Continuous-in-time ResShift

Arina Chumachenko
Skoltech, MIPT

Alex Fokin
Skoltech, MIPT

Pavel Bartenev
Skoltech, MIPT

Julia Sergeeva
Skoltech

Daniil Shlenskii
Skoltech

1 Introduction

Image super-resolution aims to reconstruct a high-resolution image from a given low-resolution counterpart. Current state-of-the-art methods [JH20, RBLPE21, YGL$^+$24] in this realm are diffusion-based, so for inference they require many model evaluations, which can be a problem, for instance, in on-line super-resolving. There is a recently proposed approach called ResShift[YWL23], which supposed to alleviate aforementioned challenge. Even though it was successful, there is still a need to do really high number of model evaluations. We are going to do a step further here. For now ResShift is described by Markov chain similarly to diffusion models at the very beginning. Interpretation of diffusion process as Stochastic Differential Equation (SDE) [SSDK$^+$21] allowed to use SDE-specific techniques (SDE solver, timesteps discretizations) [LZB$^+$22] and to improve quality of diffusion models using fewer number of model evaluations. So we aim to investigate ResShift performance in the continuous setup.
Source code can be found in our github repository.

2 Approach

2.1 Core Idea

The approach in the original article was to transition from high-resolution (HR) to low-resolution (LR) by gradually shifting their residual through a Markov chain of length T:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{y}) = \mathcal{N}(\mathbf{x}_t;\mathbf{x}_{t-1}+\alpha_t\mathbf{e}_0,\kappa^2\alpha_t\mathbf{I}), \quad t=1,2,\dots,T,$$

where $\mathbf{e}_0 = \mathbf{y}-\mathbf{x}_0$ is the residual (difference between HR and LR), and α_t is the scaling factor.

In the article, the posterior distribution $p(\mathbf{x}_0|\mathbf{y})$ is estimated by minimizing the negative Evidence Lower Bound (ELBO):

$$\min_\theta \sum_t D_{KL}\left[q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0,\mathbf{y})\|p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{y})\right],$$

where the inverse transition kernel $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{y})$ is assumed to have the form

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

and the targeted distribution can be expressed as

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0,\mathbf{y}) = \mathcal{N}\left(\mathbf{x}_{t-1};\frac{\eta_{t-1}}{\eta_t}\mathbf{x}_t+\frac{\alpha_t}{\eta_t}\mathbf{x}_0,\kappa^2\frac{\eta_{t-1}}{\eta_t}\alpha_t\mathbf{I}\right), \eta_t=\sum_{i=1}^t\alpha_i$$

2.2 Suggested Modification

Our modification involves rewriting the given Markov Chain as a Stochastic Differential Equation (SDE). This transformation allows the use of higher-order ODE solvers and more flexible time discretization. For diffusion models, it has been shown to improve or preserve the quality of generations while reducing the number of function evaluations (nfe).

2.3 From Markov Chain to ODE

According to ResShift paper, the original Markov Chain have the following form:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \alpha_t \mathbf{e}_0 + \sqrt{\kappa^2 \alpha_t}\epsilon_t,\ \epsilon_t \sim \mathcal{N}(0, I), \mathbf{e}_0 = \mathbf{y} - \mathbf{x}_0$$

Let's consider step $\Delta t = \frac{1}{T}$ and continuous functions $\alpha(t) = T\alpha_{t \cdot T}$, $\mathbf{x}(t) = \mathbf{x}_{t \cdot T}$, $\epsilon(\mathbf{t}) = \epsilon_{\mathbf{t \cdot T}}$
Then we can rewrite the equation:

$$\mathbf{x}(t + \Delta t) - \mathbf{x}(t) = \alpha(t)\mathbf{e}_0 \Delta t + \sqrt{\kappa^2 \alpha(t)}\sqrt{\Delta t}\epsilon(t)$$

By going at the limit $\Delta t \to 0$ we get SDE:

$$d\mathbf{x} = \alpha(t)\mathbf{e}_0 dt + \sqrt{\kappa^2 \alpha(t)} d\mathbf{w}$$

It is proven [SSDK$^+$21] that for a forward SDE of form

$$dx = f(x, t)dt + g(x, t)dw$$

one can write a corresponding reverse ODE:

$$dx = f(x, t)dt - \frac{1}{2} g(x, t)\nabla_x \log p(x_t)dt$$

For our case we will have the following ODE:

$$d\mathbf{x} = \left(\alpha(t)\mathbf{e}_0 - \frac{1}{2}\kappa^2 \alpha(t)\nabla_{\mathbf{x}} \log p(\mathbf{x}|\mathbf{y})\right) dt \tag{1}$$

2.4 Score approximation

For solving our ODE the only term we do not know is the score function $\nabla_{\mathbf{x}} \log p(\mathbf{x}|\mathbf{y})$.
For this goal we learn an estimation on it by training a neural network $s_\theta(\mathbf{x_t}, \mathbf{y})$ with loss:

$$Loss = \mathbb{E}_{q(\mathbf{x_t}|\mathbf{y})} ||s_\theta(\mathbf{x_t}, \mathbf{y}) - \nabla_{\mathbf{x}(t)} \log p(\mathbf{x}|\mathbf{y})||_2^2$$

It can be shown that loss can be rewritten in a form:

$$Loss = \mathbb{E}_{q(\mathbf{x_0})q(\mathbf{x_t}|\mathbf{x_0},y)} ||s_\theta(x_t, y) - \nabla_{\mathbf{x}} \log p(\mathbf{x}|\mathbf{x_0}, \mathbf{y})||_2^2 + const(\theta)$$

Because we know the analytical form of $p(\mathbf{x_t}|\mathbf{x_0}, \mathbf{y}) = \mathcal{N}(\mathbf{x_t}|\mathbf{x_0} + \eta_t \mathbf{e_0}|\kappa^2 \alpha_t I)$, we can substitute it into the score and get expression for score function:

$$\nabla_{\mathbf{x}} \log p(\mathbf{x}|\mathbf{x_0}, \mathbf{y}) = \nabla_{\mathbf{x}} \log(\mathcal{N}(\mathbf{x_t}|\mathbf{x_0} + \eta_t \mathbf{e_0}|\kappa^2 \alpha_t I)) = -\frac{\mathbf{x_t} - (\mathbf{x_0} + \eta_t \mathbf{e_0})}{\kappa^2 \alpha_t}$$

Loss of the original ResShift model can be used to obtain the estimation $s_\theta(\mathbf{x_t}, \mathbf{y})$

$$\begin{aligned}\mathbb{E}_{q(\mathbf{x_0})q(\mathbf{x_t}|\mathbf{x_0},\mathbf{y})} ||f_\theta(\mathbf{x_t}, \mathbf{y}) - \mathbf{x_0}||_2^2 &= (\kappa^2 \alpha_t)^2 || - (\frac{x_t - (f_\theta(x_t, y) + \eta_t e_0)}{\kappa^2 \alpha_t}) - -\frac{x_t - (x_0 + \eta_t e_0)}{\kappa^2 \alpha_t}||_2^2 \\ &= (\kappa^2 \alpha_t)^2 || - \left(\frac{\mathbf{x_t} - (f_\theta(\mathbf{x_t}, \mathbf{y}) + \eta_t \mathbf{e_0})}{\kappa^2 \alpha_t}\right) - \nabla_{\mathbf{x}} \log p(\mathbf{x}|\mathbf{x_0}, \mathbf{y})||_2^2.\end{aligned}$$

So we have the following connections between two considered parametrizations.

$$s_\theta(\mathbf{x_t}, \mathbf{y}) = -\frac{\mathbf{x_t} - f_\theta(\mathbf{x_t}, \mathbf{y})(1 - \eta_t) - \eta_t \mathbf{y}}{\kappa^2 \alpha_t}.$$

2.5 Noise schedule

The function $\alpha(t)$ for the equation 1 is obtained by fitting the suggested sequence

$$\alpha_t,\ t = 1, \ldots, T$$

with a function of the form

$$\alpha(t) = a \cdot (e^{\lambda \cdot t} - 1), \tag{2}$$

where the parameters are found to be:

$$a = 1.111, \lambda = 0.489.$$

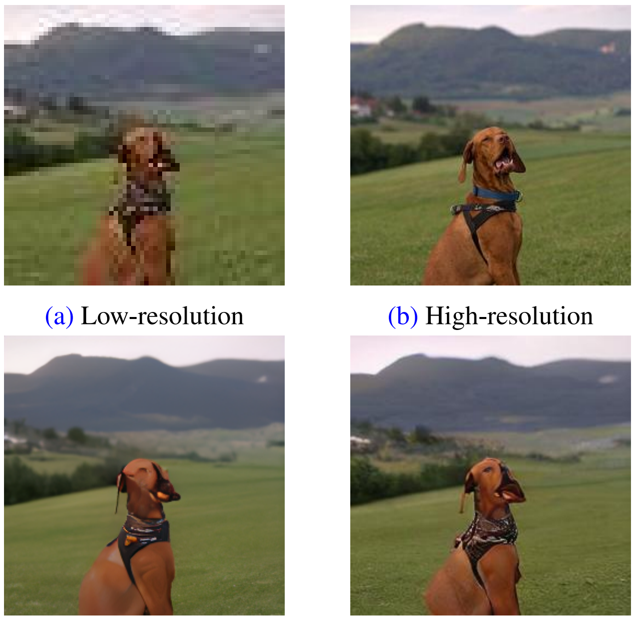
This step is performed to preserve the marginal distribution

$$q(\mathbf{x}_t | \mathbf{x}_0, \mathbf{y}) = \mathcal{N}\left(\mathbf{x}_t; \mathbf{x}_0 + \eta_t \mathbf{e}_0, \kappa^2 \eta_t \mathbf{I}\right).$$

3 Practical part

3.1 Training

Example of how our model works is presented on the Figure 1.



(a) Low-resolution (b) High-resolution

(c) Original result (d) Our result

Figure 1: Comparison of our model and model from the original paper

3.1.1 Dataset

For fine-tuning we follow training setup from the original paper: high-resolution images are taken from Imagenet-256 train dataset and low-resolution images are their x4 downscales with realsrgan augmentations.

3.1.2 Training details

During training, we use a fixed learning rate of 5e-5 and update the weight parameters for 50K iterations. We choose the batch size of 16 and microbatch size of 8. Adam optimizer were chosen with the default PyTorch settings. As for the network architecture, we employ UNet architecture in DDPM. Following the ResShift approach self-attention layer in UNet was replaced by the Swin Transformer block.

During training timesteps t were uniformly sampled from the $(\frac{1}{T}, 1)$ interval with $T = 15$. They were used to calculate α_t with respect to the formula 2.

Before feeding to neural network the timesteps were rescaled to be uniformly distributed in the interval of (0, T).

Loss is the same as in the original paper:

$$Loss = \mathbb{E}_{q(\mathbf{x_0})q(\mathbf{x_t}|\mathbf{x_0},\mathbf{y})}||f_\theta(\mathbf{x_t},\mathbf{y}) - \mathbf{x_0}||_2^2.$$

3.2 Experiments

For testing we used 1000 high-resolution images from Imagenet-256 test dataset getting low-resolution images using x4 bicubic downscale.

We fixed the number of models evaluation and vary different ODE solver hyperparameters:

- ODE solver: Euler/Heun;
- Different ρ values for different time discretizations corresponding to the following formula

  $$t_i = \left(1 + \frac{i}{N-1}\left((1/T)^{(1/\rho)} - 1\right)\right)^\rho,$$

  where $N - 1$ is the number of steps of ODE solver.
  If $\rho < 0$, we use

  $$t_{N-1-i} = 1 - \left(1 + \frac{i}{N-1}\left((1/T)^{(1/\rho)} - 1\right)\right)^\rho$$

  The idea is the following: the higher ρ the more steps are taken closer to the image ($pho = 1$: timesteps distributed evenly).

Results are performed in the Table 1. From that table we can see that our approach is superior to the one proposed in the original paper (it is also notable visually in Figure 1). Also it can be concluded that Heun solver, which takes less discretization steps (since it does 2 model evulation per step), gets more benefits by taking those steps "closer" to the noised loweres.

solver	rho	LPIPS $\downarrow$	SSIM $\uparrow$	PSNR $\uparrow$
Baseline		0.376	0.428	20.36
Euler	1	0.364	0.457	20.29
Euler	4	0.363	0.451	20.45
Euler	8	0.362	0.449	**20.50**
Euler	-4	0.362	0.453	20.33
Euler	-8	**0.361**	0.453	20.35
Heun	1	0.375	**0.473**	19.93
Heun	4	0.374	0.472	19.53
Heun	8	0.374	0.472	19.96
Heun	-4	0.365	0.462	20.10
Heun	-8	0.364	0.459	20.13

Table 1: Results for nfe=15 and various ODE solvers and discretizations.

4 Conclusion

The results of this work are the following:

- We are derived all necessary math to work with ResShift described as SDE.
- We showed that it works in practice.
- We showed that such a setup works better then original discrete-in-time setup.
- We showed in case of low number of discretization steps it's more preferrable to take most of them "closer to noised image".

5 Contributions

Pavel Bartenev: Adapting training pipeline to the case of continuous steps. Implementing the function of calculating diffusion noise multipliers in continuous case and integrating it into the code.

Daniil Shlenskii: Managing the team, theory formulas derivation, writing code for inference.

Arina Chumachenko: Working on model training.

Julia Sergeeva Writing solver of reverse ODE, working on inference

Alex Fokin Writing solver of reverse ODE, working on inference

References

[JH20] Pieter Abbeel Jonathan Ho, Ajay Jain. Denoising diffusion probabilistic models, 2020.

[LZB$^+$22] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, and Jun Zhu Chongxuan Li. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps, 2022.

[RBLPE21] Robin Rombach, Andreas Blattmann, Dominik Lorenz, and Björn Ommer Patrick Esser. High-resolution image synthesis with latent diffusion models, 2021.

[SSDK$^+$21] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations, 2021.

[YGL$^+$24] Fanghua Yu, Jinjin Gu, Zheyuan Li, Jinfan Hu, Xiangtao Kong, Xintao Wang, Jingwen He, Yu Qiao, and Chao Dong. Scaling up to excellence: Practicing model scaling for photo-realistic image restoration in the wild, 2024.

[YWL23] Zongsheng Yue, Jianyi Wang, and Chen Change Loy. Resshift: Efficient diffusion model for image super-resolution by residual shifting, 2023.