Generative Denoising for Medical Image Reconstruction

Using Score-Based Diffusion Models

Siddhant Chaurasia

Table of contents

Motivation & Background

Clinical Challenge in CT Imaging

Key challenges in CT imaging:

- Radiation dose concerns; trade-off between image quality and radiation exposure
- Need for fast acquisition to reduce motion artifacts

Sparse-view CT as a solution:

- Reduces radiation dose
- Faster acquisition times
- But creates undersampled data → reconstruction problem

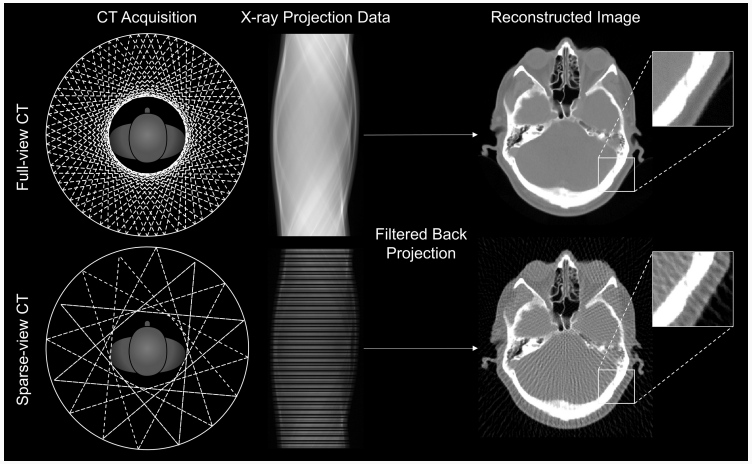


Figure 1: Sparse views CT (Adapted from Chung2022[1]).

Research Goal

Develop advanced reconstruction algorithms to produce high-quality images from limited data

CT Reconstruction

Forward Model:

$$y = \mathcal{A}x + \eta$$

Where:

y: Measured scanner data (sinogram).

$\mathcal{A}$: System model (Radon transform, describes scanner physics).

x: The true underlying image we want to find.

η: Measurement noise and model inaccuracies.

Inverse Problem:

$$\hat{x} = \underset{x}{\operatorname{argmin}} \quad \underbrace{\|\mathcal{A}x - y\|^2}_{\substack{\text{Data Fidelity} \\ \text{(Match measurements)}}} \quad + \quad \underbrace{\lambda R(x)}_{\substack{\text{Regularization} \\ \text{(Incorporate prior knowledge)}}}$$

Goal: Find the image estimate $\hat{x}$ that best fits the measurements while also incorporating prior knowledge or assumptions about the image via the regularizer $R(x)$. The parameter λ controls the trade-off.

Diffusion Models

Diffusion Models: Core Idea

Generate data by learning to reverse a noise-adding process.

How it Works:

- **Forward:** Add noise gradually. (Fixed)
- **Reverse:** Learn to remove noise step by step. (Generative)

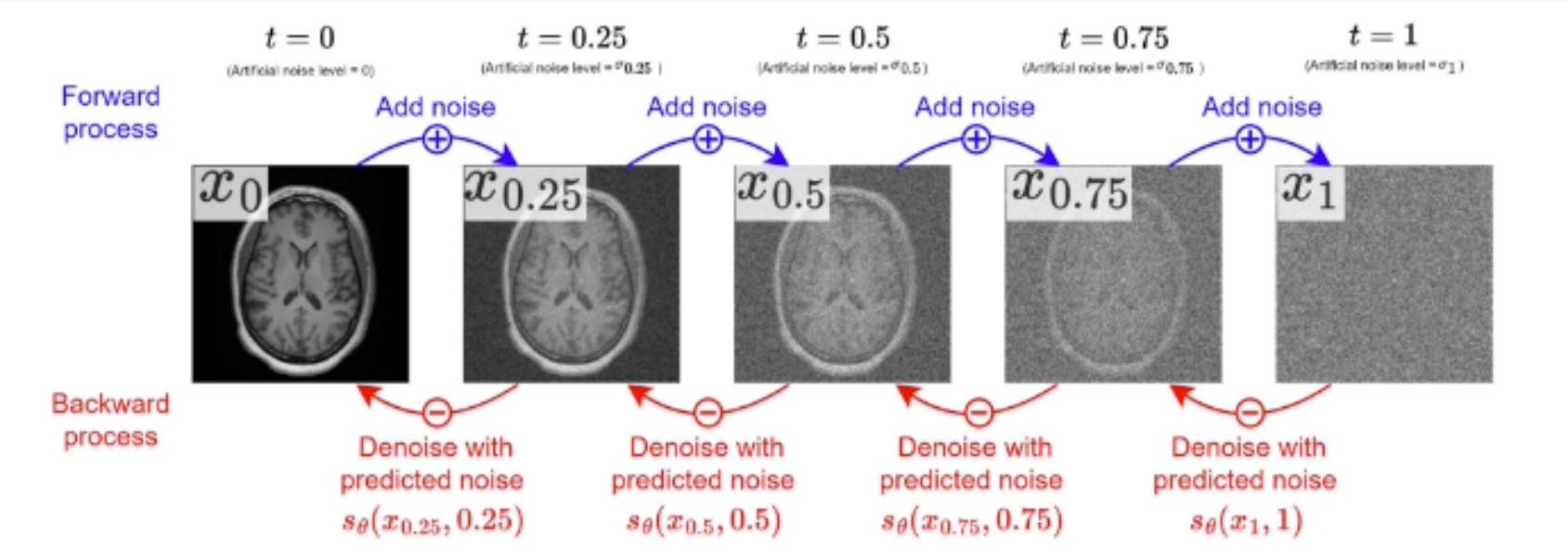


Figure 2: Diffusion Process (Adapted from Reader24) [5].

Task & Dataset

Generative Denoising of CT Patches

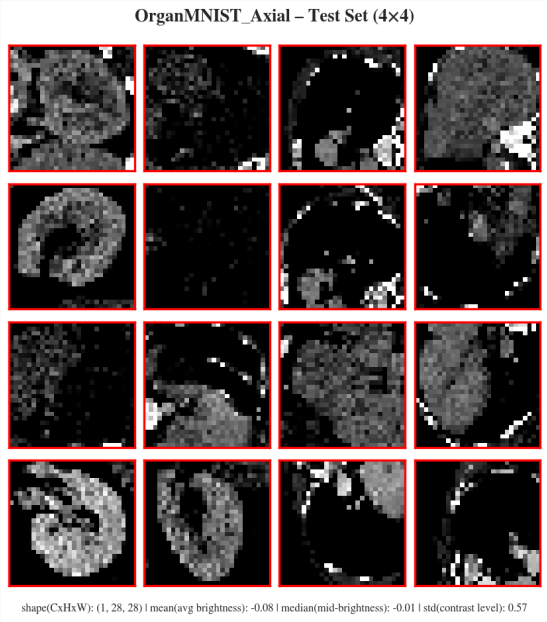
Objective: Reconstruct clean 28x28 CT image patches ($\boldsymbol{x}$) from noisy versions ($\boldsymbol{y}$), where noise is simulated as additive white Gaussian noise ($\boldsymbol{y} = \boldsymbol{x} + \mathcal{N}(0, \sigma_n^2 \mathbf{I})$).

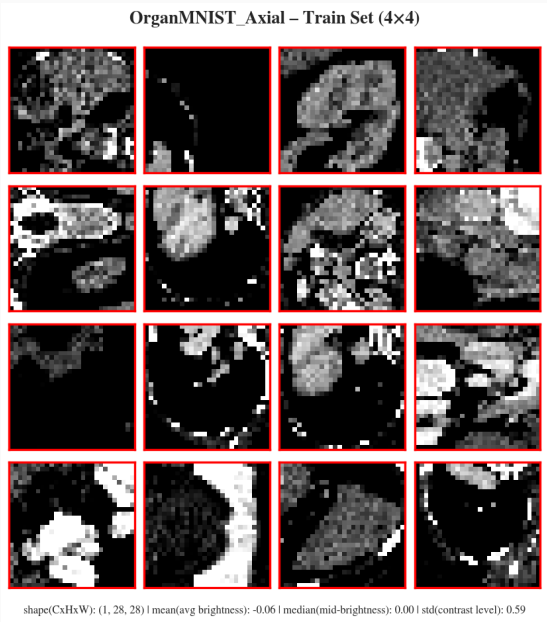
Challenge: Effectively remove significant noise ($\sigma_n = 0.3$ in [-1, 1] space) while preserving fine anatomical details within small image patches.

My Approach Utilize a Score-Based Diffusion Model, conditioned on the noisy input image $\boldsymbol{y}$, to perform generative denoising via a learned reverse process.

Dataset: OrganAMNIST (from MedMNIST)

- Publicly available MedMNIST [6]
- Derived from the Liver Tumor Segmentation Benchmark (LiTS) CT scans.
- Content: 2D axial CT patches (28×28 grayscale) centered on 11 organs (labels discarded).
- Preprocessing: HU values windowed using an abdominal window; images normalized to $[-1, 1]$.





Stochastic Differential Equations in Diffusion Models

VP-SDE: Mathematical Formulation

Forward Process (Data $\rightarrow$ Noise)

- Continuous SDE ($t \in [0, 1]$):

$$d\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}dt + \sqrt{\beta(t)}d\mathbf{w}$$

- Gaussian Marginal:

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

- Direct Sampling Formula:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\mathbf{z}$$

Key Terms:

- $\beta(t)$: Noise schedule
- $\bar{\alpha}_t$: Cumulative product $\prod_{i=1}^{t}(1 - \beta_i)$
- $\mathbf{w}$: Wiener process
- $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

Key Insight

VP-SDE gradually transforms data into noise with controlled variance, enabling flexible sampling.

VP-SDE: Implementation

Cosine Noise Schedule:

```
s = 0.008
steps = torch.linspace(0, n_steps, n_steps + 1)

f_t = torch.cos(((steps / n_steps) + s) /
       (1 + s) * math.pi / 2) ** 2
alpha_bar = f_t / f_t[0]

betas = (1 - (alpha_bar[1:] / alpha_bar[:-1]))
       .clamp(0.0001, 0.9999)

sqrt_alpha_bar = torch.sqrt(alpha_bar[1:])
sqrt_one_minus_alpha_bar = torch.sqrt(
    1 - alpha_bar[1:])
```

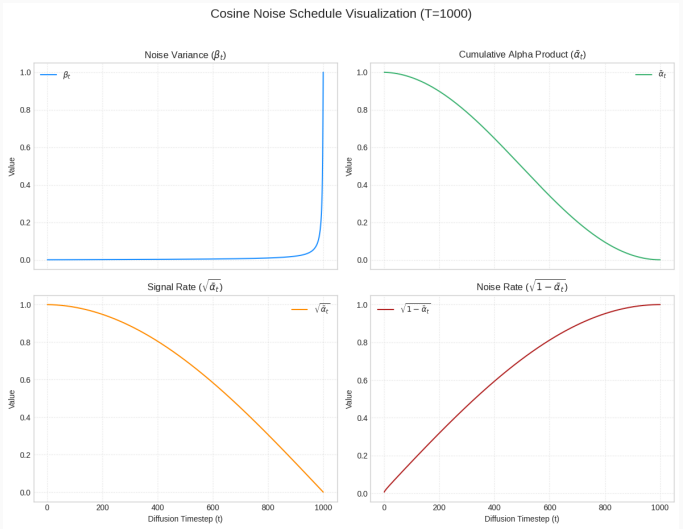


Figure 3: Visualization of cosine noise schedule

Time Indexing:

```
index = (t * (n_steps - 1))
    .long()
    .clamp(0, n_steps - 1)
```

Forward Process: SDE Implementation

Forward Process Functions:

```python
def marginal_prob(self, x0, t):
    """Get mean & std of q(x_t|x_0)"""
    alpha_t = self.alpha(t).view(-1, 1, 1, 1)
    sigma_t = self.sigma(t)

    mean = alpha_t * x0
    std = sigma_t

    return mean, std

def forward(self, x0, t):
    """Add noise to data: sample from q(x_t|x_0)"""
    mean, std = self.marginal_prob(x0, t)

    z = torch.randn_like(x0)

    x_t = mean + std.view(-1, 1, 1, 1) * z

    return x_t, z
```

Key Operations:

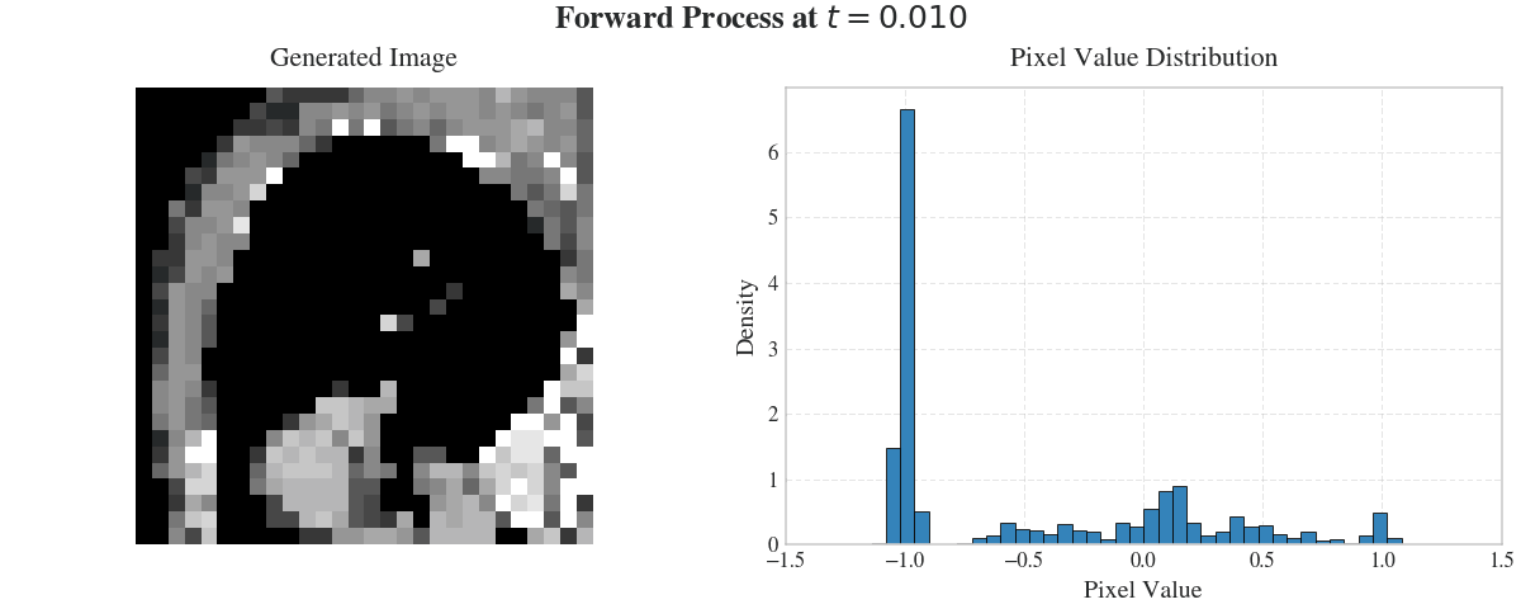
- Get parameters: $\sqrt{\bar{\alpha}_t}$ and $\sqrt{1 - \bar{\alpha}_t}$
- Calculate mean: $\mu_t = \alpha_t \cdot \mathbf{x}_0$
- Sample noise: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
- Compute noisy image:

$$\mathbf{x}_t = \mu_t + \sigma_t \cdot \mathbf{z}$$

Implementation Notes

- Forward process enables direct sampling at any timestep t
- Variance parameters are pre-computed for efficiency
- Function returns both $\mathbf{x}_t$ and the noise $\mathbf{z}$ (for training)
- Shape management critical for batched operations

Forward Process Visualization



Forward Process: Clean Image → Pure Noise

- Shows gradual transformation from clean CT image to pure Gaussian noise
- Controlled by noise schedule $\beta(t)$ as t increases from 0 to 1
- Training objective: Learn to reverse this process by predicting noise $\mathbf{z}$

cont.

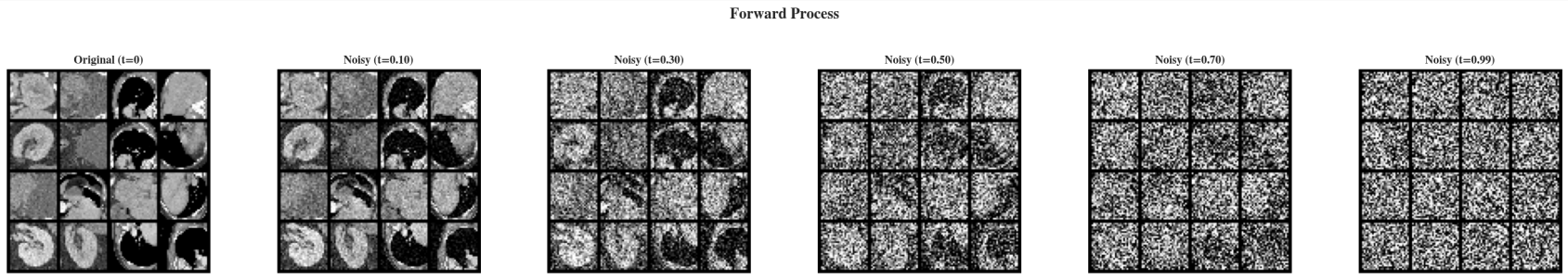


Figure 4: Forward Process

Model Architecture

Architecture Overview

```
--- Model Architecture ---
Unet(
  (time_mlp): Sequential(
    (0): Linear(in_features=1, out_features=384, bias=True)
    (1): SiLU()
    (2): Linear(in_features=384, out_features=96, bias=True)
  )
  (conv_in): Conv2d(1, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (down1): Conv2d(96, 192, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (down2): Conv2d(192, 384, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (mid1): Conv2d(384, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (mid_attn): SelfAttention(
    (norm): GroupNorm(12, 384, eps=1e-05, affine=True)
    (to_qkv): Conv2d(384, 1152, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (to_out): Conv2d(384, 384, kernel_size=(1, 1), stride=(1, 1))
  )
  (mid_t): Linear(in_features=96, out_features=384, bias=True)
  (mid2): Conv2d(384, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (up1): ConvTranspose2d(384, 192, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (skip_conv1): Conv2d(192, 192, kernel_size=(1, 1), stride=(1, 1))
  (up2): ConvTranspose2d(384, 96, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (skip_conv_in): Conv2d(96, 96, kernel_size=(1, 1), stride=(1, 1))
  (conv_out): Conv2d(192, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
)
------------------------------------------------
Total Parameters: 6,614,881
------------------------------------------------
```

Figure 5: U-Net with attention

- **Input:** Noisy image $\mathbf{x}_t$ & timestep t
- **Encoder:** 3 levels (96 → 192 → 384 channels)
- **Bottleneck:** Self-attention + time injection
- **Decoder:** Upsample + skip-connections
- **Output:** Predicted noise $\mathbf{z}$ (same size)

Stats: 6.6 M parameters, for 28×28 CT patches

Time Conditioning & Self-Attention

Time Conditioning:

Embed continuous timestep *t* into a feature vector and inject at the bottleneck.

```
self.time_mlp = nn.Sequential(
  nn.Linear(1, hidden_dim*4),
  nn.SiLU(),
  nn.Linear(hidden_dim*4, hidden_dim)
)

h += self.mid_t(t_emb).view(-1, hidden_dim, 1, 1)
```

Self-Attention Block :

Multi-head attention lets each pixel gather context from all others.

```
class SelfAttention(nn.Module):
    def __init__(self, channels, num_heads=8):
        self.scale = (channels//num_heads)**-0.5
        self.to_qkv = nn.Conv2d(channels, channels*3,
       1, bias=False)
        self.to_out = nn.Conv2d(channels, channels,
      1)

    def forward(self, x):
        B,C,H,W = x.shape
        qkv = self.to_qkv(x).chunk(3, dim=1)
        q,k,v = [t.view(B, -1, H*W)
                    .reshape(B, h, d, H*W)
                    .permute(0,1,3,2)
                 for t,(h,d) in zip(qkv, [(8,C//8)
      ]*3)]
        attn = torch.softmax((q @ k.transpose(-1,-2))
      *self.scale, -1)
        out = (attn @ v).permute(0,1,3,2).reshape(B,C
      ,H,W)
        return self.to_out(out) + x
```

Training

Loss Function

1. Noise Prediction Loss (MSE):

$$\mathcal{L}_{\text{noise}} = \mathbb{E}\left[\|\mathsf{s}_\theta(\mathsf{x}_t, t) - \mathsf{z}\|^2\right]$$

2. Frequency-Domain Loss (L1):

$$\mathcal{L}_{\text{freq}} = \mathbb{E}\left[\| \, |\mathcal{F}(\hat{\mathsf{x}}_0)| - |\mathcal{F}(\mathsf{x}_0)| \|_1\right]$$

$\mathcal{F}$=2D FFT, $|\cdot|$=magnitude

3. Combined Objective:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{noise}} + \lambda \, \mathcal{L}_{\text{freq}} \quad (\lambda = 0.1)$$

x_t*: noisy image at time t;* x_0*: ground-truth clean image;* $\hat{\mathsf{x}}_0$*: reconstructed image;* z*: sampled Gaussian noise;* $\mathsf{s}_\theta(\cdot)$*: score network (predicted noise);* λ*: frequency-loss weight*

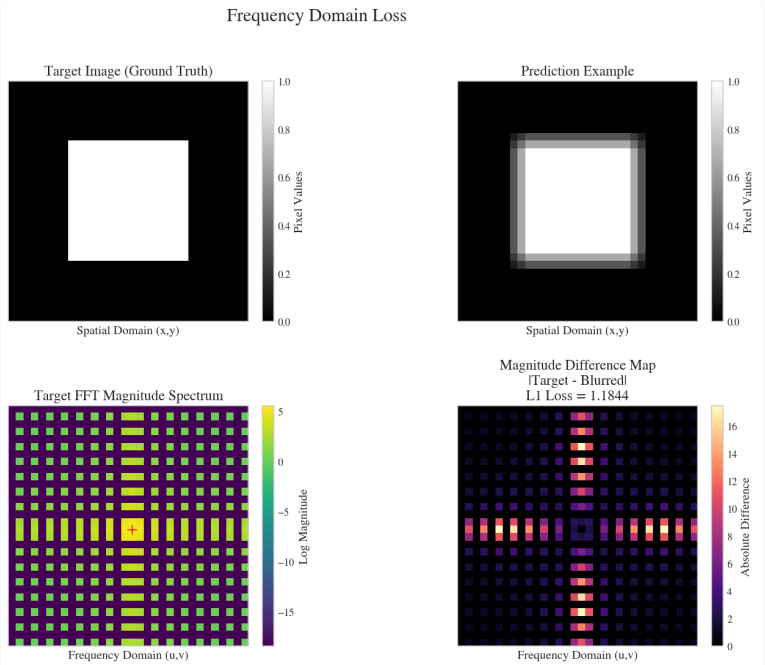


Figure 6: FFT Domain Loss example

Implementation of Loss Components

```
def diffusion_loss(model, sde, x0, lambda_freq=0.1)
      :
{
    t = torch.rand(x0.size(0), device=x0.device)
    x_t, z = sde.forward(x0, t)

    z_pred = model(x_t, t)
    loss_noise = F.mse_loss(z_pred, z)

    alpha = sde.alpha(t).view(-1,1,1,1)
    sigma = sde.sigma(t).view(-1,1,1,1)
    x0_pred = (x_t - sigma*z_pred) / alpha
    loss_freq = frequency_domain_loss(x0_pred, x0)

    total = loss_noise + lambda_freq * loss_freq
    return total, loss_noise, loss_freq
}
```

```
def frequency_domain_loss(pred_x0, target_x0, norm=
      'l1'):
{
    pred_x0 = pred_x0.float()
    target_x0 = target_x0.float()

    pred_fft = torch.fft.fft2(pred_x0, dim=(-2,-1))
    target_fft = torch.fft.fft2(target_x0, dim
      =(-2,-1))

    pred_mag = torch.abs(pred_fft)
    target_mag = torch.abs(target_fft)

    loss = F.l1_loss(pred_mag, target_mag)
    return loss
}
```

Why add a frequency loss?

- Aims to preserve high-frequency details often lost with pure MSE.
- Aligns spectral content between prediction & target images.
- Can potentially improve perceptual quality and structure preservation.

Optimization & Training Progress

Optimizer & Hyperparameters

- **Optimizer:** AdamW (wd=1e-4)
- **LR:** 5e-5 w/ CosineAnnealingWarmRestarts
- **Batch size:** 64
- **Epochs:** 100
- **Loss Fn:** MSE(z_{pred}, z) + 0.1 × L1(FFT Mag)
- **Grad clip:** max-norm 1.0

Final Loss Values (Epoch 100):

- **Train Total:** 0.51
- **Validation Total:** 0.77

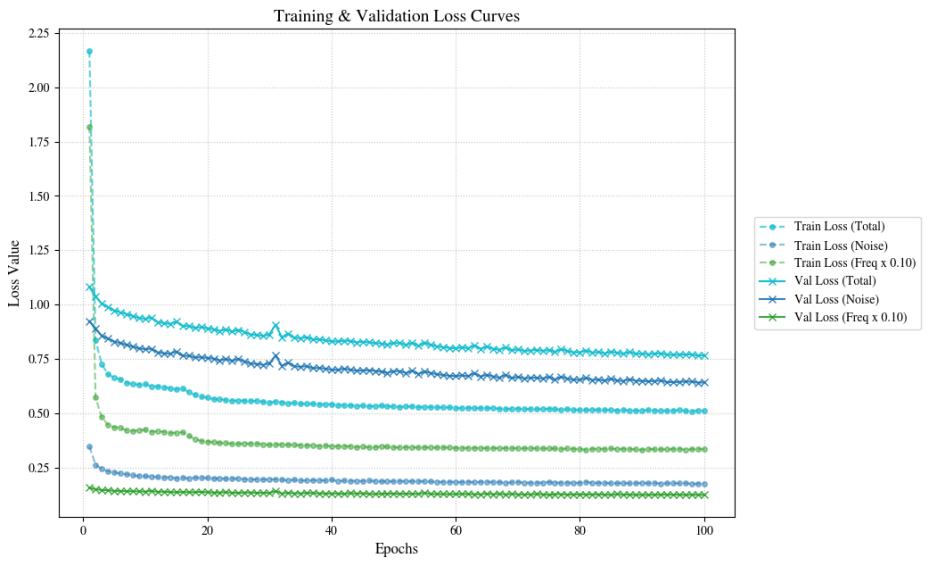


Figure 7: Training & validation loss (Total) over epochs.

Inference

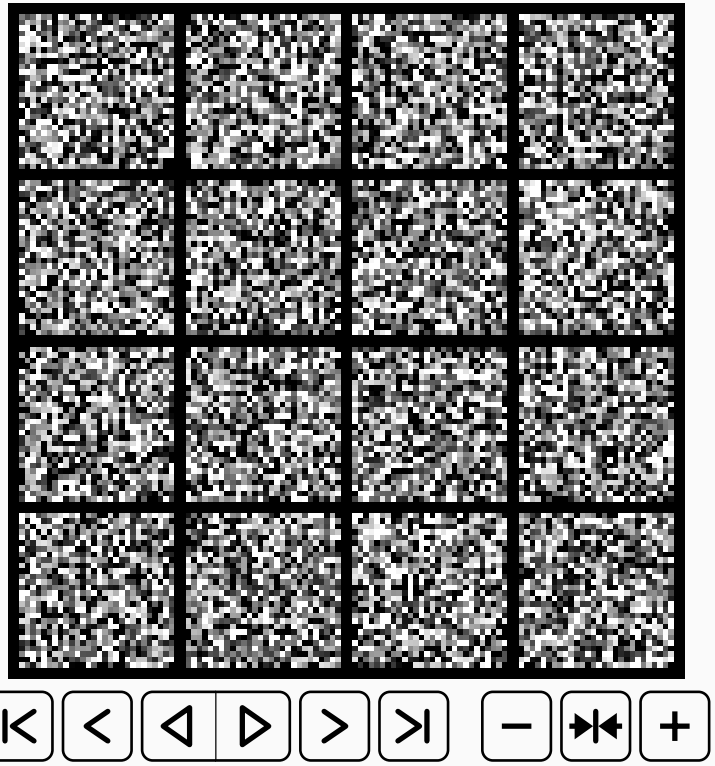
Generative Denoising Algorithm

```
@torch.no_grad()
def generative_denoising(y_noisy, model, sde,
                         sigma_n, n_steps, eps=1e-5):
    x = torch.randn_like(y_noisy)
    time_steps = torch.linspace(1.0, eps, n_steps+1)

    for i in range(n_steps):
        t_curr, t_next = time_steps[i], time_steps[i+1]
        t_vec = torch.full((x.size(0),), t_curr, device=
      x.device)

        σ_t = sde.sigma(t_vec).view(-1,1,1,1)
        λ_t = σ_t**2 / σ(_t**2 + sigma_n**2)
        x = λ(1-_t)*x + λ_t*y_noisy

        z_pred = model(x, t_vec)
        α_t = sde.alpha(t_vec).view(-1,1,1,1)
        x0_pred = (x - σ_t*z_pred) / α_t
        x0_pred.clamp_(-1,1)

        mean, std = sde.marginal_prob(x0_pred, t_next)
        z = torch.randn_like(x) if i < n_steps-1 else 0
        x = mean + std.view(-1,1,1,1)*z

    return x0_pred
```

Key Steps:

- **Initialization:** $x_T \sim \mathcal{N}(0, I)$ as starting noise.
- **Measurement Conditioning:** $\lambda_t = \frac{\sigma_t^2}{\sigma_t^2+\sigma_n^2}$, then $x_t \leftarrow (1-\lambda_t)x_t + \lambda_t y$.
- **Score Estimation:** Predict noise with $\mathbf{s}_\theta(x_t, t) = z_{\text{pred}}$.
- **Clean Estimate (Tweedie):** $\hat{x}_0 = (x_t - \sigma_t z_{\text{pred}})/\alpha_t$.
- **Reverse SDE Sampling:** Draw $x_{t-\Delta t} \sim \mathcal{N}(\mu, \sigma^2)$ from the learned posterior.

Reverse Process Visualization



Reverse Process: Noise → Clean Image

- Shows the generative denoising process from random noise to clean reconstruction
- Algorithm iteratively refines estimate while maintaining consistency with noisy input
- Uses 1000 denoising steps to balance detail recovery and stability

cont.

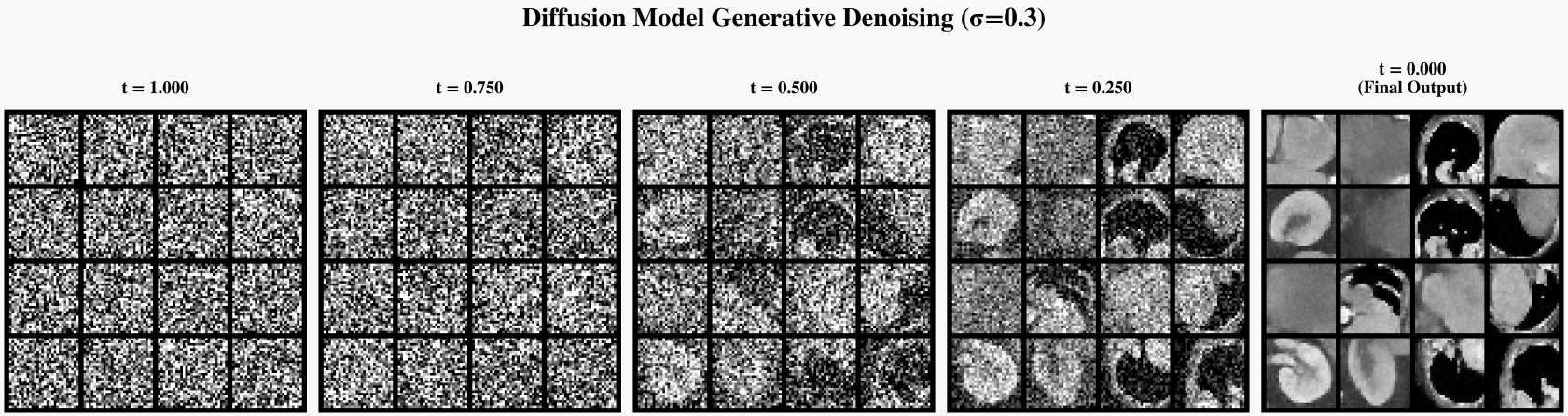


Figure 8: Reverse Process

Results

Results

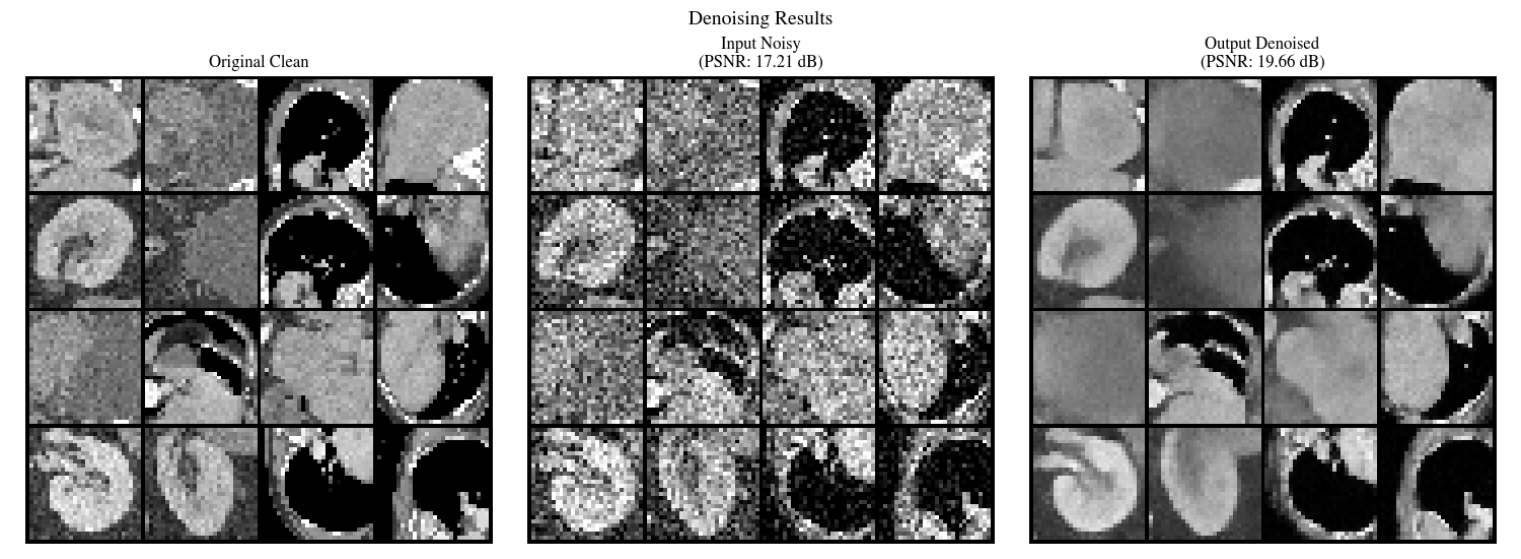


Figure 9: Comparison: Ground Truth (left), Noisy Input ($\sigma_n = 0.3$, middle), Denoised Output (right)

Evaluation Setup

- **Test Data:** 16 OrganAMNIST patches (unseen during training).
- **Input Noise:** Additive White Gaussian Noise (AWGN) with $\sigma_n = 0.3$.
- **Denoising:** $N = 1000$ reverse diffusion steps.

Quantitative Results & Key Findings

Quantitative Metrics

Image	PSNR (dB) ↑	SSIM ↑
Noisy Input	17.21	0.5372
Denoised Output	19.66	0.5364
Improvement	**+2.44**	**–0.0008**

Key Findings

- **+2.44 dB PSNR gain** demonstrates the model how to effectively remove Gaussian noise.
- **A**natomical details preserved despite minor SSIM change.
- **Fine-detail recovery** (e.g. vessel edges, tissue boundaries) is improved by the frequency-domain loss term.
- **Conditional generative denoising** yields diverse, realistic reconstructions based on provided noisy input

Conclusions & Future Work

Conclusions & Future Work

Conclusions

- Successfully applied conditional diffusion for denoising medical patches.
- Achieved significant PSNR gain (+2.44 dB); details preserved visually.
- Demonstrated potential for generative models in low-data medical imaging.

Future Work

- **Physics-Informed:** Incorporate forward model ($\mathcal{A}$) for inverse problems [4]; explore diffusion directly on measurement data (e.g., sinograms) [3].
- **Scale & Efficiency:** Extend models to full 2D slices / 3D volumes; investigate faster sampling (e.g., ODE solvers).
- **Hybrid Methods:** Combine generative prior with classical data-fidelity terms (inspired by ML/MAP [2]).
- **Clinical Validation:** Evaluate on realistic phantom/patient data; compare with clinical benchmarks & MGH prototypes.

References

[1] Kevin J. Chung et al. **“Low-dose CT Perfusion with Sparse-view Filtered Back Projection in Acute Ischemic Stroke”.** In: *Academic Radiology* 29.10 (2022), pp. 1502–1511. DOI: 10.1016/j.acra.2022.01.018. URL: https://doi.org/10.1016/j.acra.2022.01.018.

[2] L. A. Shepp and Y. Vardi. **“Maximum Likelihood Reconstruction for Emission Tomography”.** In: *IEEE Transactions on Medical Imaging* 1.2 (1982), pp. 113–122. DOI: 10.1109/TMI.1982.4307558.

[3] Yang Song et al. ***Score-Based Generative Modeling through Stochastic Differential Equations.*** 2021. arXiv: 2011.13456 [cs.LG]. URL: https://arxiv.org/abs/2011.13456.

References ii

[4] Yang Song et al. ***Solving Inverse Problems in Medical Imaging with Score-Based Generative Models.*** 2022. arXiv: 2111.08005 [eess.IV]. URL: https://arxiv.org/abs/2111.08005.

[5] George Webber and Andrew Reader. **"Diffusion models for medical image reconstruction".** English. In: *British Journal of Radiology | Artificial Intelligence* 1.1 (Aug. 2024). ISSN: 2976-8705. DOI: 10.1093/bjrai/ubae013.

[6] J. Yang, R. Shi, D. Wei, et al. **"MedMNIST v2 - A large-scale lightweight benchmark for 2D and 3D biomedical image classification".** In: *Scientific Data* 10.41 (2023). Received 29 November 2021, Accepted 26 September 2022, Published 19 January 2023. DOI: 10.1038/s41597-022-01721-8. URL: https://doi.org/10.1038/s41597-022-01721-8.

Thank You & Questions

Thank you!

I’m happy to take any questions!