpha}_t)}\right)d\mathbf{x}_0 \\
&= \underbrace{\frac{1}{\sqrt{(2\pi)^k}}\exp\left(-\frac{||\mathbf{x}_t||_2^2}{2}\right)}_{=\mathcal{N}(\mathbf{x}_t;0,\mathbf{I})} \cdot \underbrace{\frac{1}{\sqrt{(2\pi(1-\bar{\alpha}_t))^k}}\int \exp\left(-\frac{||\mathbf{x}_0 - \sqrt{\bar{\alpha}_t}\mathbf{x}_t||_2^2}{2(1-\bar{\alpha}_t)}\right)d\mathbf{x}_0}_{=1}.
\end{aligned}
$$

Thus, we see from

$$
\mathbf{x}_{t-1} \sim \boldsymbol{\varepsilon}_{t-1} \qquad\qquad \mathbf{x}_t \sim \sqrt{1-\beta_t}\boldsymbol{\varepsilon}_{t-1} + \sqrt{\beta_t}\boldsymbol{\varepsilon}_t
$$

that $\mathrm{Cov}(\mathbf{x}_{t-1}, \mathbf{x}_t) = \sqrt{1-\beta_t}\mathbf{I}$, so $\mathbf{x}_{t-1}|\mathbf{x}_t \sim \mathcal{N}(\sqrt{1-\beta_t}\mathbf{x}_t, \beta_t\mathbf{I})$.

4 The Loss Function

Our goal is to maximize the likelihood $p_\theta(\mathbf{x}_0)$ over our dataset. For this, we derive the *evidence lower bound (ELBO)* as in [28].

$$\begin{aligned}
-\log p_\theta(\mathbf{x}_0) &= \mathbb{E}_{\mathbf{x}_{1:T}\sim q(\cdot|\mathbf{x}_0)}[-\log p_\theta(\mathbf{x}_0)] \\
&= \mathbb{E}_{\mathbf{x}_{1:T}\sim q(\cdot|\mathbf{x}_0)}\left[\log\left[\frac{p_\theta(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right]\right] \\
&= \mathbb{E}_{\mathbf{x}_{1:T}\sim q(\cdot|\mathbf{x}_0)}\left[\log\left[\frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\frac{p_\theta(\mathbf{x}_{1:T}|\mathbf{x}_0)}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right]\right] \\
&= \underbrace{\mathbb{E}_{\mathbf{x}_{1:T}\sim q(\cdot|\mathbf{x}_0)}\left[\log\left[\frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right]\right]}_{\substack{=\mathcal{L}_\theta(\mathbf{x}_0)\\ \text{(ELBO)}}} - \underbrace{\mathbb{E}_{\mathbf{x}_{1:T}\sim q(\cdot|\mathbf{x}_0)}\left[\log\left[\frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right]\right]}_{=D_{\mathrm{KL}}(q(\mathbf{x}_{1:T}|\mathbf{x}_0)||p_\theta(\mathbf{x}_{1:T}|\mathbf{x}_0))}
\end{aligned}$$

We can rewrite the ELBO as

$$\begin{aligned}
\mathcal{L}_\theta(\mathbf{x}_0) &= \mathbb{E}_{\mathbf{x}_{1:T}\sim q(\cdot|\mathbf{x}_0)}\left[\log\left[\frac{\prod_{t=1}^T q(\mathbf{x}_t|\mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_T)\prod_{t=1}^T p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}\right]\right] \\
&= \mathbb{E}_{\mathbf{x}_{1:T}\sim q(\cdot|\mathbf{x}_0)}\left[-\log p_\theta(\mathbf{x}_T) + \log\left[\frac{q(\mathbf{x}_1|\mathbf{x}_0)}{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}\right] + \sum_{t=2}^T \log\left[\frac{q(\mathbf{x}_t|\mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}\right]\right].
\end{aligned}$$

Now note that

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = q(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{x}_0) = \frac{q(\mathbf{x}_t,\mathbf{x}_{t-1},\mathbf{x}_0)}{q(\mathbf{x}_{t-1},\mathbf{x}_0)} = \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)q(\mathbf{x}_t,\mathbf{x}_0)}{q(\mathbf{x}_{t-1},\mathbf{x}_0)} = \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)q(\mathbf{x}_t|\mathbf{x}_0)}{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}$$

so we can write

$$\begin{aligned}
\sum_{t=2}^T \log\left[\frac{q(\mathbf{x}_t|\mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}\right] &= \sum_{t=2}^T \log\left[\frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}\right] + \sum_{t=2}^T \log\left[\frac{q(\mathbf{x}_t|\mathbf{x}_0)}{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}\right] \\
&= \sum_{t=2}^T \log\left[\frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}\right] + \log\left[\frac{q(\mathbf{x}_T|\mathbf{x}_0)}{q(\mathbf{x}_1|\mathbf{x}_0)}\right]
\end{aligned}$$

and obtain

$$\begin{aligned}
\mathcal{L}_\theta(\mathbf{x}_0) &= \mathbb{E}_{\mathbf{x}_{1:T}\sim q(\cdot|\mathbf{x}_0)}\left[\log\left[\frac{q(\mathbf{x}_T|\mathbf{x}_0)}{p_\theta(\mathbf{x}_T)}\right] + \sum_{t=2}^T \log\left[\frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}\right] - \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)\right] \\
&= \underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0)||p_\theta(\mathbf{x}_T))}_{L_T} + \sum_{t=2}^T \underbrace{\mathbb{E}_{\mathbf{x}_t\sim q(\cdot|\mathbf{x}_0)}\left[D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))\right]}_{L_{t-1}} \\
&\quad - \underbrace{\mathbb{E}_{\mathbf{x}_1\sim q(\cdot|\mathbf{x}_0)}\left[\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)\right]}_{L_0}.
\end{aligned}$$

For the correctness of the KL terms, note that

$$q(\mathbf{x}_t,\mathbf{x}_{t-1}|\mathbf{x}_0) = \frac{q(\mathbf{x}_t,\mathbf{x}_0)}{q(\mathbf{x}_t,\mathbf{x}_0)}\frac{q(\mathbf{x}_t,\mathbf{x}_{t-1},\mathbf{x}_0)}{q(\mathbf{x}_0)} = q(\mathbf{x}_t|\mathbf{x}_0)q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0).$$

The Term L_T is the *prior matching term* and ensures that the forward process destroys all information about $\mathbf{x}_0$ such that $q(\mathbf{x}_T) \approx \mathcal{N}(\mathbf{x}_T; 0, \mathbf{I})$. If the variances β_t are treated as hyperparameters, L_T is constant as q has no learnable parameters and $p_\theta(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; 0, \mathbf{I})$, so it can be ignored during training. Moreover, by design we choose β_t and T such that $q(\mathbf{x}_T) \approx \mathcal{N}(\mathbf{x}_T; 0, \mathbf{I})$ so that L_T is (close to) zero anyway. If the variance schedule is learned, the term is crucial to ensure that the two processes do not cheat by modeling the identity function.

The terms L_{t-1} are the *denoising matching terms* which force the reverse process to reverse the forward process as good as possible. The KL-terms compare Gaussians with known distributions and can be computed analytically.

The term L_0 is the *reconstruction term* which ensures that the reverse process attempts to produce data points which mimic our training examples $\mathbf{x}_0$. In [18], L_0, and thus the sampler from $\mathbf{x}_1$ to $\mathbf{x}_0$, is modeled using a separate discrete decoder derived from $\mathcal{N}(\mathbf{x}_0; \boldsymbol{\mu}_\theta(\mathbf{x}_1, 1), \boldsymbol{\Sigma}_\theta(\mathbf{x}_1, 1))$, i.e., as a parameterless function of $\boldsymbol{\mu}_\theta(\mathbf{x}_1, 1)$ and $\boldsymbol{\Sigma}_\theta(\mathbf{x}_1, 1)$, c.f. Section 5.

5 Simplifications and Hyperparameter Choices

The Variance Schedule β_t In the original paper [18], the variances β_t are scheduled to increase linearly from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$, where $T = 1000$. The variances are deliberately chosen small w.r.t. the data from $[-1, 1]$ so that the forward process creates only minor distortions and the assumption that the reverse process can be modeled using Gaussians is defensible. Also, this choice ensures that the distribution $q(\mathbf{x}_T|\mathbf{x}_0)$ of $\mathbf{x}_T$ is reasonably close to $\mathcal{N}(0, \mathbf{I})$, being reported as $L_T = D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0)||\mathcal{N}(0, \mathbf{I})) \approx 10^{-5}$ bits per dimension in the experiments.

Parameterizing $\boldsymbol{\Sigma}_\theta$ The authors of [18] found that learning a diagonal reverse process covariance matrix $\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)$ lead to unstable training and worse sample quality and fixed $\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t) = \sigma_t^2 \mathbf{I}$ as hyperparameters with either $\sigma_t^2 = \beta_t$ or $\sigma_t^2 = \tilde{\beta}_t$, both of which choices lead to similar results. Note that these choices of σ_t^2 correspond to the variances of $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ for $\mathbf{x}_0 \sim \mathcal{N}(0, \mathbf{I})$ and $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ for $\mathbf{x}_0$ fixed, respectively.

Parameterizing L_0 In the original paper [18], the data $\mathbf{x}_0$ to be generated are images, so the decoder sampling from $p_\theta(\mathbf{x}_0|\mathbf{x}_1)$ should produce valid images. For this, the image data is assumed to have integer values in $\{0, \ldots, 255\}$ and is scaled linearly to $[-1, 1]$ to match the standard local prior $p_\theta(\mathbf{x}_T) \sim \mathcal{N}(0, \mathbf{I})$. To obtain a discrete decoder, $p_\theta(\mathbf{x}_0|\mathbf{x}_1)$ is modeled as

$$p_\theta(\mathbf{x}_0|\mathbf{x}_1) = \prod_{i=1}^{D} \int_{\delta_-(\mathbf{x}_{0,i})}^{\delta_+(\mathbf{x}_{0,i})} \mathcal{N}(x; \boldsymbol{\mu}_\theta(\mathbf{x}_1, 1)_i, \sigma_1^2) dx$$

$$\delta_+(x) = \begin{cases} \infty & \text{if } x = 1 \\ x + \frac{1}{255} & \text{if } x < 1 \end{cases} \qquad \delta_-(x) = \begin{cases} -\infty & \text{if } x = -1 \\ x - \frac{1}{255} & \text{if } x > -1 \end{cases}$$

where D is the dimensionality of the image. In other words, we measure for each pixel $\mathbf{x}_{0,i}$ how much probability mass of $\mathcal{N}(\boldsymbol{\mu}_\theta(\mathbf{x}_1, 1)_i, \sigma_1^2)$ is in the correct ground truth bin for that pixel. Our model thus has to predict a mean vector $\boldsymbol{\mu}_\theta(\mathbf{x}_1, 1)$. In [18], the final sampling step outputs $\boldsymbol{\mu}_\theta(\mathbf{x}_1, 1)$ noiselessly and the training loss is modeled with a loss term as described in the next section.

Simplifying L_t for $t > 1$ The KL-divergence $D_{\mathrm{KL}}(\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) \parallel \mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2))$ of two k-dimensional multivariate Gaussians is

$$\frac{1}{2}\left(\mathrm{tr}(\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\Sigma}_1) - k + (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^{\mathrm{T}}\boldsymbol{\Sigma}_2^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1) + \log\left(\frac{\det \boldsymbol{\Sigma}_2}{\det \boldsymbol{\Sigma}_1}\right)\right).$$

Thus, we obtain

$$\begin{aligned} D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) &= D_{\mathrm{KL}}(\mathcal{N}(\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t\mathbf{I}) \parallel \mathcal{N}(\boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \sigma_t^2\mathbf{I})) \\ &= \frac{1}{2\sigma_t^2}||\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) - \boldsymbol{\mu}_\theta(\mathbf{x}_t, t)||_2^2 + C \end{aligned}$$

for some constant C which does not depend on θ. Thus, L_t becomes

$$L_t = \mathbb{E}_{\mathbf{x}_t \sim q(\cdot|\mathbf{x}_0)}\left[\frac{1}{2\sigma_t^2}||\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) - \boldsymbol{\mu}_\theta(\mathbf{x}_t, t)||_2^2\right] + C.$$

During training, we may assume that $\mathbf{x}_t$ is sampled via $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}_t$ so that

$$\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \boldsymbol{\varepsilon}_t) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\varepsilon}_t\right).$$

This motivates the parameterization

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t)\right)$$

such that L_t becomes

$$L_t = \mathbb{E}_{\mathbf{x}_0 \sim p^*, \boldsymbol{\varepsilon}_t \sim \mathcal{N}(0,\mathbf{I})}\left[\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}||\boldsymbol{\varepsilon}_t - \boldsymbol{\varepsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}_t, t)||_2^2\right] + C.$$

As this loss essentially minimizes the distance between $\boldsymbol{\varepsilon}_t$ and $\boldsymbol{\varepsilon}_\theta$, the authors of [18] propose a further simplification to

$$L_{\mathrm{simple}}(\theta) = \mathbb{E}_{t \sim \mathcal{U}, \mathbf{x}_0 \sim p^*, \boldsymbol{\varepsilon} \sim \mathcal{N}(0,\mathbf{I})}\left[||\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}, t)||_2^2\right],$$

where $\mathcal{U}$ is the uniform distribution on $0, \ldots, T-1$. The authors report better performance with this choice than when predicting $\boldsymbol{\varepsilon}$ or $\tilde{\boldsymbol{\mu}}$ using the more complex loss terms. For $t > 1$, the loss term corresponds to a rescaling of the loss term derived above. For $t = 0$, the loss term corresponds to an approximation of the integral in L_0 by the Gaussian probability density function at $\mathbf{x}_{0,i}$, scaled by the bin width and ignoring scaling factors like σ_1^2.

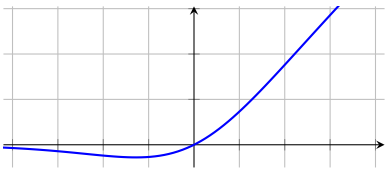
In conclusion, during training we pick a training sample $\mathbf{x}_0$, sample $t \sim \mathcal{U}([0, T-1])$ and $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \mathbf{I})$, and then perform a gradient descent step on $||\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}, t)||_2^2$. Recall that mathematically, this corresponds to applying gradient descent on a Monte Carlo estimate of the expected value in L_{simple}, i.e., we apply the usual stochastic gradient descent.

For sampling, we first sample $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$, then repeatedly sample $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$ and $\mathbf{x}_{t-1} = \boldsymbol{\mu}_\theta(\mathbf{x}_t, t) + \sigma_t \mathbf{z} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$ up to $\mathbf{x}_1$ and output $\mathbf{x}_0 = \boldsymbol{\mu}_\theta(\mathbf{x}_1, 1)$. Note that during sampling, the model means $\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$ are always clipped to $[-1, 1]$, the valid interval of pixel value bins.

6 The U-Net Backbone

In this section, we review the implementation of the neural network which predicts $\boldsymbol{\varepsilon}_\theta$. The overall architecture is a *U-Net* [45], following the PixelCNN++ [49] backbone based on a Wide ResNet [63]. For extended details, the original implementation is available on GitHub[1]. Throughout the network, the U-Net operates on tensors of shape (B, H, W, C). Here, B denotes the batch size (fixed throughout the network), H and W are the height and width of the image (downscaled and then upscaled throughout the network), and C is the number of feature channels (upscaled and then downscaled throughout the network). The input to the network is a batch of noisy images, i.e., a tensor of shape $(B, H, W, 3)$, and the output is a tensor of the same shape containing predictions for $\boldsymbol{\varepsilon}_\theta$. Specifically, $B = 128$ was chosen for images of size 32×32 and $B = 64$ for larger images.

The Activation Function The activation function of choice is *Swish* [40], defined by $\text{swish}(x) = \frac{x}{1+e^{-x}}$ and depicted below.



Positional Encoding The network uses temporal embeddings very similar to those of the original transformer paper [59]. More precisely, the positional encoding vector τ_t at time t of dimension $2d$ (here $2d = 128$) is defined as

$$\eta_i^t = t \cdot 10^{-\frac{4i}{d-1}} = \frac{t}{10000^{\frac{i}{d-1}}} \qquad (i = 0, \ldots, d-1)$$
$$\tau_t = (\sin \eta_0^t, \sin \eta_1^t, \ldots, \sin \eta_{d-1}^t, \cos \eta_0^t, \cos \eta_1^t, \ldots, \cos \eta_{d-1}^t).$$

Dropout A *dropout* layer [16] randomly sets a fraction p of the weights of its input tensor to 0. Here, p was set to 0.1 for the system trained on the CIFAR10 data set and to 0 on all other data sets (CelebA-HQ 256×256, LSUN) in all dropout layers.

[1] `https://github.com/hojonathanho/diffusion`

Group Normalization The network makes repeated use of *group normalization* [61]. A group normalization layer groups the channels of a batch $\mathbf{x}$ of images of shape (B, H, W, C) into G groups (where G must divide C) and "normalizes" the mean and standard deviation by computing these statistics over all channels in the group. More precisely, for fixed batch index b and each group index $0 \leq g \leq G-1$, we define

$$\mu_g^b = \frac{1}{\frac{C}{G} \cdot H \cdot W} \sum_{c=1+g\frac{C}{G}}^{(g+1)\frac{C}{G}} \sum_{h=1}^{H} \sum_{w=1}^{W} \mathbf{x}_{(b,h,w,c)}$$

$$\sigma_g^b = \sqrt{\frac{1}{\frac{C}{G} \cdot H \cdot W} \sum_{c=1+g\frac{C}{G}}^{(g+1)\frac{C}{G}} \sum_{h=1}^{H} \sum_{w=1}^{W} (\mathbf{x}_{(b,h,w,c)} - \mu_g)^2 + \epsilon},$$

where $\epsilon = 10^{-6}$ is a small constant to avoid division by 0 later on. The output tensor $\mathbf{y}$, also of shape (B, H, W, C), is for $g = \left\lfloor \frac{c-1}{C/G} \right\rfloor$ defined by

$$\mathbf{y}_{(b,h,w,c)} = \gamma_c \frac{\mathbf{x}_{(b,h,w,c)} - \mu_g}{\sigma_g} + \beta_c,$$

where γ and β are trainable scale and shift parameter vectors of dimension C. The original implementation uses some TensorFlow default hyperparameters such that here, $G = 32$, γ is initialized with 1's, and β is initialized with 0's.

Dense Layers Unsurprisingly, the network employs dense layers, i.e., layers consisting of a matrix W of shape $(C_{\text{in}}, C_{\text{out}})$ together with a bias b of shape (C_{out}). Dense layers are applied to 1-dimensional tensors of shape (C_{in}) as well as to multi-dimensional layers of shape (B, H, W, C_{in}). In both cases, the output is computed as a matrix multiplication plus the bias on the last dimension, producing outputs of shape (C_{out}) and $(B, H, W, C_{\text{out}})$, respectively. As the input dimension C_{in} is given by the input to the layer, we only denote the output dimension C_{out} for dense layers. The bias weights are initialized as 0 and the layer weights are initialized by *variance scaling* with a uniform distribution in fan-avg mode, i.e., with

$$\text{fan-avg} = \frac{C_{\text{in}} + C_{\text{out}}}{2} \qquad \text{limit} = \sqrt{\frac{3 \cdot \text{scale}}{\text{fan-avg}}},$$

weights are drawn uniformly from $[-\text{limit}, \text{limit}]$. The parameter "scale" is chosen either 1 or 10^{-10} for all dense layers. We consider scale $= 1$ as the "default" and only denote deviations.

Convolutional Layers The convolutional layers of the network all have kernel size 3×3. We denote the number of input channels by C_{in} and the number of output channels by C_{out}. As the number of input channels C_{in} is given by the input to the layer, we only denote the number of output channels C_{out} as in $3 \times 3 \times C_{\text{out}}$. Each convolutional layer has a bias b of shape (C_{out}) initialized by 0. Like the dense

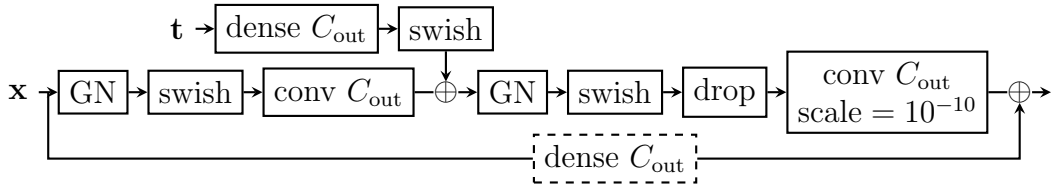
layer weights, the kernel weights are initialized by *variance scaling* with uniform distribution in fan-avg mode, i.e., with

$$\text{fan-avg} = \frac{3 \cdot 3 \cdot C_{\text{in}} + 3 \cdot 3 \cdot C_{\text{out}}}{2} \qquad \text{limit} = \sqrt{\frac{3 \cdot \text{scale}}{\text{fan-avg}}}$$

weights are drawn uniformly from $[-\text{limit}, \text{limit}]$. Like for dense layers, the parameter "scale" is chosen either 1 or 10^{-10} for all convolutional layers. All convolutional layers have either stride $s = 1$ and preserve the dimensions H and W, for which one row / column of 0 padding is added at the top, bottom, left, and right, or stride $s = 2$ and divide the dimensions H and W by 2 (all dimensions are powers of 2) for which one row / column of 0 padding is added at the bottom and right.[2] In the following, we consider stride $s = 1$ and scale $= 1$ as the "default" and only denote deviations.

Downsampling and Upsampling Blocks Downsampling and upsampling blocks obtain an input $\mathbf{x}$ of shape (B, H, W, C) and produce an output of shape $(B, \frac{H}{2}, \frac{W}{2}, C)$ or $(B, 2H, 2W, C)$, respectively. Downsampling blocks are simply convolutional layers with output dimension C and stride $s = 2$ as described above. Upsampling blocks first apply a nearest neighbor upsampling to double the dimensions H and W, i.e., each "pixel" $\mathbf{x}_{(b,h,w)}$ is copied to produce a square of 4 identical pixels, after which a convolution with output dimension C and stride $s = 1$ is applied.

ResNet Blocks The network employs residual neural network (ResNet) blocks [14]. The input to a ResNet block is a tensor of shape (B, H, W, C_{in}) and the output a tensor of shape $(B, H, W, C_{\text{out}})$. Like for the other network blocks, we denote only C_{out} when specifying a ResNet block. The precise implementation is as follows.



Here, GN stands for a group normalization layer and $\mathbf{t}$ is a positional embedding of shape $(B, 512)$. The dashed dense layer on the residual connection is only used if $C_{\text{in}} \neq C_{\text{out}}$, otherwise this connection is an identity connection. Note that the positional embeddings $\mathbf{t}$ are of shape (B, C_{out}) after the dense layer and are *broadcast* for addition to the transformations of $\mathbf{x}$, i.e., they are implicitly converted to a tensor of shape $(B, H, W, C_{\text{out}})$ via $\mathbf{t}_{(b,h,w,c)} = \mathbf{t}_{(b,c)}$.

Attention Blocks The network also employs *scaled dot-product attention* as introduced in [59]. Note that this implementation uses only one attention head. The input is a tensor $\mathbf{x}$ of shape (B, H, W, C) and the output is of the same shape. Three

[2] according to the default TensorFlow implementation of conv2d for padding="SAME"

separate dense layers of output dimension C are used to produce projections of $\mathbf{x}$ for use as *queries*, *keys*, and *values*, i.e.,

$$\mathbf{q}, \mathbf{k}, \mathbf{v} = \text{dense}_q(\mathbf{x}), \text{dense}_k(\mathbf{x}), \text{dense}_v(\mathbf{x})$$

all of shape (B, H, W, C). In the following, we consider $\mathbf{q}_{(b,h,w)}$ as a vector of dimension C, similar for $\mathbf{k}, \mathbf{v}$. We recall the general idea behind attention. For each query $\mathbf{q}_{(b,w,h)}$, we generate a "useful information" by computing a convex combination of all values $(\mathbf{v}_{(b,h,w)})_{h,w}$. For a query $\mathbf{q}_{(b,h,w)}$, the coefficient for value $\mathbf{v}_{(b,h',w')}$ is given by the correlation (i.e., dot-product) of query $\mathbf{q}_{(b,h,w)}$ with key $\mathbf{k}_{(b,h',w')}$. Obviously, these coefficients (also called *scores*) do not lead to a convex combination, so a softmax is applied over all scores produced for $\mathbf{q}_{(b,h,w)}$.

The score matrix is computed as

$$S_{(b,h,w,h',w')} = \frac{\mathbf{q}(b,h,w)^{\mathrm{T}}\mathbf{k}(b,h',w')}{\sqrt{C}}$$

and the coefficient matrix A is computed by taking the *softmax* over the last two dimensions of S, i.e., with $s_{(b,h,w,h',w')} = \exp(S_{(b,h,w,h',w')})$ and $Z = \sum_{h',w'} s_{(b,h,w,h',w')}$, we let $A_{(b,h,w,h',w')} = s_{(b,h,w,h',w')}/Z$. More succinctly, we may write

$$A_{(b,h,w)} = \frac{\exp(S_{(b,h,w)})}{||\exp(S_{(b,h,w)})||_1},$$

where the exponential function is applied pointwise and $||.||_1$ denotes the Manhattan norm. The preliminary attention $\mathbf{y}$ with shape (B, H, W, C) is computed via

$$\mathbf{y}_{(b,h,w)} = \sum_{h',w'} A_{(b,h,w,h',w')} \cdot \mathbf{v}_{(b,h',w')}.$$

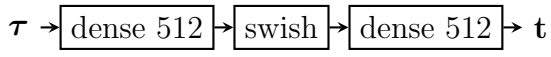
The final output of the layer is the sum of the input $\mathbf{x}$ and a projection of $\mathbf{y}$, so that

$$\text{output} = \mathbf{x} + \text{dense}(\mathbf{y}),$$

where the dense layer is of output dimension C and scale $= 10^{-10}$.

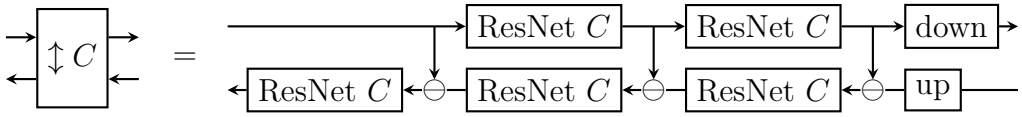
The U-Net A U-Net [45] is a dedicated network architecture for picture-to-picture transformation scenarios, i.e., the input and output usually have the same shape (B, H, W, C), although the number of channels may differ depending on the goal. This is opposed to networks which reduce a batch of images to a tensor of shape (B, C_{out}), for instance to predict the objects present in the images.

Here, the input to the U-Net is a batch $\mathbf{x}$ of noisy images of shape $(B, H, W, 3)$ and the output is a batch of predictions for $\boldsymbol{\varepsilon}_\theta$, also of shape $(B, H, W, 3)$. To sample the input images, a batch of time steps $t \in \{0, 1, \ldots, T-1\}^B$ is sampled uniformly from $\{0, 1, \ldots, T-1\}$ (here, $T = 1000$) and $\mathbf{x}$ is sampled as $\mathbf{x}_b \sim q(\mathbf{x}_{t_b+1}|\mathbf{x}_0 = \tilde{\mathbf{x}}_b)$, where $\tilde{\mathbf{x}}$ is a batch of training images of shape $(B, H, W, 3)$. A batch of positional encodings $\boldsymbol{\tau} = (\tau_{t_1}, \ldots, \tau_{t_B})$ of dimension $2d = 128$ is created as described above and transformed into an embedding $\mathbf{t}$ of shape $(B, 512)$ as follows.



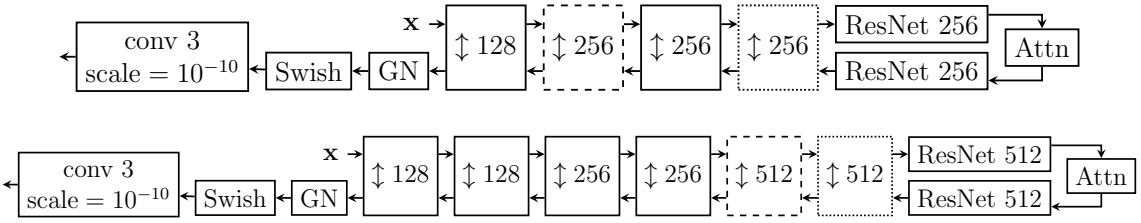
The tensor $\mathbf{t}$ is the positional embedding input to *all* ResNet blocks of the network.

The U-Net consists of a gradual downsampling phase followed by a gradual upsampling phase, where downsampling blocks and upsampling blocks of matching dimensions are also connected by a *skip connection*, i.e., the output of the downsampling block is concatenated to the (linear) input of the upsampling block. Here, the concatenation of two tensors $\mathbf{x}, \mathbf{y}$ of shape (B, H, W, C_1),(B, H, W, C_2) is a tensor $\mathbf{z}$ of shape $(B, H, W, C_1 + C_2)$, where $\mathbf{z}_{(b,h,w)}$ is the concatenation of $\mathbf{x}_{(b,h,w)}$ and $\mathbf{y}_{(b,h,w)}$. Graphically, we denote a concatenation by $\ominus$. We define such a coupled upsampling and downsampling block as follows.



If the input tensor (before downsampling/upsampling) to such a block is of height (and thus also width) $H = 16$, each of the 5 ResNet blocks is immediately followed by an attention block, i.e., before use for concatenation.

There are two specific configurations for the U-Net depending on the size of the input images which is either 32×32 or 256×256. The configuration for 32×32 images uses 4 feature resolutions down to 4×4 while the configuration for 256×256 images uses 6 feature resolutions down to 8×8. The precise configurations are as follows.



Here, the dashed blocks contain attention blocks after each residual block. Moreover, in the deepest scaling block, downsampling and upsampling is omitted.

The U-Net specification is usually (i.e., also for derivative models by other authors) given by a base channel number (here 128) and an array of channel multipliers (here $[1, 2, 2, 2]$ and $[1, 1, 2, 2, 4, 4]$, respectively) which determines the number of downsampling and upsampling blocks as well as their channel counts. The spacial dimension are halved after each block, with the exception of the deepest block, and doubled accordingly during upsampling.

The Optimizer After experimentation with Adam [25] and RMSProp, Adam was chosen as the optimizer with default hyperparameters ($\beta_1 = 0.09$, $\beta_2 = 0.999$) and a learning rate of $2 \cdot 10^{-4}$ for 32×32 images and $2 \cdot 10^{-5}$ for 256×256 images. We note that improved systems by other authors are also optimized using AdamW [31].

EMA EMA (exponential moving average) is applied to model parameters with a decay factor $\beta = 0.9999$. More precisely, a record $\tilde{\theta}_t$ of model parameters is kept and updated according to $\tilde{\theta}_t = \beta \cdot \tilde{\theta}_{t-1} + (1-\beta)\theta_t$, where θ_t are the model parameters of the current optimization step and $\tilde{\theta}_0 = 0$. The initialization bias towards 0 can be corrected via $\hat{\theta}_t = \tilde{\theta}_t/(1-\beta^t)$. The final EMA model parameters are used for inference but not during training. This technique aims to prevent the training data encountered during the last optimization steps before training stopped from dominating the final model weights. Also, EMA reduces the effect of overfitting common towards the end of training.

6.1 Architecture Improvements

In [35] and [8], Dhariwal and Nichol experiment with several improvements to the U-Net architecture which we summarize in the following. Full details can be found in the official GitHub repositories.[3][4] The first two improvements we review are from [35], the remainder from [8].

Cosine Schedule In [35], Nichol and Dhariwal find that the linear noise schedule of [18] is sub-optimal for smaller images and suggest a cosine schedule of the form

$$\bar{\alpha}_t = \frac{f(t)}{f(0)} \qquad\qquad f(t) = \cos\left(\frac{t/T + s}{1+s} \cdot \frac{\pi}{2}\right)^2,$$

where then $\beta_t = 1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}$ and β_t is additionally clipped to be no larger than 0.999 in order to avoid singularities when t approaches T. This choice of schedule leads to a linear drop-off of $\bar{\alpha}_t$ in the middle of the process while near the beginning and end, the changes to $\bar{\alpha}_t$ are small. The schedule takes longer to "destroy" the original information $\mathbf{x}_0$ than the linear schedule. The small offset $s = 0.008$ prevents β_t from being too small near $t = 0$ and was chosen so that $\sqrt{\beta_0}$ is slightly smaller than the pixel bin size $\frac{1}{127.5}$ (see Section 5). The choice of $\cos^2$ was more or less arbitrary and other functions with similar shapes may perform comparably.

Interpolating between β_t and $\tilde{\beta}_t$ In [35], Dhariwal and Nichol suggest to learn $\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t) = \exp(v \log \beta_t + (1-v)\log \tilde{\beta}_t)\mathbf{I}$ as an interpolation between the two choices suggested by Ho et al. [18]. Here, $v = v_\theta(\mathbf{x}_t, t)$ is obtained by doubling the output channels of the U-Net implementing $\boldsymbol{\varepsilon}_\theta(\mathbf{x}, t)$ from 3 (the RGB channels) to 6 (one v for each RGB channel). The coefficient v was not constrained to $[0, 1]$, but was found to be learned to be in $[0, 1]$ during the experiments anyway. As the loss term $L_{\text{simple}}(\theta)$ suggested in [18] does not provide signals to update $\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)$ (see Section 5), the loss is instead modeled as a hybrid loss

$$L_{\text{hybrid}}(\theta) = L_{\text{simple}}(\theta) + \lambda \mathcal{L}_\theta(\mathbf{x}_0),$$

where $\lambda = 0.001$ is chosen small such that $\mathcal{L}_\theta(\mathbf{x}_0)$ does not overwhelm $L_{\text{simple}}(\theta)$ and a stop gradient is applied to the $\mu_\theta(\mathbf{x}_t, t)$ output in $\mathcal{L}_\theta(\mathbf{x}_0)$ such that it guides $\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)$ while $L_{\text{simple}}(\theta)$ remains the main source of updates to $\mu_\theta(\mathbf{x}_t, t)$.

[3] `https://github.com/openai/improved-diffusion`

[4] `https://github.com/openai/guided-diffusion`

BigGAN Up/Downsampling While the original model employs convolutional layers for upsampling and downsampling, the authors report an improved sampling quality when using BigGAN [3] style residual blocks instead, as suggested in [55]. More precisely, each upsampling and downsampling block of the original implementation is replaced by a residual block identical to the one described above but with a scaling layer added before the first convolution (after the swish activation) and before the dense layer on the skip connection, respectively. Upsampling layers consist of a nearest neighbor interpolations to double the dimension and downsampling layers consist of a 2×2 average pooling layers with stride 2 to halve the dimension. Similar to the original implementation, these upsampling and downsampling ResNet blocks are omitted near the U-turn of the U-Net.

Multi-Head and Multi-Resolution Attention The original model employs single-head scalar dot product attention at the 16×16 resolution of the network. Here, the authors suggest to employ multi-head attention at the resolutions 32×32, 16×16, and 8×8. They experiment with using a fixed number of heads for all resolutions (2,4,8) or a fixed number of channels per head for all resolutions (32,64,128) and report better performance with multi-resolution attention and with more heads/fewer channels per head, identifying 64 channels per head as optimal.

Depth over Width The authors investigate whether increasing the depth of the network (number of layers) while reducing its width (number of parameters per layer, i.e., number of channels for convolutional layers) improves performance. More precisely, they increase the number of ResNet blocks per downsampling and upsampling block from 2 to 4 (the BigGAN upsampling/downsampling ResNet block is not counted as a ResNet block here) while reducing the number of channels. While this change improves quality, it also increases training time and is for this reason not pursued further.

Adaptive Group Normalization The authors further report improved sampling quality by using a new mechanism they term *adaptive group normalization (AdaGN)* which modifies how the time step embedding is "injected" into the ResNet blocks. In the original implementation, a time step embedding $\mathbf{t}$ is scaled by a dense layer with swish activation to an embedding y with the correct number of channels for addition to the output h of the first convolutional layer. The original model then computes $\text{GroupNorm}(h + y)$. Similar to adaptive instance norm (AdaIN) [24] and feature-wise linear modulation (FiLM) [38], AdaGN instead computes $(1 + y_s) \odot \text{GroupNorm}(h) + y_b$ as the next step, where $y = [y_s, y_b]$ is an embedding with twice the number of channels of h and $\odot$ denotes the Hadamard (i.e., pointwise) product.

7 Alternatives to Predicting $\boldsymbol{\varepsilon}_t$

Recall the loss term

$$L_t = \mathbb{E}_{\mathbf{x}_t \sim q(\cdot|\mathbf{x}_0)} \left[\frac{1}{2\sigma_t^2} ||\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) - \boldsymbol{\mu}_\theta(\mathbf{x}_t, t)||_2^2 \right] + C.$$

Instead of predicting the noise $\boldsymbol{\varepsilon}_t$ as derived in Section 5, we could either predict $\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0)$ directly or predict $\mathbf{x}_0$ as follows. Since

$$\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) = \sqrt{\alpha_t}\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0$$

we can parameterize

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \sqrt{\alpha_t}\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_\theta(\mathbf{x}_t, t).$$

Then L_t above becomes

$$L_t = \mathbb{E}_{\mathbf{x}_t \sim q(\cdot|\mathbf{x}_0)}\left[\frac{\bar{\alpha}_{t-1}\beta_t^2}{2\sigma_t^2(1-\bar{\alpha}_t)^2}||\mathbf{x}_0 - \mathbf{x}_\theta(\mathbf{x}_t, t)||_2^2\right] + C.$$

As mentioned in Section 5, both approaches were reported to yield inferior results in [18].

There is one more approach we can take by applying *Tweedie's Formula* [11]. Tweedie's Formula states that for a random variable $\boldsymbol{\mu}$ with density $p_{\boldsymbol{\mu}}$ and $X \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\Sigma}$ is a constant diagonal covariance matrix, we have

$$\mathbb{E}_{\boldsymbol{\mu}\sim p_{\boldsymbol{\mu}}}[\boldsymbol{\mu} \mid X = x] = x + \boldsymbol{\Sigma}\nabla_x \log p_X(x).$$

Note that Tweedie's Formula applies in more generality, including for non-diagonal covariance matrices, but this version of the theorem suffices for us. For a derivation, consult the appendix. By applying Tweedie's Formula to $\mathbf{x}_t \sim \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$ with a constant (Dirac distributed) $\mathbf{x}_0$, we obtain $\sqrt{\bar{\alpha}_t}\mathbf{x}_0 = \mathbf{x}_t + (1-\bar{\alpha}_t)\nabla_{\mathbf{x}_t}\log q(\mathbf{x}_t|\mathbf{x}_0)$. By writing $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}_t$ with $\boldsymbol{\varepsilon}_t \sim \mathcal{N}(0, \mathbf{I})$, this formula is easy to verify by hand without employing Tweedie's Formula and this also reveals that

$$\begin{aligned}
\nabla_{\mathbf{x}_t}\log q(\mathbf{x}_t|\mathbf{x}_0) &= \nabla_{\mathbf{x}_t}\left(\log\frac{1}{\sqrt{(2\pi\det(1-\bar{\alpha}_t)\mathbf{I})^k}} - \frac{1}{2}\sum_i \frac{(\mathbf{x}_{t,i} - \sqrt{\bar{\alpha}_t}\mathbf{x}_{0,i})^2}{1-\bar{\alpha}_t}\right) \\
&= -\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{1-\bar{\alpha}_t} \\
&= -\frac{\boldsymbol{\varepsilon}_t}{\sqrt{1-\bar{\alpha}_t}}.
\end{aligned}$$

We can now compute

$$\begin{aligned}
\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) &= \sqrt{\alpha_t}\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t + (1-\bar{\alpha}_t)\nabla_{\mathbf{x}_t}\log q(\mathbf{x}_t|\mathbf{x}_0)) \\
&= (\alpha_t - \bar{\alpha}_t + \beta_t)\frac{\mathbf{x}_t}{(1-\bar{\alpha}_t)\sqrt{\alpha_t}} + \frac{\beta_t}{\sqrt{\alpha_t}}\nabla_{\mathbf{x}_t}\log q(\mathbf{x}_t|\mathbf{x}_0) \\
&= \frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t + \frac{\beta_t}{\sqrt{\alpha_t}}\nabla_{\mathbf{x}_t}\log q(\mathbf{x}_t|\mathbf{x}_0).
\end{aligned}$$

Accordingly, we may choose to parameterize

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t + \frac{\beta_t}{\sqrt{\alpha_t}}s_\theta(\mathbf{x}_t, t)$$

in which case L_t becomes

$$L_t = \mathbb{E}_{\mathbf{x}_t \sim q(\cdot|\mathbf{x}_0)} \left[\frac{\beta_t^2}{2\sigma_t^2 \alpha_t} ||\nabla_{\mathbf{x}_t} \log q(\mathbf{x}_t|\mathbf{x}_0) - s_\theta(\mathbf{x}_t, t)||_2^2 \right] + C.$$

Clearly, $s_\theta(\mathbf{x}_t, t)$ is encouraged to learn a rescaling of the objective of $\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t)$. The log derivative $\nabla_{\mathbf{x}_t} \log q(\mathbf{x}_t|\mathbf{x}_0)$ is also called the *score* function of $\mathbf{x}_t$. This formulation of the loss function shows a close connection to score based generative models.

Connection to SDEs and Score Based Models Assume that instead of the discrete time steps $t = 0, 1, \ldots, T$ we consider $[0, T]$ as a continuous interval of time steps and model

$$\begin{aligned} q(\mathbf{x}_{t+\Delta t} - \mathbf{x}_t|\mathbf{x}_t) &= \mathcal{N}(f(\mathbf{x}_t, t)\Delta t, g^2(t)\Delta t^2) \text{ or} \\ \mathbf{x}_{t+\Delta t} - \mathbf{x}_t &= f(\mathbf{x}_t, t)\Delta t + g(t)\Delta t \cdot \boldsymbol{\varepsilon} \text{ with } \boldsymbol{\varepsilon} \sim \mathcal{N}(0, \mathbf{I}), \end{aligned}$$

then we can consider f and g as the drift and diffusion functions of the stochastic differential equation

$$d\mathbf{x}_t = f(\mathbf{x}_t, t)dt + g(t)dW_t,$$

where W_t is a Wiener process. For example, the process we defined for DDPMs becomes

$$d\mathbf{x}_t = -\frac{1}{2}\beta_t \mathbf{x}_t dt + \sqrt{\beta_t} dW_t,$$

see Appendix B of [55] for a derivation. Due to a result for SDEs [1], such a forward diffusion process can be reversed in closed form with the SDE

$$d\mathbf{x}_t = \Big[f(\mathbf{x}_t, t) - g(t)^2 \nabla_{\mathbf{x}} \log p(\mathbf{x}_t) \Big] dt + g(t) dW_t,$$

where p is the density of the process $(\mathbf{x}_t)_{t\in[0,T]}$ solving the forward SDE with $\mathbf{x}_0 \sim p^*$. This allows sampling from p^* by applying the reverse process to a random sample $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$. Score based models aim to approximate the score $s(\mathbf{x}_t, t) = \nabla_{\mathbf{x}} \log p(\mathbf{x}_t)$ and then approximate the reverse process via discrete sampling. For an introduction to score based models and their connection to DDPMs, see the articles by Song [54] and by Das [5, 6].

8 Speedups – Strided Sampling, DDIM, and LDM

Sampling from a diffusion model as described in [18] is very time consuming as $T = 1000$ denoising steps have to be computed. Song et al. report in [53] that "For example, it takes around 20 hours to sample 50k images of size 32×32 from a DDPM, but less than a minute to do so from a GAN on a Nvidia 2080 Ti GPU." In the following, we review some approaches to increase sampling speed.

Strided Sampling The first approach, suggested in [35], uses a strided sampling procedure. Note that this approach also employs the interpolated parameterization of $\mathbf{\Sigma}_\theta$ and the cosine variance schedule as explained in Section 6.1. In their experiments, the authors train a diffusion model with $T = 4000$ steps and experiment with sampling only $n = 25, 50, 100, 200, 400, 1000, 4000$ steps as follows. A sampling schedule $(S_t)_{t=1}^n$ is selected by spacing n points evenly in $[1, T]$ (including 1 and T) and rounding to the nearest integer. Then sampling is conducted as if the model had been trained using this sampling schedule. More precisely, during training we assume that

$$\mathbf{x}_t \sim \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}) \qquad \text{and} \qquad \mathbf{x}_t \sim \mathcal{N}(\sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}).$$

Preserving the first assumption, β_{S_t} and $\tilde{\beta}_{S_t}$ are computed as

$$\beta_{S_t} = 1 - \frac{\bar{\alpha}_{S_t}}{\bar{\alpha}_{S_{t-1}}} \qquad \text{and} \qquad \tilde{\beta}_{S_t} = \beta_{S_t}\frac{1-\bar{\alpha}_{S_{t-1}}}{1-\bar{\alpha}_{S_t}}$$

such that

$$\mathbf{x}_{S_t} \sim \mathcal{N}(\sqrt{\bar{\alpha}_{S_t}}\mathbf{x}_0, (1-\bar{\alpha}_{S_t})\mathbf{I}) \qquad \text{and} \qquad \mathbf{x}_{S_t} \sim \mathcal{N}(\sqrt{1-\beta_{S_t}}\mathbf{x}_{S_{t-1}}, \beta_{S_t}\mathbf{I}).$$

The reverse process then samples $\mathbf{x}_{S_{t-1}} \mid \mathbf{x}_{S_t} \sim \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_{S_t}, S_t), \mathbf{\Sigma}_\theta(\mathbf{x}_{S_t}, S_t))$. The authors found that using the hybrid loss function L_{hybrid} and 100 sampling steps already produces near optimal Fréchet Inception Distance (FID) [15]. In comparison to DDIM (see next paragraph), the authors found that DDIM produces better samples with fewer than 50 sampling steps but worse samples when using 50 or more steps.

DDIM In [53], Song et al. introduce *denoising diffusion implicit models*. The authors observe that the training of DDPMs only requires the marginals of $\mathbf{x}_t$ given $\mathbf{x}_0$ to satisfy

$$\mathbf{x}_t \mid \mathbf{x}_0 \sim \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}).$$

Thus, any forward process q_σ with these marginal distributions, including a non-Markovian process, is a valid candidate to replace the Markovian process used in [18]. With this motivation, the authors define a process q_σ whose reverse marginal density $q_\sigma(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ uses a fixed variance schedule $\sigma = (\sigma_t^2)_{t=1}^T$ as follows.

$$\begin{aligned}
\mathbf{x}_{t-1} &= \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}}\boldsymbol{\varepsilon}_{t-1} \\
&= \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\boldsymbol{\varepsilon}_t + \sigma_t\boldsymbol{\varepsilon} \\
&= \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1-\bar{\alpha}_t}} + \sigma_t\boldsymbol{\varepsilon}
\end{aligned}$$

such that

$$q_\sigma(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \underbrace{\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1-\bar{\alpha}_t}}}_{=\boldsymbol{\mu}_{\sigma,t}(\mathbf{x}_t,\mathbf{x}_0)}, \sigma_t^2\mathbf{I}).$$

The Markovian process q satisfies $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t)$ implying that q_σ is Markovian if and only if $\sigma_t^2 = \tilde{\beta}_t$.

The joint density $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$ and the density $q_\sigma(\mathbf{x}_T|\mathbf{x}_0)$ are chosen as

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0) \quad \text{and} \quad q_\sigma(\mathbf{x}_T|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_T; \sqrt{\bar{\alpha}_T}\mathbf{x}_0, (1-\bar{\alpha}_T)\mathbf{I})$$

such that the derivation of the loss function in Section 4 is valid without any changes: simply replace $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ by $q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0)$. The reverse process is modeled as

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_\theta(\mathbf{x}_t, t), \sigma_t^2\mathbf{I}).$$

The KL divergence in the loss term L_t then becomes

$$\begin{aligned} D_{\mathrm{KL}}(q_\sigma(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) &= D_{\mathrm{KL}}(\mathcal{N}(\boldsymbol{\mu}_{\sigma,t}(\mathbf{x}_t, \mathbf{x}_0), \sigma_t^2\mathbf{I}) \parallel \mathcal{N}(\boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \sigma_t^2\mathbf{I})) \\ &= \frac{1}{2\sigma_t^2}||\boldsymbol{\mu}_{\sigma,t}(\mathbf{x}_t, \mathbf{x}_0) - \boldsymbol{\mu}_\theta(\mathbf{x}_t, t)||_2^2. \end{aligned}$$

Again, during training we may assume $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}_t$ such that

$$\boldsymbol{\mu}_{\sigma,t}(\mathbf{x}_t, \mathbf{x}_0) = \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\boldsymbol{\varepsilon}_t.$$

which motivates modeling

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t)$$

and L_t becomes

$$L_t = \mathbb{E}_{\mathbf{x}_0 \sim p^*, \boldsymbol{\varepsilon}_t \sim \mathcal{N}(0,\mathbf{I})}\left[\frac{1-\bar{\alpha}_{t-1}-\sigma_t^2}{2\sigma_t^2}||\boldsymbol{\varepsilon}_t - \boldsymbol{\varepsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}_t, t)||_2^2\right].$$

Note that σ_t here is not the same as in Section 4, where it was chosen as either β_t or $\tilde{\beta}_t$. This loss term is a rescaling of the loss term derived in Section 4 (ignoring the constant C) which yields the important observation that the models trained for DDPM and DDIM optimize the same objective. Thus, it is possible to sample from a DDPM trained model using the DDIM sampling algorithm

$$\mathbf{x}_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\underbrace{\left(\frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t)}{\sqrt{\bar{\alpha}_t}}\right)}_{\text{predicted } \mathbf{x}_0} + \underbrace{\sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t)}_{\text{direction pointing to } \mathbf{x}_t} + \underbrace{\sigma_t\boldsymbol{\varepsilon}}_{\text{random noise}},$$

with the definition $\alpha_0 = 1$. In particular, the same model can be used to sample using different choices of σ_t. The special case $\sigma_t = 0$, where the sampling process becomes deterministic, is what the authors define as *denoising diffusion implicit models*.

To increase sampling speed, strided sampling as described above is applied using the DDIM sampling algorithm. In their experiments, the authors choose the variance schedule $\sigma_{S_t}^2 = \eta\tilde{\beta}_{S_t}$ for sampling schedules of the form $(S_t)_{t=1}^n \subseteq [1, T]$ with a hyperparameter $\eta > 0$. For $\eta = 1$, this corresponds to (strided) DDPM sampling

and for $\eta = 0$, to (strided) DDIM sampling. The authors report a higher sample quality with DDIM sampling when n is small. In particular, DDIM produces samples comparable 1000 step DDPM sampling in only 20 to 100 steps. Moreover, DDIM displays "sample consistency" due to its deterministic nature in the sense that the initial latent $\mathbf{x}_T$ determines the high level features of the output regardless of the number of sampling steps and varying the number of sampling steps only affects minor details. This also indicates that DDIM is suited for latent space interpolation as sampling from an interpolated initial latent is more likely to produce an output whose high level features are also interpolated while DDPM may simply "sample out" high level features during the sampling process.

LDM In DDPMs, the latents $\mathbf{x}_1, \dots, \mathbf{x}_T$ have the same dimensionality as the data $\mathbf{x}_0$. The authors of [44] suggest to instead perform the denoising process in a lower dimensional latent space to generate high level features and then use a dedicated decoder which generates low level features (detail) and maps the denoised latent into the image data space. This proves to significantly lower computational costs both for training and for sampling. The denoising process is the same as in DDPMs apart from minor differences (e.g., the initialization of convolutional layer weights, no bias in the linear dense layers of the attention blocks). Therefore, we describe in the following the encoder and decoder architecture used to move between the data space and the latent space. Full details can be found in the official GitHub repository.[5]

The encoder-decoder pair is constructed as a variational autoencoder (VAE) [27, 28]. The authors experiment both with "vanilla" VAEs as well as with vector quantized VAEs (VQ-VAEs) [42, 58]. For vanilla VAEs, the encoder $\mathcal{E}$ takes a batch of image inputs $\mathbf{x}$ of shape (B, H, W, C) and outputs means $\mathcal{E}_\mu(\mathbf{x})$ and variances $\mathcal{E}_{\sigma^2}(\mathbf{x})$ both of shape (B, h, w, c), where (h, w, c) is the latent space dimensionality. The latent $\mathbf{z}$ is generated by sampling each entry from a normal distribution using its predicted mean and variance. For VQ-VAEs, $\mathcal{E}$ outputs a single tensor of shape (B, h, w, c) serving as the latent $\mathbf{z}$. The encoder is constructed exactly like the downsampling part of the U-Net in Section 6 including the ResNet-Attn-ResNet sequence after the downsampling. It is then followed by a final GroupNorm-Swish-Conv sequence where the last Conv layer preserves spatial dimensions (which become the latent space spatial dimensions $h \times w$) by using a 3×3 kernel with a padding of size 1 on all sides and fixes the number of channels to either $2c$ or c, depending on whether it implements a VAE or a VQ-VAE. In the first case, the first half of channels is treated as the predicted means and the second half as the logarithms of the predicted variances.

The decoder $\mathcal{D}$ first quantizes the input if it is implemented as a VQ-VAE. A 3×3 convolution restores the shape the encoder produced before forcing the channel number c or $2c$. After this, the decoder follows the U-Net layout of Section 6, repeating the ResNet-Attn-ResNet sequence into the upsampling blocks but without any skip connections from the encoder. The information loss from the original data $\mathbf{x}$ to its reconstruction $\mathbf{x}' = \mathcal{D}(\mathcal{E}(\mathbf{x}))$ is based on [12] and measured as a combination of

[5] `https://github.com/CompVis/latent-diffusion`

1. a *reconstruction loss* $L_{\text{rec}}(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||_1$,
2. a *perceptual loss* (LPIPS[6]) [64] which computes the distance between two images as the L_2 distance of their activations in the deepest layer of a feature extraction network like AlexNet or VGG (here: AlexNet?),
3. a *regularization loss*, either as the VQ-VAE commitment loss or as the KL-divergence between a standard normal distribution and the normal distribution given by the predicted means and variances, and
4. a patch-based [22] adverserial loss [10, 12, 62] of the form $-D_\psi(\mathcal{D}(\mathcal{E}(\mathbf{x}))$ of a discriminator D_ψ which is trained to tell "real" images $\mathbf{x}$ from "fake" (reconstructed) images $\mathbf{x}'$.

The discriminator D_ψ is trained jointly with the autoencoder. It is implemented as a stack of conv-norm-LeakyReLU blocks. The convolutional layers downscale the image and increase channels by using 4×4 convolutions with a stride of 2 and the norm is always either a batch norm [21] or an activation norm [26]. A final convolutional layer projects the channel dimension to 1 and each entry in the resulting map is treated as the logit for the "realness" of an associated patch. The predictions are averaged and larger predictions are treated as higher perceived realness. The discriminator is trained using a standard GAN loss $\frac{1}{2}(\text{softplus}(-D_\psi(\mathbf{x}))+\text{softplus}(\mathcal{D}(\mathcal{E}(\mathbf{x}'))))$ where $\text{softplus}(x) = \log(1 + e^x)$.

9 Conditioned Generation

So far, we have only considered modeling the data distribution $p^*(\mathbf{x})$. However, it is desirable to be able to model conditional data distributions $p^*(\mathbf{x}|y)$ conditioned on some label y such that the model output can be controlled via y. Such a label could be a text encoding for text-to-image synthesis as (Imagen [47], DALLE-2 [41], Stable Diffusion [44]), a partially drawn image to be inpainted [44], or a low resolution image to perform super-resolution on [19, 44]. In the following, we review approaches to enhance the models discussed so far with conditioning mechanisms.

AdaGN Every ResNet block in the U-Net for $\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t)$ obtains a time step embedding $\mathbf{t}$. Dhariwal and Nichol suggest to convert the label y for $\mathbf{x}_t$ into a label embedding $\mathbf{y}$ of the same dimension as $\mathbf{t}$ using a standard embedding layer. Each ResNet block then obtains $\mathbf{t} + \mathbf{y}$ instead of only $\mathbf{t}$ as a combined temporal and class embedding.

Note that the class label information is not reflected in the loss function such that the diffusion process is not guided to produce an output of the provided class. Rather, it provides additional information to the network which it can leverage but also learn to ignore. This approach is motivated by the fact that Lucic et al. [32] found that even creating synthetic labels by applying a mini-batch k-means clustering [50] on the representation of a feature extractor trained to recognize the rotation degree of a randomly rotated image [30] improved the FID scores of GANs. However, the conditioning mechanisms discussed in the following paragraphs suggest that models learn to leverage "meaningful" semantic information.

[6] `https://github.com/richzhang/PerceptualSimilarity`

Classifier Guidance In [8], Dhariwal and Nichol train a classifier $p_\phi(y|\mathbf{x}_t)$ on noisy images $\mathbf{x}_t$ and use the gradients $\nabla_{\mathbf{x}_t} \log p_\phi(y|\mathbf{x}_t)$ to guide the diffusion process. For a clean derivation, we first make some observations. Recall that we assume

$$q(\mathbf{x}_T|\mathbf{x}_0) \approx \mathcal{N}(\mathbf{x}_T; 0, \mathbf{I}) = p_\theta(\mathbf{x}_T) \qquad \text{and} \qquad q(\mathbf{x}_t|\mathbf{x}_{t+1}, \mathbf{x}_0) \approx p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1}).$$

By an induction, we can thus show as follows that in general $q(\mathbf{x}_t|\mathbf{x}_0) \approx p_\theta(\mathbf{x}_t)$.

$$\begin{aligned}
q(\mathbf{x}_t|\mathbf{x}_0) &= \int_{\mathbf{x}_{t+1}} q(\mathbf{x}_t, \mathbf{x}_{t+1}|\mathbf{x}_0) d\mathbf{x}_{t+1} \\
&= \int_{\mathbf{x}_{t+1}} \frac{q(\mathbf{x}_t, \mathbf{x}_{t+1}, \mathbf{x}_0)}{q(\mathbf{x}_0)} \frac{q(\mathbf{x}_{t+1}, \mathbf{x}_0)}{q(\mathbf{x}_{t+1}, \mathbf{x}_0)} d\mathbf{x}_{t+1} \\
&= \int_{\mathbf{x}_{t+1}} q(\mathbf{x}_t|\mathbf{x}_{t+1}, \mathbf{x}_0) q(\mathbf{x}_{t+1}|\mathbf{x}_0) d\mathbf{x}_{t+1} \\
&\approx \int_{\mathbf{x}_{t+1}} p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1}) p_\theta(\mathbf{x}_{t+1}) d\mathbf{x}_{t+1} \qquad \text{(induction hypothesis)} \\
&= \int_{\mathbf{x}_{t+1}} p_\theta(\mathbf{x}_t, \mathbf{x}_{t+1}) d\mathbf{x}_{t+1} \\
&= p_\theta(\mathbf{x}_t)
\end{aligned}$$

This implies

$$\nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t) \approx \nabla_{\mathbf{x}_t} \log q(\mathbf{x}_t|\mathbf{x}_0) = -\frac{\boldsymbol{\varepsilon}_t}{\sqrt{1-\bar{\alpha}_t}} \approx -\frac{\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t)}{\sqrt{1-\bar{\alpha}_t}}.$$

Now assume we want to guide the denoising process p_θ by a class label y in the form of $p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1}, y)$. Then instead of predicting (a rescaling of) $\nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t)$, we want to predict a conditioned score $\nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t|y)$. We can compute [33]

$$\begin{aligned}
\nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t|y) &= \nabla_{\mathbf{x}_t} \log \left(\frac{p_\theta(\mathbf{x}_t) p_\theta(y|\mathbf{x}_t)}{p_\theta(y)} \right) \\
&= \nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t) + \nabla_{\mathbf{x}_t} \log p_\theta(y|\mathbf{x}_t) \\
&\approx -\frac{\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t)}{\sqrt{1-\bar{\alpha}_t}} + \nabla_{\mathbf{x}_t} \log p_\phi(y|\mathbf{x}_t) \\
&= -\frac{1}{\sqrt{1-\bar{\alpha}_t}} (\underbrace{\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t) - \sqrt{1-\bar{\alpha}_t} \nabla_{\mathbf{x}_t} \log p_\phi(y|\mathbf{x}_t)}_{\hat{\boldsymbol{\varepsilon}}_{\theta,\phi}(\mathbf{x}_t, t)}),
\end{aligned}$$

where $p_\phi(y|\mathbf{x}_t)$ is a class label predictor for noisy images. The authors found that the guidance had to be strengthened for optimal results and modeled $\hat{\boldsymbol{\varepsilon}}_{\theta,\phi}$ with a hyperparameter γ as

$$\hat{\boldsymbol{\varepsilon}}_{\theta,\phi}(\mathbf{x}_t, t) = \boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t) - \sqrt{1-\bar{\alpha}_t} \gamma \nabla_{\mathbf{x}_t} \log p_\phi(y|\mathbf{x}_t).$$

The choices for γ range from 0.5 to 11 (usually $\gamma > 1$) and is adapted to the training set of the model. The actual implementation of $\log p_\phi(y|\mathbf{x}_t)$ employs the U-Net architecture up to the U-turn. The gradients are computed automatically using PyTorch. For extended details, consult the official GitHub repository.[7]

[7] `https://github.com/openai/guided-diffusion`, `guided_diffusion/unet.py/EncoderUNetModel`, `guided_diffusion/script_util.py/create_classifier`, `scripts/classifier_sample.py/cond_fn`

Classifier-Free Guidance In [17], Ho and Salimans develop a guidance method which avoids using a dedicated classifier, as training such a classifier in itself is costly; employing existing classifiers is not optimal as these are not trained on noisy images. Instead, the authors train both an unconditional denoising diffusion model $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ parameterized through an estimator $\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t)$ and a conditional denoising diffusion model $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t, y)$ parameterized through an estimator $\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t, y)$. Effectively, both models can be trained in parallel, only training $\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t, y)$, by creating a synthetic "void" label $\emptyset$ and during training randomly choose either the correct label y for $\mathbf{x}_t$ to train the conditional model or choose $y = \emptyset$ to train the unconditional model.

A motivation for the approach of Ho and Salimans can be derived as follows. In the section on classifier guidance, we saw that

$$\nabla_{\mathbf{x}_t} \log p_\theta(y|\mathbf{x}_t) = \nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t|y) - \nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t).$$

Thus, if we use the classifier $p_\theta(y|\mathbf{x}_t) = p_\theta(\mathbf{x}_t|y)p_\theta(\mathbf{x}_t)^{-1}p_\theta(y)$ for classifier guidance, the weighted score function becomes

$$\begin{aligned}\nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t|y) &= \nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t) + \gamma \nabla_{\mathbf{x}_t} \log p_\theta(y|\mathbf{x}_t) \\ &= \nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t) + \gamma(\nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t|y) - \nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t)) \\ &= \gamma \nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t|y) + (1-\gamma)\nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t).\end{aligned}$$

With the parameterizations

$$\nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t) \approx -\frac{\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t)}{\sqrt{1-\bar{\alpha}_t}} \qquad \nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t|y) \approx -\frac{\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t, y)}{\sqrt{1-\bar{\alpha}_t}} \qquad \gamma = 1 + w$$

the authors suggest a sampling via

$$\hat{\boldsymbol{\varepsilon}}(\mathbf{x}_t, t, y) = (1+w)\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t, y) - w\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t),$$

where w controls the influence of the conditional diffusion process. The authors report that the choice of w provides a trade-off between FID score [15] and Inception Score (IS) [48]. Sweeping the value of w from 0.0 to 4.0 reduces FID while increasing IS.

There is no official GitHub implementation,[8] but the paper suggests that the only architectural change is to provide an embedding $\mathbf{y}$ of y to $\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t, \mathbf{y})$ for addition to $\mathbf{t}$ which can be implemented using a standard embedding layer.

GLIDE In [36], Nichol et al. develop a diffusion denoising model they call GLIDE (Guided Language to Image Diffusion for Generation and Editing) which is able to simultaneously condition on both images and text. They experiment both with classifier-free guidance as well as with classifier guidance using CLIP [39] to guide the denoising process, which turns out to provide worse results than classifier-free guidance. The base architecture is the ADM model from [8] discussed in Section 6.1. For full details, consult the official GitHub repository.[9]

[8] an unofficial implementation can be found under `https://github.com/coderpiaobozhe/classifier-free-diffusion-guidance-Pytorch`

[9] `https://github.com/openai/glide-text2im`

The models are conditioned only on text during training. For this, the text is converted to a sequence of tokens and fed to a transformer [59] to produce a sequence of embeddings which are used in two ways. First, the embedding of the last token in the sequence is passed in place of a class embedding to the ResNet blocks as described in the paragraph on AdaGN, i.e., it is added to the temporal embedding vector. Second, the embedding sequence is separately projected to the dimensionality of each attention block in the U-Net and concatenated to the key and value sequences of the (self) attention layers.

Conditioning on images is achieved by fine-tuning the trained text-conditioned models. For this, the U-Net is modified to accommodate 4 input channels in addition to the 3 input channels containing the noisy input image: 3 new RGB channels for the image to be conditioned on and a mask channel to indicate the region masked for inpainting (the RGB channels of masked regions are zeroed out, resulting in black regions, but not all black regions in an image should be inpainted). The weights introduced to attend to the new channels are initialized to 0 before fine-tuning. The authors experiment with inpainting, super resolution, and super resolution inpainting (both tasks simultaneously).

CLIP [39] (short for Contrastive Language-Image Pre-training) is an approach for learning joint representations between text and images. A CLIP model consists of an image encoder $f(\mathbf{x})$ and a caption encoder $g(c)$, both producing real vectors. During training, batches $(\mathbf{x}_1, c_1), \ldots, (\mathbf{x}_n, c_n)$ of image/caption pairs from a training set are randomly sampled and a contrastive cross-entropy loss aims to maximizes the dot products $f(\mathbf{x}_i)^{\mathrm{T}} g(c_i)$ and minimize the dot products $f(\mathbf{x}_i)^{\mathrm{T}} g(c_j)$ with $i \neq j$. Full details can be found in the paper and the GitHub repository.[10] As pre-trained CLIP models are not trained on noisy images, the authors train a dedicated CLIP model where the image encoder $f(\mathbf{x}, t)$ takes time step information into account. CLIP guidance is realized by modeling $\log p_\phi(c|\mathbf{x}_t) \approx f(\mathbf{x}_t, t)^{\mathrm{T}} g(c)$.

Cascaded Diffusion Models In [19], Ho et al. propose an iterative application of diffusion denoising models for super resolution. At the lowest resolution, a diffusion model $p_\theta(\mathbf{z}_0|\mathbf{z})$ upsamples a low resolution image $\mathbf{z}$ to an image $\mathbf{z}_0$. This image is in turn upsampled by a higher resolution diffusion model $p_\theta(\mathbf{x}_0|\mathbf{z}_0)$, a process which can be iterated. The models are further conditioned on class information c. The conditioning is achieved as in the previous models, i.e., the conditioning image is concatenated to the input of the U-Net and the class information is added to the temporal embedding via a class embedding. The model architecture employs the first two improvements discussed in Section 6.1 from [35].

The authors found that training the *cascading pipeline* of super resolution diffusion models benefits from data augmentation being applied to the lower resolution conditioning image. Specifically, the authors notice a positive effect when randomly (e.g., with 50% probability) adding Gaussian noise to the conditioning image $\mathbf{z}$ for low resolution upsamplers and randomly adding Gaussian blur to the conditioning image $\mathbf{z}_0$ for higher resolution upsamplers. During inference, no noise or blur is added. The authors term this form of data augmentation *noise conditioning augmentation*.

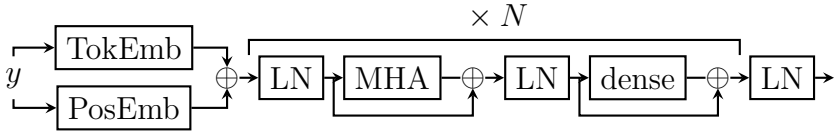
[10] `https://github.com/openai/CLIP`

Moreover, the authors experiment with a *truncated* training process, where a low resolution model $p_\theta(\mathbf{z}_t|\mathbf{z})$ is only trained to upsample from timesteps T to $s > 0$ while the higher resolution upsampler $p_\theta(\mathbf{x}_0|\mathbf{z}_s)$ is trained to super resolve from noisy inputs $\mathbf{z}_s$. In a denoising step from $\mathbf{x}_t$ to $\mathbf{x}_{t-1}$, the high resolution upsampler obtains $\mathbf{x}_t, \mathbf{z}_s$, and embeddings for t and s. During sampling, the authors suggest not to truncate the sampling process but run the full process to sample $\mathbf{z}_0$ as $p_\theta(\mathbf{z}_0|\mathbf{z})$ from $\mathbf{z}$, then sample $\mathbf{z}'_s$ from $q(\mathbf{z}'_s|\mathbf{z}_0)$, and continue the upsampling process with $\mathbf{z}'_s$ instead of $\mathbf{z}_s$ as the conditioning image for the higher resolution upsampler.

Cross Attention (LDM/Stable Diffusion) In [44], which also introduces latent diffusion models, Rombach et al. experiment with two modifications of the conditioning methods used in GLIDE. As in GLIDE, the base architecture is the ADM model from [8] discussed in Section 6.1. Classifier-free guidance is chosen to guide the denoising process.

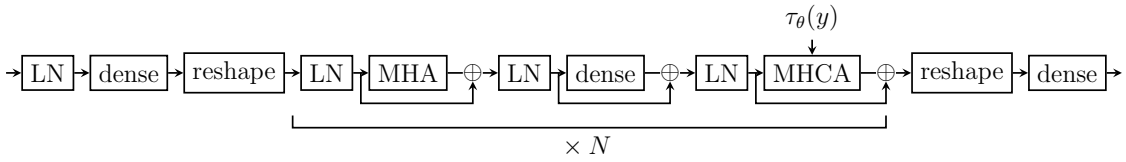
First, GLIDE fine-tuned a pre-trained model for image-to-image synthesis tasks. In contrast, Rombach et al. concatenate a downsampled version of the conditioning information to the input $\mathbf{x}_t$ of $\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t)$ during training. The authors employ this approach to train models for semantic synthesis (by providing a semantic map of $\mathbf{x}_0$), super-resolution (by providing a low-resolution version of $\mathbf{x}_0$), and inpainting.

The second method enhances the attention mechanism used in GLIDE to attend to text conditioning information. Here, the authors replace the self attention blocks of the U-Net implementing $\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t)$ with cross attention blocks. First, a domain specific encoder τ_θ projects the conditioning information y to an intermediate representation $\tau_\theta(y) \in \mathbb{R}^{M \times d_\tau}$. For class conditioning, τ_θ can be implemented as a simple embedding layer. For text-to-image and layout-to-image synthesis, τ_θ is modeled as a transformer as follows.



Here, y is a token sequence (e.g., of words or of bounding box location and class information), TokEmb and PosEmb are trainable embedding layers for the token embedding and the positional embedding, respectively, LN denotes layer norms [2], MHA is the usual multi-head self attention, and N denotes the depth of the transformer, i.e., the number of self attention blocks. In particular, BERT [7] serves as an implementation for τ_θ in the text-to-image scenario.

The representation $\tau_\theta(y)$ is used as the input to the keys and values in a multi-head attention mechanism. Recall that in the self attention mechanism explained in Section 6, the queries $\mathbf{q}$, keys $\mathbf{k}$, and values $\mathbf{v}$ are all computed from the current hidden state $\mathbf{x}$. For cross attention, the queries are obtained by applying a dense layer dense_q to $\mathbf{x}$ and the keys and values are obtained by applying dense layers dense_k and dense_v to $\tau_\theta(y)$. To employ the usual multi-head attention formulas for linear sequences, the state $\mathbf{x}$ of shape (B, H, W, C) is first reshaped to $(B, H \cdot W, C)$ and the output reshaped to (B, H, W, C) again. The explanation in Section 6 avoided reshaping, but the mechanism is the same. The attention blocks of the U-Net are replaced by a transformer block with the following architecture.



Here, "reshape" denotes the two reshapings described above, MHA is a multi-head self attention layer, MHCA a multi-head cross attention layer, and $N = 1$ is chosen for most models, but $N = 3$ has also been used to train a model.

unCLIP (DALLE-2) In [41], Ramesh et al. further develop the GLIDE model by enhancing the text conditioning mechanism with CLIP embeddings. CLIP guidance is not explored further and dropped in favor of classifier-free guidance. In particular, the authors employ a standard CLIP model trained on "normal" (not noisy) images producing image embeddings $\mathbf{z}^i$ from images $\mathbf{x}$ and text embeddings $\mathbf{z}^t$ from captions c. The proposed text-to-image model consists of a decoder $p_\theta(\mathbf{x}|\mathbf{z}^i, c)$ sampling images from CLIP image embeddings (and optionally the caption) and a prior $p_\theta(\mathbf{z}^i|c)$ sampling CLIP image embeddings from captions. As the decoder is trained to invert the CLIP image embedding, the model is named *unCLIP*. An unofficial implementation can be found on GitHub.[11]

The decoder $p_\theta(\mathbf{x}|\mathbf{z}^i, c)$ is a denoising diffusion model modifying the ADM architecture from [8] by projecting and adding the CLIP image embedding $\mathbf{z}^i$ to the timestep embedding and by projecting $\mathbf{z}^i$ into four extra tokens of context which are, together with transformer generated embeddings of the caption c, concatenated to the key and value sequences of the networks attention layers. This approach is very similar to how GLIDE provides text conditioning information to the network, in particular retaining the provision of text embeddings to the attention layers.

For the prior $p_\theta(\mathbf{z}^i|c)$, two approaches are investigated. Both approaches employ classifier-free guidance and during training drop the text conditioning information 10% of the time.

The first approach trains a diffusion denoising model which predicts $\mathbf{z}^i_{t-1} = \mathbf{z}^i_{t-1}(c, \mathbf{z}^t, t, \mathbf{z}^i_t, \emptyset)$. More precisely, the first approach employs a decoder-only transformer with a causal attention mask on a sequence consisting of, in order: the encoded text, the CLIP text embedding, an embedding for the diffusion timestep, the noised CLIP image embedding, and a final embedding (here denoted by $\emptyset$ whose output from the transformer is used to predict the unnoised CLIP image embedding $\mathbf{z}^i$. Instead of the $\boldsymbol{\varepsilon}$-prediction usually employed to optimize diffusion models, the authors find that directly predicting $\mathbf{z}_i$ works better and use the loss term

$$L_{\text{prior}} = \mathbb{E}_{t\sim[0,T],\mathbf{z}^i_t\sim q(\cdot|\mathbf{z}^i)}\left[||\mathbf{z}^i_{t-1}(c, \mathbf{z}^t, t, \mathbf{z}^i_t, \emptyset) - \mathbf{z}^i||_2^2\right].$$

The second approach models the prior $p_\theta(\mathbf{x}|\mathbf{z}^i, c)$ autoregressively. We only briefly summarize this method as it performs slightly worse than the first method and full implementation details are not published. First, principal component analysis (PCA) [37] is applied to the CLIP image embeddings $\mathbf{z}_i$, reducing the latent space

[11] `https://github.com/lucidrains/DALLE2-pytorch`

dimensionality (here, 319 out of 1024 are kept, causing only a 1% reconstruction loss). The principal components are ordered by decreasing eigenvalue magnitude and quantized each to 1024 discrete buckets producing a sequence which a transformer with causal attention mask is trained to predict. To condition on the text information, the sequence is prepended by embeddings of c, $\mathbf{z}^t$, and a token indicating the (quantized) dot product $(\mathbf{z}^i)^{\mathrm{T}}\mathbf{z}^t$ between the text embedding and the image embedding.

Imagen In [47], Saharia et al. present the text-to-image model *Imagen* which introduces three major innovations. First, the text embeddings for text conditioning are not trained specifically for the diffusion model like in GLIDE, which trains a text encoder transformer jointly with the model, or unCLIP, where the text embeddings come from a CLIP model trained on the training data. Instead, the text embeddings are generated by the pretrained large language model T5-XXL. These embeddings are not trained with computer vision tasks in mind, but training on a data set much larger than the captions of the training set images seems to outweigh this initial drawback.

Second, the authors introduce *dynamic thresholding*. Recall that $\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t)$ predicts pixel value means in the range $[-1, 1]$ which map linearly (see Section 5) to $[0, 255]$. During training, the input $\mathbf{x}_t$ to $\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t)$ is always a (3 channel) image with pixel values in $[-1, 1]$, but the output may exceed this range. Although the training loss encourages $\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t)$ to stay withing the bounds, the necessity to predict black and white pixels (i.e., -1 and 1) leads to $\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t)$ not being clipped to $[-1, 1]$. During sampling, previous models therefore clipped the output of $\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t)$ to $[-1, 1]$ before sampling $\mathbf{x}_{t-1}$, a technique which may be referred to as *static thresholding*. This however causes pixel value predictions to be biased towards the boundaries, resulting in overly saturated images, in particular when using classifier-free guidance with a large guidance weight w. As an alternative to static thresholding, the authors propose *dynamic thresholding* in which a boundary s is defined as a percentile p of absolute pixel values (i.e., $s = \min\{s' \mid \text{at least } p \text{ percent of all pixel values are within } [-s', s']\}$) and if $s > 1$, then the output of $\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t)$ is clipped to $[-s, s]$ and divided by s, resulting in a smoother restriction to the interval $[-1, 1]$. Here, p was chosen for example as $p = 99.5\%$.

Third and finally the authors introduce a modified U-Net architecture they term "Efficient U-Net". Unfortunately, it is not possible to have full certainty over all details of the implementation as no official implementation has been published. There exist (at least) two unofficial implementations,[12][13] but both deviate from the architecture specified in the paper. As the paper itself contains somewhat contradictory statements on the architecture, it is not clear whether the paper leaves out certain details or whether the descriptions are, in fact, accurate. The following are some of the puzzles in the paper.

- The paper states that the base architecture is the U-Net from [35], i.e., with the first 2 improvements from Section 6.1. However, training parameters include a

[12] `http://github.com/lucidrains/imagen-pytorch`
[13] `https://github.com/cene555/imagen-pytorch`

number of attention heads, the use of BigGAN upsampling and downsampling, and a parameter `use_scale_shift_norm` which usually specifies the use of AdaGN. This indicates the architecture includes (some of) the improvements from [8], i.e., the rest of Section 6.1.

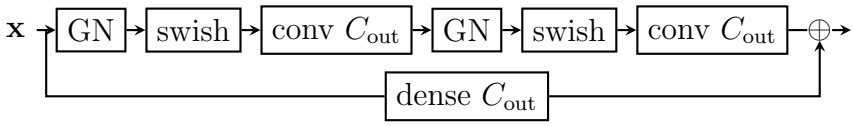
- The ResNet block does not specify any dropout layer. Training parameters specified in Section F of the paper indicate that no dropout was used, so this may be accurate.

- The ResNet block does not obtain a temporal embedding. This may be accurate as the downsampling and upsampling blocks do specify where the temporal embedding is included, but it is still surprising as the deepest downsampling blocks then contain up to 8 ResNet blocks in a row without outside embedding (temporal, text) input.

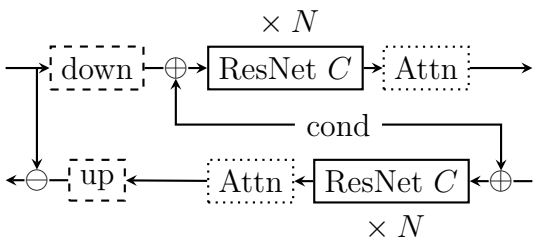
The following is thus my best guess at the Efficient U-Net architecture.

The overall model architecture is a cascading diffusion model with a base 64×64 denoising model and two super-resolution denoising models $64 \times 64 \to 256 \times 256$ and $256 \times 256 \to 1024 \times 1024$. Each of the three denoising models is designed and trained with specific modifications. Both super-resolution models are trained using noise conditioning augmentation as suggested for cascading diffusion models [19].

The ResNet blocks lose the temporal embedding and the dropout layer. It is unclear whether any of the Group Norm layers is implemented as AdaGN and whether the dense layer on the residual connection is always in effect as the paper suggests (previous architectures only use the projection when the number of input channels differs from the number of output channels). According to the paper, a single ResNet block with C_{out} output channels has the following architecture.

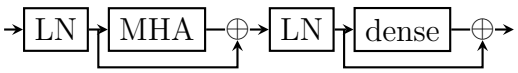


The downsampling and upsampling blocks are again paired by skip connections, which are scaled by $1/\sqrt{2}$. The actual downsampling and upsampling layers as well as the attention layers are optional, as before. The downsampling layer is a 3×3 convolution with stride $s = 2$ and the upsampling layer also uses a 3×3 convolution, very likely preceded by a nearest neighbor upsampling. Efficient U-Net is designed such that deeper downsampling blocks and their upsampling partners contain more ResNet blocks than the blocks near the top of the network. The following is the design of a downsampling/upsampling pair.



It is not clear how the conditional embeddings are fed into the blocks, so the following assumes the usual addition to the hidden state. The conditional embedding consists of the temporal embedding, the noise conditioning augmentation level (a scalar in $[0, 1]$ indicating how much noise was added), and an attention pooled text embedding vector. The nature of these embeddings is also not clear. As the authors use a time continuous diffusion process with $t \in [0, 1]$, the embeddings are likely created as for [29], i.e., a sinusoidal positional encoding vector for $1000t$ is used.[14] Creating an embedding for the noise level in the same way seems natural, but this is pure speculation. The attention pooled text embedding mechanism is also not specified further, but following [39] and a mention of Layer Normalization in the paper, the attention pooled vector is likely created by a Layer Normalization followed by a multi-head attention where q is the mean pooling of the T5-XXL text embeddings and k and v are the T5-XXL embeddings. A different natural choice would be to choose q as a vector of trainable parameters [57]. Following up the multi-head attention with another Layer Normalization and a dense layer would be common practice and would also allow projecting to the dimension of the temporal embedding vector. Following the design of previous ResNet blocks and considering the need to project to the dimensionality of the block, the combined embedding vector is likely fed to a dense layer with swish activation.

The only information on the attention layer design is the use of Layer Normalization and the use of twice the number of channels within the multi-head attention, compared to the number of input and output channels. Moreover, the text embeddings are projected and concatenated to the keys and values of the self attention layers. Attention blocks are used at resolutions 32×32, 16×16, and 8×8 in the 64×64 model and the $64 \times 64 \to 256 \times 256$ model. The $256 \times 256 \to 1024 \times 1024$ model does not use self attention, so the attention layers become pure cross attention layers. Following [44], the attention layer may be implemented as follows.



The full Efficient U-Net architecture begins with a 3×3 stride $s = 1$ convolution from the 3 input channels to 128 output channels, is followed by pairs of downsampling and upsampling blocks (specific to each of the three diffusion models), and closes with a dense layer to 3 output channels. Further details on the architecture are specified in Section F of the paper, such as the downsampling/upsampling block configurations for the different models (although I cannot fully follow them) or the noise schedules and optimizer (hyperparameter) choice. We just note that both the cosine schedule and the linear schedule are used as a noise schedule and that Adam and Adafactor [51] are used for optimization.

10 Fine-Tuning Text-to-Image Models

If a text-to-image diffusion model did not encounter any image of a gömböc during training or if none of the gömböc images was labeled accordingly, the model cannot

[14] `http://github.com/google-research/vdm`

sample an image of one. Fine-tuning aims to inject a new identifier-object pairing into the "knowledge base" of the model which allows images of the object to be sampled by using the identifier. In the following, we review some well-known fine-tuning approaches.

LoRA LoRA (Low Rank Adaptation) [20] is a general approach to fine-tuning neural networks. Abstractly, fine-tuning a network with model parameters θ produces a new set of parameters $\theta' = \theta + \Delta\theta$. For large networks, this comes with costly fine-tuning training process, as all of the parameters θ have to be fine-tuned, and with a large set of new parameters which have to be stored. LoRA aims to resolve this problem by training a much smaller set of parameters ϕ from which $\Delta\theta$ can be synthesized. Specifically, assume a pretrained weight matrix $W \in R^{k\times d}$ in a neural network. LoRA modifies the model architecture and models $W' = W + \frac{\alpha}{r}BA$, where $A \in \mathbb{R}^{r\times d}$ and $B \in \mathbb{R}^{k\times r}$ such that $r \ll \min(k, d)$ and α is a hyperparameter. During fine-tuning, only A and B are trainable while W is frozen. In some experiments of [20], r could be chosen as small as 1 or 2. In [20], A was initialized with a normal distribution and B was initialized as 0.

The authors notably investigate the effectiveness of fine-tuning only the projection matrices W_q, W_k, W_v, W of the multi-head attention layers of several transformer encoders and large language models with positive results. This very approach is used to fine-tune the cross attention layers of pretrained Stable Diffusion models.

Textual Inversion In addition to the computational and storage costs of fine-tuning a model, fine-tuning on a very small dataset can lead to *catastrophic forgetting* of prior knowledge [9, 34]. In [13], the authors therefore suggest a method of identifier-object injection which avoids retraining any model weights. Instead, the fine-tuning process optimizes the embedding vector of a new word added to the vocabulary of the model.

The authors experiment with an LDM [44] model in which BERT [7] is used to produce text embeddings for text conditioning. The first layer of a BERT encoder maps each word of a sequence of input words (i.e., string tokens) to an embedding vector. The sequence of embedding vectors is then fed into a multi-layer transformer which produces the final embedding vectors. The authors suggest to add a new word S_* to the vocabulary, initialize its embedding vector with the embedding vector of an existing word which broadly fits the object's category (e.g., "person" or "man" for images of one specific individual in these categories) and then optimize the usual LDM training loss while freezing all weights but those of the embedding vector.

More specifically, as few as 3-5 images of the object to be injected are randomly sampled from, embedded to the latent space by the encoder $\mathcal{E}$, noised by randomly sampled noise, and the amount of added noise is predicted using the noisy image $\mathbf{z}_t$, the time t, and a text embedding generated by BERT from a randomly sampled caption $c(S_*)$. The captions are derived from the CLIP ImageNet templates [39] and contain prompts of the form "A photo of S_*" or "A rendition of S_*". The full list is given in the paper.

DreamBooth In contrast to the previous approaches, DreamBooth [46] does fine-tune all model weights but aims to prevent catastrophic forgetting, over-fitting, and mode collapse by using a different training loss termed *autogenous class-specific prior preservation loss*. In addition, the identifier for the object to be injected into the model is carefully chosen to be amenable for fine-tuning.

As in the previous methods, DreamBooth aims to bind a new identifier to the object chosen for injection. One detail of text encoders, in particular of BERT, we glossed over so far is that words are usually not tokenized as whole words but as subwords. For instance, the word "example" might be divided into the three tokens "ex–amp–le•", where • is an end-of-word marker. Thus, if the identifier is tokenized into very common subwords, the fine-tuning process is prone to interfere with prior training. This is in particular the case with seemingly very unique identifiers such as "xxy5syt00", which may be tokenized into single letters, resulting in very common tokens. The authors therefore identify a sequence of k "rare" tokens in the vocabulary, e.g., tokens added later during the WordPiece token generation algorithm used for BERT. According to the authors, a relatively short sequence of $k \in \{1, 2, 3\}$ tokens consisting of up to 3 Unicode characters (without spaces) sampled randomly uniformly in the T5-XXL tokenizer range of $\{5000, \ldots, 10000\}$ works well.

DreamBooth fine-tunes the diffusion model on a small number of images of the object to be injected. To bind the identifier to the object, the images are labeled with the caption $c =$ "a [identifier] [class noun]", where the class noun is coarse class descriptor of the subject (like for Textual Inversion) and can be user-specified or obtained from a classifier. In order to prevent overfitting "class noun" to the new object during training, a number of N (e.g., $N = 1000$) images $\mathbf{x}_{\text{pr}}^{(1)}, \ldots, \mathbf{x}_{\text{pr}}^{(N)}$ are sampled from the (prior) model before fine-tuning using the caption $c_{\text{pr}} =$ "a [class noun]". The model is then fine-tuned with a loss function that encourages the model to keep the sampled images in its output space, i.e., the following loss is optimized.

$$\mathbb{E}_{\mathbf{x}_0, \mathbf{x}_{\text{pr}}^{(i)}, t, t', \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}'} \big[||\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}, t, c)||_2^2 + \lambda ||\boldsymbol{\varepsilon}' - \boldsymbol{\varepsilon}_\theta(\sqrt{\bar{\alpha}_{t'}}\mathbf{x}_{\text{pr}}^{(i)} + \sqrt{1-\bar{\alpha}_{t'}}\boldsymbol{\varepsilon}', t', c_{\text{pr}})||_2^2 \big]$$

Here, $\mathbf{x}_0$ is uniformly sampled from the images of the object, $i \sim \mathcal{U}([1, N])$, $t, t' \sim \mathcal{U}([1, T])$, and $\boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \sim \mathcal{N}(0, \mathbf{I})$. The parameter λ controls the strength of the class-specific prior preservation loss. The authors trained models for ~ 1000 iterations with $\lambda = 1$ and a learning rate of 10^{-5} for Imagen and $5 \cdot 10^{-6}$ for Stable Diffusion on only 3-5 images each.

Appendix

Tweedie's Formula

Let X, Y be two independent random variables with densities p_X, p_Y and joint density $p_{X,Y}$ (i.e., the density of (X, Y)). Then

1. $p_{X,Y} = p_X p_Y$,

2. the density of $X+Y$ is $p_{X+Y}(y) = \int_{\mathbb{R}} p_X(x)p_Y(y-x)dx$,
3. the density of $Y \mid X$ satisfies $p_{Y|X}(y \mid x)p_X(x) = p_{X,Y}(x,y)$.

The first statements can be derived relatively easily by applying Fubini's Theorem and observing generators of the Borel algebra.

$$\begin{aligned}\int_{x\le c, y\le d} p_{X,Y}(x,y)d(x,y) &= \mathbb{P}(X \le c, Y \le d)\\ &= \mathbb{P}(X \le c)\mathbb{P}(Y \le d)\\ &= \int_{x\le c}\int_{y\le d} p_X(x)p_Y(y)dydx\\ &= \int_{x\le c, y\le d} p_X(x)p_Y(y)d(x,y)\end{aligned}$$

$$\begin{aligned}\mathbb{P}(X+Y \le c) &= \int_{x+y\le c} p_{X,Y}(x,y)d(x,y)\\ &= \int_{\mathbb{R}}\int_{y\le c-x} p_X(x)p_Y(y)dydx\\ &= \int_{\mathbb{R}} p_X(x)\int_{y\le c} p_Y(y-x)dydx\\ &= \int_{y\le c}\int_{\mathbb{R}} p_X(x)p_Y(y-x)dxdy\end{aligned}$$

The last statement requires some work for a rigorous derivation which we do not cover here.

We show that for a random variable $\boldsymbol{\mu}$ with density $p_{\boldsymbol{\mu}}$ and $X \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\Sigma}$ is a constant diagonal covariance matrix, we have

$$\mathbb{E}_{\boldsymbol{\mu}\sim p_{\boldsymbol{\mu}}}[\boldsymbol{\mu} \mid X = x] = x + \boldsymbol{\Sigma}\nabla_x \log p_X(x).$$

We only consider the case where $\boldsymbol{\mu}$,X, and $\boldsymbol{\Sigma}$ are scalars, from which the general statement follows by applying this result component-wise.

Assume that μ is a random variable with density p_μ and $X \sim \mathcal{N}(\mu, \sigma^2)$ where σ is constant. In particular, we may assume $X = \mu + Z$ with $Z \sim \mathcal{N}(0, \sigma^2)$. Then the density of X is $p_X(x) = \int p_\mu(\mu)\mathcal{N}(x-\mu; 0, \sigma^2)d\mu$ and the density of $\mu \mid X$ is $p_{\mu|X=x} = \frac{p_{\mu,X}(.,x)}{p_X(x)} = \frac{p_{X|\mu}(x)p_\mu}{p_X(x)}$. Thus,

$$\begin{aligned}\frac{\mathbb{E}_{\mu\sim p_\mu}[\mu \mid X = x] - x}{\sigma^2} &= \int \frac{\mu - x}{\sigma^2} p_{\mu|X=x}(\mu)d\mu\\ &= \frac{\int \frac{\mu-x}{\sigma^2} p_\mu(\mu)\mathcal{N}(x; \mu, \sigma^2)d\mu}{\int p_\mu(\mu)\mathcal{N}(x-\mu; 0, \sigma^2)d\mu}\\ &= \frac{d}{dx}\log \int p_\mu(\mu)\mathcal{N}(x-\mu; 0, \sigma^2)d\mu\end{aligned}$$

which means $\mathbb{E}_{\mu\sim p_\mu}[\mu \mid X = x] = x + \sigma^2 \frac{d}{dx}\log p_X(x)$. Note that we may swap integration and differentiation as the integrand is an L^1 function for x fixed and the derivatives are L^1 functions if we assume that $\mathbb{E}[\mu] < \infty$.

References

[1] Brian D.O. Anderson. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12(3):313–326, 1982. URL https://doi.org/10.1016/0304-4149(82)90051-5.

[2] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. URL http://arxiv.org/abs/1607.06450.

[3] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. *CoRR*, abs/1809.11096, 2018. URL http://arxiv.org/abs/1809.11096.

[4] Hanqun Cao, Cheng Tan, Zhangyang Gao, Guangyong Chen, Pheng-Ann Heng, and Stan Z. Li. A survey on generative diffusion model. *CoRR*, abs/2209.02646, 2022. URL https://arxiv.org/abs/2209.02646.

[5] Ayan Das. An introduction to diffusion probabilistic models. *http://ayandas.me*, Dec 2021. URL https://ayandas.me/blog-tut/2021/12/04/diffusion-prob-models.

[6] Ayan Das. Generative modelling with score functions. *http://ayandas.me*, July 2021. URL https://ayandas.me/blog-tut/2021/07/14/generative-model-score-function.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

[8] Prafulla Dhariwal and Alex Nichol. Diffusion models beat GANs on image synthesis. *CoRR*, abs/2105.05233, 2021. URL https://arxiv.org/abs/2105.05233.

[9] Yuxuan Ding, Lingqiao Liu, Chunna Tian, Jingyuan Yang, and Haoxuan Ding. Don't stop learning: Towards continual learning for the CLIP model. *CoRR*, abs/2207.09248, 2022. URL https://arxiv.org/abs/2207.09248.

[10] Alexey Dosovitskiy and Thomas Brox. Generating images with perceptual similarity metrics based on deep networks. *CoRR*, abs/1602.02644, 2016. URL http://arxiv.org/abs/1602.02644.

[11] Bradley Efron. Tweedie's formula and selection bias. *Journal of the American Statistical Association*, 106(496):1602–1614, 2011.

[12] Patrick Esser, Robin Rombach, and Björn Ommer. Taming transformers for high-resolution image synthesis. *CoRR*, abs/2012.09841, 2020. URL https://arxiv.org/abs/2012.09841.

[13] Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H. Bermano, Gal Chechik, and Daniel Cohen-Or. An image is worth one word: Personalizing

text-to-image generation using textual inversion. *CoRR*, abs/2208.01618, 2022. URL https://arxiv.org/abs/2208.01618.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

[15] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, Günter Klambauer, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a Nash equilibrium. *CoRR*, abs/1706.08500, 2017. URL http://arxiv.org/abs/1706.08500.

[16] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. URL http://arxiv.org/abs/1207.0580.

[17] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *CoRR*, abs/2207.12598, 2022. URL https://arxiv.org/abs/2207.12598.

[18] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *CoRR*, abs/2006.11239, 2020. URL https://arxiv.org/abs/2006.11239.

[19] Jonathan Ho, Chitwan Saharia, William Chan, David J. Fleet, Mohammad Norouzi, and Tim Salimans. Cascaded diffusion models for high fidelity image generation. *CoRR*, abs/2106.15282, 2021. URL https://arxiv.org/abs/2106.15282.

[20] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. *CoRR*, abs/2106.09685, 2021. URL https://arxiv.org/abs/2106.09685.

[21] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL http://arxiv.org/abs/1502.03167.

[22] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016. URL http://arxiv.org/abs/1611.07004.

[23] Sergios Karagiannakos and Nikolaos Adaloglou. Diffusion models: toward state-of-the-art image generation. *https://theaisummer.com/*, 2022. URL https://theaisummer.com/diffusion-models/.

[24] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *CoRR*, abs/1812.04948, 2018. URL http://arxiv.org/abs/1812.04948.

[25] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

[26] Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *CoRR*, abs/1807.03039, 2018. URL http://arxiv.org/abs/1807.03039.

[27] Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. *CoRR*, abs/1312.6114, 2013. URL http://arxiv.org/abs/1312.6114.

[28] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *CoRR*, abs/1906.02691, 2019. URL http://arxiv.org/abs/1906.02691.

[29] Diederik P. Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. *CoRR*, abs/2107.00630, 2021. URL https://arxiv.org/abs/2107.00630.

[30] Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. Revisiting self-supervised visual representation learning. *CoRR*, abs/1901.09005, 2019. URL http://arxiv.org/abs/1901.09005.

[31] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *CoRR*, abs/1711.05101, 2017. URL http://arxiv.org/abs/1711.05101.

[32] Mario Lucic, Michael Tschannen, Marvin Ritter, Xiaohua Zhai, Olivier Bachem, and Sylvain Gelly. High-fidelity image generation with fewer labels. *CoRR*, abs/1903.02271, 2019. URL http://arxiv.org/abs/1903.02271.

[33] Calvin Luo. Understanding diffusion models: A unified perspective. *CoRR*, abs/2208.11970, 2022. URL http://arxiv.org/abs/2208.11970.

[34] Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pages 109–165. Academic Press, 1989.

[35] Alex Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. *CoRR*, abs/2102.09672, 2021. URL https://arxiv.org/abs/2102.09672.

[36] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. GLIDE: towards photorealistic image generation and editing with text-guided diffusion models. *CoRR*, abs/2112.10741, 2021. URL https://arxiv.org/abs/2112.10741.

[37] Karl Pearson. LIII. On lines and planes of closest fit to systems of points in space. 1901. URL https://doi.org/10.1080/14786440109462720.

[38] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. FiLM: Visual reasoning with a general conditioning layer. *CoRR*, abs/1709.07871, 2017. URL http://arxiv.org/abs/1709.07871.

[39] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. *CoRR*, abs/2103.00020, 2021. URL https://arxiv.org/abs/2103.00020.

[40] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. *CoRR*, abs/1710.05941, 2017. URL http://arxiv.org/abs/1710.05941.

[41] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with CLIP latents. *CoRR*, abs/2204.06125, 2022. URL http://arxiv.org/abs/2204.06125.

[42] Ali Razavi, Aäron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with VQ-VAE-2. *CoRR*, abs/1906.00446, 2019. URL http://arxiv.org/abs/1906.00446.

[43] Niels Rogge and Kashif Rasul. The annotated diffusion model. *https://huggingface.co/*, July 2022. URL https://huggingface.co/blog/annotated-diffusion/.

[44] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. *CoRR*, abs/2112.10752, 2021. URL http://arxiv.org/abs/2112.10752.

[45] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL http://arxiv.org/abs/1505.04597.

[46] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. DreamBooth: Fine tuning text-to-image diffusion models for subject-driven generation. *CoRR*, abs/2208.12242, 2022. URL https://doi.org/10.48550/arXiv.2208.12242.

[47] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Raphael Gontijo Lopes, Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. *CoRR*, abs/2205.11487, 2022. URL http://arxiv.org/abs/2205.11487.

[48] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. *CoRR*, abs/1606.03498, 2016. URL http://arxiv.org/abs/1606.03498.

[49] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P. Kingma. PixelCNN++: Improving the PixelCNN with discretized logistic mixture likelihood and other modifications. *CoRR*, abs/1701.05517, 2017. URL http://arxiv.org/abs/1701.05517.

[50] D. Sculley. Web-scale k-means clustering. In *International Conference on World Wide Web*. ACM, 2010. URL https://doi.org/10.1145/1772690.1772862.

[51] Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. *CoRR*, abs/1804.04235, 2018. URL http://arxiv.org/abs/1804.04235.

[52] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *CoRR*, abs/1503.03585, 2015. URL http://arxiv.org/abs/1503.03585.

[53] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *CoRR*, abs/2010.02502, 2020. URL http://arxiv.org/abs/2010.02502.

[54] Yang Song. Generative modeling by estimating gradients of the data distribution. *http://yang-song.net*, May 2021. URL http://yang-song.net/blog/2021/score/.

[55] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *CoRR*, abs/2011.13456, 2020. URL https://arxiv.org/abs/2011.13456.

[56] Alexandre Thiéry. Reverse diffusions, score & Tweedie. *https://alexxthiery.github.io/*, June 2023. URL https://alexxthiery.github.io/posts/reverse_and_tweedie/reverse_and_tweedie.html.

[57] Hugo Touvron, Matthieu Cord, Alaaeldin El-Nouby, Piotr Bojanowski, Armand Joulin, Gabriel Synnaeve, and Hervé Jégou. Augmenting convolutional networks with attention-based aggregation. *CoRR*, abs/2112.13692, 2021. URL https://arxiv.org/abs/2112.13692.

[58] Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. *CoRR*, abs/1711.00937, 2017. URL http://arxiv.org/abs/1711.00937.

[59] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

[60] Lilian Weng. What are diffusion models? *lilianweng.github.io*, July 2021. URL https://lilianweng.github.io/posts/2021-07-11-diffusion-models/.

[61] Yuxin Wu and Kaiming He. Group normalization. *CoRR*, abs/1803.08494, 2018. URL http://arxiv.org/abs/1803.08494.

[62] Jiahui Yu, Xin Li, Jing Yu Koh, Han Zhang, Ruoming Pang, James Qin, Alexander Ku, Yuanzhong Xu, Jason Baldridge, and Yonghui Wu. Vector-quantized image modeling with improved VQGAN. *CoRR*, abs/2110.04627, 2021. URL https://arxiv.org/abs/2110.04627.

[63] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016. URL `http://arxiv.org/abs/1605.07146`.

[64] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. *CoRR*, abs/1801.03924, 2018. URL `http://arxiv.org/abs/1801.03924`.