Lab Vision

Richard Varhelyi (s35rvarh)
Uni Bonn

1 Probabilistic Diffusion Models

1.1 Theory

Diffusion models are latent variable models that consist of an encoder and a decoder. The encoder takes a data sample $\mathbf{x}$ and maps it through a series of latent variables $\mathbf{z}_1 ... \mathbf{z}_T$. This process is defined as a Markov chain that gradually adds Gaussian noise to the data according to a scheduler $\beta_1 ... \beta_T$:

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

The diffusion kernel is the closed-form conditional distribution $q(\mathbf{x}_t \mid \mathbf{x}_0)$ which is the result of composing all the small Gaussian transitions $q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$ into a single Gaussian from time 0 to t. This can be directly computed in the forward process in the form:

$$q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}\left(\sqrt{\bar{\alpha}_t}\,\mathbf{x}_0, (1-\bar{\alpha}_t)I\right), \qquad \bar{\alpha}_t = \prod_{t=1}^{T}(1-\beta_t)$$

Equivalently, in reparameterized form for sampling:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, I).$$

The β_t scheduler is designed such that $\bar{\alpha}_t \to 0$, thus $q(\mathbf{x}_T \mid \mathbf{x}_0) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$. This property is crucial to speed up the training process, because we can directly sample $\mathbf{x}_t$ from $\mathbf{x}_0$ at any arbitrary time t. The backward process is a generative approach where we try to generate a new image $\mathbf{x}$. The process is defined as:

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t),$$

where, $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \underbrace{\mu_\theta(\mathbf{x}_t, t)}_{\substack{\text{Trainable network} \\ \text{(U-Net, Denoising Autoencoder)}}} \sigma^2 \mathbf{I})$ and $p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$.

For training, we would like to learn the objective distribution. The likelihood of the observed variable $\mathbf{x}$ and the latent variables $\mathbf{z}_t$ is defined as:

$$Pr(\mathbf{x}, \mathbf{z}_{1...T} | \phi_{1...T}) = Pr(\mathbf{x} | \mathbf{z}_1, \phi_1) \prod_{t=2}^{T} Pr(\mathbf{z}_{t-1} | \mathbf{z}_t, \phi_t) \cdot Pr(\mathbf{z}_T)$$

Marginalization over latent variables is:

$$Pr(\mathbf{x}|\phi_{1\ldots T}) = \int Pr(\mathbf{x}, \mathbf{z}_{1\ldots T}|\phi_{1\ldots T})\, d\mathbf{z}_{1\ldots T}$$

For training, maximize log-likelihood:

$$\hat{\phi}_{1\ldots T} = \underset{\phi_{1\ldots T}}{\operatorname{argmax}} \left[\sum_{i=1}^{I} \log \left[Pr(\mathbf{x}_i|\phi_{1\ldots T}) \right] \right]$$

However, evaluating the marginal distribution involves integrating over all possible trajectories from noise to the data manifold, which is intractable. Instead we can maximize a lower bound of the log likelihood.

$$\begin{aligned}
\log \Pr(x \mid \phi_{1\ldots T}) &= \log \int \Pr(x, z_{1\ldots T} \mid \phi_{1\ldots T})\, dz_{1\ldots T} \\
&= \log \int q(z_{1\ldots T} \mid x) \frac{\Pr(x, z_{1\ldots T} \mid \phi_{1\ldots T})}{q(z_{1\ldots T} \mid x)}\, dz_{1\ldots T} \\
&\geq \int q(z_{1\ldots T} \mid x) \log \frac{\Pr(x, z_{1\ldots T} \mid \phi_{1\ldots T})}{q(z_{1\ldots T} \mid x)}\, dz_{1\ldots T},
\end{aligned}$$

This gives us the evidence lower bound (ELBO):

$$\text{ELBO}(\phi_{1\ldots T}) = \int q(z_{1\ldots T} \mid x) \log \frac{\Pr(x, z_{1\ldots T} \mid \phi_{1\ldots T})}{q(z_{1\ldots T} \mid x)}\, dz_{1\ldots T}.$$

To fit the model, we maximize the ELBO with respect to the parameters $\phi_{1\ldots T}$. The simplified loss function is:

$$L[\phi_{1\ldots T}] = \sum_{i=1}^{I} \sum_{t=1}^{T} \left|\left| \mathbf{g}_t \left[\sqrt{\alpha_t} \cdot \mathbf{x}_i + \sqrt{1-\alpha_t} \cdot \epsilon_{it},\, \phi_t \right] - \epsilon_{it} \right|\right|^2,$$

The training algorithm in practice can be implemented as follows:

Algorithm 18.1: Diffusion model training

Input: Training data $\mathbf{x}$
Output: Model parameters $\boldsymbol{\phi}_t$
repeat
  for $i \in \mathcal{B}$ **do** // For every training example index in batch
    $t \sim \text{Uniform}[1, \ldots T]$ // Sample random timestep
    $\boldsymbol{\epsilon} \sim \text{Norm}[\mathbf{0}, \mathbf{I}]$ // Sample noise
    $\ell_i = \left|\left| \mathbf{g}_t \left[\sqrt{\alpha_t}\mathbf{x}_i + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}, \boldsymbol{\phi}_t \right] - \boldsymbol{\epsilon} \right|\right|^2$ // Compute individual loss
  Accumulate losses for batch and take gradient step
until converged

For generating a new image $\mathbf{x}$, the algorithm can be implemented as:

Algorithm 18.2: Sampling

Input: Model, $\mathbf{g}_t[\bullet, \boldsymbol{\phi}_t]$
Output: Sample, $\mathbf{x}$
$\mathbf{z}_T \sim \text{Norm}_{\mathbf{z}}[\mathbf{0}, \mathbf{I}]$ // Sample last latent variable
for $t = T \ldots 2$ **do**
  $\hat{\mathbf{z}}_{t-1} = \frac{1}{\sqrt{1-\beta_t}}\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}\sqrt{1-\beta_t}}\mathbf{g}_t[\mathbf{z}_t, \boldsymbol{\phi}_t]$ // Predict previous latent variable
  $\boldsymbol{\epsilon} \sim \text{Norm}_{\boldsymbol{\epsilon}}[\mathbf{0}, \mathbf{I}]$ // Draw new noise vector
  $\mathbf{z}_{t-1} = \hat{\mathbf{z}}_{t-1} + \sigma_t \boldsymbol{\epsilon}$ // Add noise to previous latent variable
$\mathbf{x} = \frac{1}{\sqrt{1-\beta_1}}\mathbf{z}_1 - \frac{\beta_1}{\sqrt{1-\alpha_1}\sqrt{1-\beta_1}}\mathbf{g}_1[\mathbf{z}_1, \boldsymbol{\phi}_1]$ // Generate sample from $\mathbf{z}_1$ without noise

Now comes the question how to guide our image generation process. We want to condition the process on a new condition $\mathbf{c}$. Given a conditioning image $\mathbf{c} \in \mathbb{R}^{H \times W \times 3}$, the conditional reverse process modifies the unconditional denoising process to incorporate this guidance. The forward process remains unchanged. The reverse process conditioned on $\mathbf{c}$ is defined as:

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{c}) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t, \mathbf{c}), \Sigma_\theta(\mathbf{x}_t, t, \mathbf{c}))$$

The training objective becomes:

$$L[\phi_{1\ldots T}] = \sum_{i=1}^{I}\sum_{t=1}^{T} \left|\left| \mathbf{g}_t\left[\sqrt{\alpha_t} \cdot \mathbf{x}_i + \sqrt{1-\alpha_t} \cdot \epsilon_{it},\, \mathbf{c}_i,\, \phi_t\right] - \epsilon_{it} \right|\right|^2,$$

1.2 Implementation of the baseline model

For the implementation and training of the diffusion model, the CelebA dataset has been used, which contains $202,599$ face images. To reduce computational cost and accelerate training, all images has been resized to 64×64 pixels. Before training, the images are normalized to the range $[-1, 1]$, aligning the data distribution with a zero-mean Gaussian. This normalization facilitates more stable training and is consistent with the assumptions commonly made in diffusion-based generative models. In the next step the β scheduler has been implemented, which follows the implementation of the original paper going from value 1×10^{-4} to 0.02. Following this, the forward diffusion process has been implemented. During this phase, Gaussian noise is added incrementally to the data according to the predefined β schedule. The evolution of this process is illustrated in Figure 1.

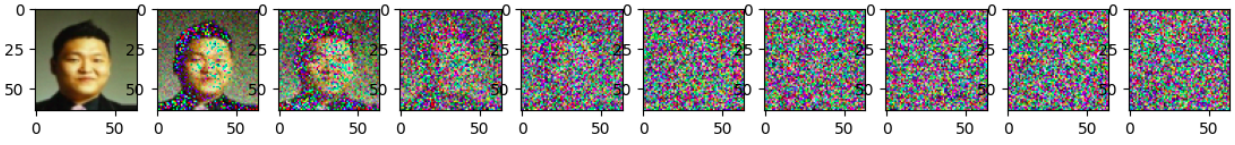

Figure 1: Visualization of the forward diffusion process over increasing time steps.

The most challenging component has been the implementation of the neural network responsible for learning the denoising process. For this purpose, a U-Net architecture has been employed,

augmented with sinusoidal positional embeddings to encode the diffusion time step t at each layer. In addition, self-attention modules have been integrated to enable the network to capture long-range dependencies and global spatial relationships within the images. Due to their computational cost, self-attention layers have been applied only at deeper levels of the network and at the bottleneck. The model has been trained using the MSE loss, optimized with the Adam optimizer and a learning rate of 1×10^{-4}. Following the methodology of the original paper, an exponential moving average (EMA) of the model parameters has been maintained during training. This technique stabilizes the optimization process and leads to smoother convergence. Training was performed on a NVIDIA P100 GPU with 16GB of VRAM for 50 epochs, using a total of $T = 1000$ diffusion steps. It took approximately 10 hours. The qualitative result of the image generation process is presented in Figure 2.

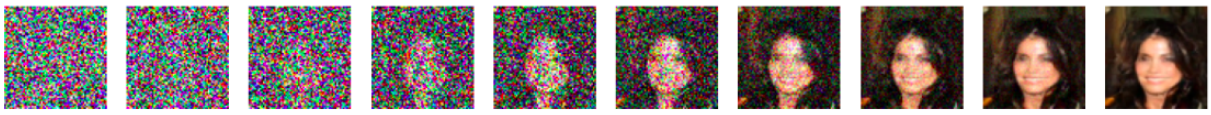

Figure 2: Visualization of the backward diffusion process over increasing time steps.

2 Diffusion in latent space

2.1 Variational Autoencoders

Variational Autoencoders are probabilistic generative models. They aim to learn a distribution $p(\mathbf{x})$ over the data, which can be represtented as a multi-dimensional variable $\mathbf{x}$. Our goal is to generate new data from a given distribution, $p(\mathbf{x})$, which represents our dataset. The problem is, we don't know the exact shape or properties of $p(\mathbf{x})$. We only have access to some samples, for instance images from our training split. To make working with $p(\mathbf{x})$ easier, we introduce another distribution, $p(\mathbf{z})$, called the latent distribution. This distribution represents latent variables in a lower-dimensional space that capture the core features of the data. We assume that the latent distribution is in fact a normal distribution $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$. This assumption enables efficient sampling and provides a regularized structure for the latent space. Given a latent variable $\mathbf{z}$, the generative model defines a likelihood distribution $p_\theta(\mathbf{x} \mid \mathbf{z})$, which measures the probability of reconstructing an observation $\mathbf{x}$ from $\mathbf{z}$. This distribution is parameterized by a decoder network with parameters $\boldsymbol{\theta}$. A central challenge is the computation of the posterior distribution $p_\theta(\mathbf{z} \mid \mathbf{x})$. For neural network–based generative models, this posterior is generally intractable due to the nonlinear mapping introduced by the decoder. To address this issue, variational inference is employed. Instead of computing the true posterior directly, we introduce an approximate posterior distribution $q_\phi(\mathbf{z} \mid \mathbf{x})$, parameterized by ϕ, and typically modeled as a Gaussian distribution $q(\mathbf{z} \mid \mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}_z\big(\boldsymbol{g}_\mu(\mathbf{x}, \boldsymbol{\theta}), \boldsymbol{g}_\Sigma(\mathbf{x}, \boldsymbol{\theta})\big)$, where $\boldsymbol{g}[\mathbf{x}, \boldsymbol{\theta}]$ is a neural network with parameters $\boldsymbol{\theta}$ that predicts the mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\Sigma}$ of the normal variational approximation. During training, latent samples are drawn using the reparameterization trick,

$$\mathbf{z}^* = \boldsymbol{\mu} + \boldsymbol{\Sigma}^{1/2}\epsilon^*, \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

The model is trained by maximizing the Evidence Lower Bound (ELBO), which provides a tractable surrogate objective for the marginal likelihood $\log p_\theta(\mathbf{x})$. The ELBO is defined as

$$\mathrm{ELBO}[\theta, \phi] = \int q(\mathbf{z} \mid \mathbf{x}, \theta) \, \log p(\mathbf{x} \mid \mathbf{z}, \phi) \, dz \; - \; D_{\mathrm{KL}}\big(q(\mathbf{z} \mid \mathbf{x}, \theta) \,\|\, p(\mathbf{z})\big).$$

The first term corresponds to the reconstruction objective, encouraging the decoder to accurately reproduce the input data, while the second term is the Kullback–Leibler divergence that regularizes the approximate posterior to remain close to the prior distribution. This regularization promotes a smooth and well-structured latent space. By jointly optimizing the encoder and decoder networks to maximize the ELBO, the VAE simultaneously learns an approximate posterior distribution over latent variables and a generative model capable of reconstructing new samples.

2.2 Implementation of a Variational Autoencoder

First, an encoder network is implemented to map the input data to a latent distribution. In our case, the inputs consist of $64 \times 64 \times 3$ RGB images, which are progressively downsampled to compact latent representations of size $8 \times 8 \times 4$. Using the reparameterization trick, latent vectors can be sampled from this distribution in a differentiable manner, enabling end-to-end training of the model. This formulation also allows the generation of new samples by drawing points from the learned latent space, including images that were not present in the original dataset. Each encoder layer consists of a residual block composed of two convolutional layers, each followed by group normalization and a ReLU activation function. The residual block is followed by a convolutional layer with stride 2, which downsamples the spatial resolution by a factor of two. In the final stage, the encoder outputs the parameters of the latent distribution, namely the mean $\boldsymbol{\mu}$ and the log variance $\log(\boldsymbol{\sigma}^2)$. The decoder mirrors the encoder architecture and employs the same residual block structure. However, instead of downsampling, transposed convolutional layers were used to progressively upsample the latent representations back to the original image resolution. This enables the reconstruction of $64 \times 64 \times 3$ RGB images from the latent space. Finally, a $tanh(\mathbf{x})$ activation function is applied at the output layer to constrain the pixel values to the range $[-1, 1]$, ensuring consistency with the input normalization. The model has been trained using the loss function:

$$\mathcal{L}(\theta, \phi) = \mathbb{E}_{q_\phi(z|x)} \left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right] - D_{\mathrm{KL}} \left(q_\phi(\mathbf{z} \mid \mathbf{x}) \,\|\, p(\mathbf{z})\right).$$

This is identical to the formulation in the previous chapter because

$$\mathbb{E}_{q(z|x)}[f(\mathbf{z})] = \int q(\mathbf{z} \mid \mathbf{x}) \, f(\mathbf{z}) \, d\mathbf{z}.$$

The Adam optimizer has been used with a learning rate of 1×10^{-4}. Training was performed on an NVIDIA P100 GPU with 16GB of VRAM for 20 epochs. It took approximately 4 hours. The qualitative result of the VAE is presented in Figure 3.

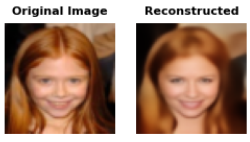

Figure 3: Comparison between original and VAE encoded and reconstructed image.

2.3 Implementation of the latent space diffusion

To integrate the diffusion model with the learned latent space, the input dimensionality has been adapted from the original image resolution of $64 \times 64 \times 3$ to a compact latent representation of $8 \times 8 \times 4$. Consequently, the network architecture has been simplified by removing several layers, as the reduced spatial resolution no longer requires a deep hierarchy of feature maps. Training has been performed following the same procedure described in Section 1.2. Operating in the compressed latent space significantly reduced the computational cost, resulting in faster convergence. Specifically, training was completed in 2 hours and 42 minutes, compared to 10 hours for the baseline diffusion model, corresponding to a speedup of nearly 4 times. During training, input images are first encoded into the latent space using the VAE encoder. Latent samples are then drawn using the reparameterization trick and provided as input to the diffusion model, which learns to denoise these latent representations. The output of the diffusion model is subsequently decoded by the VAE decoder to reconstruct images in the original data space. The qualitative result of the latent diffusion generation process is presented in Figure 4.

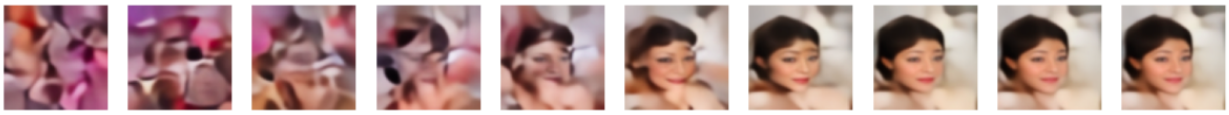

Figure 4: Visualization of the backward diffusion process in latent space.

3 Monocular Depth Estimation using diffusion

To further evaluate the flexibility of diffusion models, we apply our implementation to a downstream computer vision task. Diffusion-based approaches can be adapted to a variety of applications, including image-to-image translation, segmentation, and depth estimation. In this work, we focus on monocular depth estimation, a common task in robotic perception and 3D scene understanding. For this purpose, we employ our baseline diffusion model and train it on the NYU Depth V2 dataset, which contains $50,688$ paired RGB images and the corresponding depth maps.

Figure 5: Image and depth mask pair from the NYU Depth V2 dataset.

All samples are resized to a spatial resolution of 64×64 pixels. The RGB images are concatenated with the corresponding depth map along the channel dimension, resulting in an input tensor of size $64 \times 64 \times 4$. The data is normalized to the range $[-1, 1]$ to ensure stable training. The U-Net architecture is adapted accordingly to accept four input channels, while the network output consists of a single-channel prediction of size $64 \times 64 \times 1$, representing the estimated depth

map. During training, the diffusion model learns to denoise the depth component, and the loss is computed only on the predicted depth values. This method can be analogous to conditioning mechanisms such as feature encoders or cross-attention used in guided diffusion models. Training is performed using the Adam optimizer with a learning rate of 1×10^{-4} and a batch size of 64 for 50 epochs. A total of $T = 1000$ diffusion steps are used, and optimization is carried out using the MSE loss. The qualitative result of the predictive diffusion generation process is presented in Figure 6.

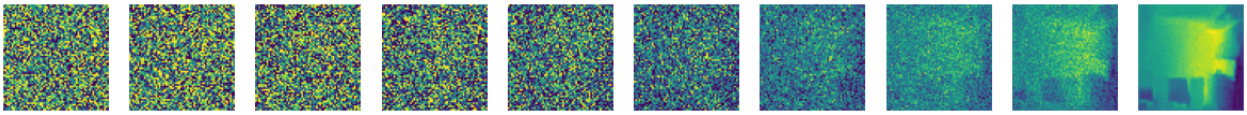

Figure 6: Image and depth mask pair from the NYU Depth V2 dataset.

For the evaluation of the method the following metrics has been used: For each pixel i, let y_i and $\hat{y}_i$ denote the ground truth and predicted depth. The proportion of pixels satisfying

$$\max\left(\frac{y_i}{\hat{y}_i}, \frac{\hat{y}_i}{y_i}\right) < \delta$$

is reported for $\delta_1 = 1.25$, $\delta_2 = 1.25^2$, and $\delta_3 = 1.25^3$. Higher values indicate better accuracy. We also report standard errors over all N valid pixels: the mean absolute relative error

$$\text{REL} = \frac{1}{N}\sum_{i=1}^{N}\frac{|y_i - \hat{y}_i|}{y_i},$$

the squared relative error

$$\text{SqRel} = \frac{1}{N}\sum_{i=1}^{N}\frac{(y_i - \hat{y}_i)^2}{y_i},$$

the root mean squared error

$$\text{RMS} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2},$$

and the log root mean squared error

$$\text{RMS}_{\log} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\log y_i - \log \hat{y}_i)^2}.$$

The method has been evaluated on the test dataset containing 654 pairs. The quantitative result of the method is presented in Table 1.

Table 1: Comparison of performances on the NYU-Depth-v2 dataset

Method	Architecture	$\delta_1 \uparrow$	$\delta_2 \uparrow$	$\delta_3 \uparrow$	REL $\downarrow$	RMS $\downarrow$	$\log_{10} \downarrow$
(ours)	DDPM†	0.661	0.882	0.957	0.217	0.814	0.286

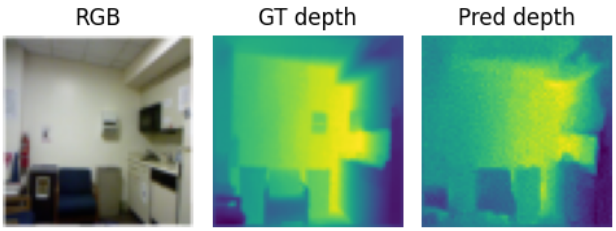

Figure 7: Image and corresponding ground truth and predicted depth map.