Analysis of Modern Diffusion Techniques for Image Reconstruction

Stephen Zhu

Abstract—Machine learning models have been helpful in the task of solving inverse problems, as they can often learn to estimate inverses from data with varying noise levels and sometimes unknown forward functions. One type of model that has shown promise in recent years is diffusion models. This project focuses on using a pre-trained diffusion model, along with three inverse problem algorithms, to attempt image reconstruction.

Index Terms—Computational Photography, Diffusion Models, Signal Processing

1 Introduction

INVERSE problems in image processing can essentially be defined as follows: given an incomplete image, how can we obtain the original image? When we take a picture, there is always some form of noise that appears in the images that is usually modeled as Gaussian or Poisson noise [1]. Occlusion or blurring are also factors that can cause distortion of the raw image. The main issue concerning these inverse problems is that they are typically ill-posed.

This stems from a number of reasons, but there are two primary ones. The first is that we cannot model noise precisely, due to not knowing the exact distribution of noise. Thus, it is impossible to completely remove noise from an image. The second is that with occlusion or blurring, there are multiple possible raw images that could have resulted in the measured image. Without perfect knowledge of the original input and the forward model (to produce the measured image), it is impossible to create a perfect reconstruction.

There are many different methods that have been investigated in the past to help mitigate these issues. For handling noise within an already captured image, bilateral filtering provides a method of localized denoising [2], and non-local means utilizes non-local patches of similar intensity to aid in denoising [1]. For handling blur in images, extended depth of field helps remove bokeh in still images [3], whereas flutter shutter is a method for removing motion blur [4]. Many other such examples exist of algorithms that can handle these and similar tasks.

More recently, avenues involving machine learning models have proven quite successful. Some models have been trained to denoise [5], [6], while others can handle deblurring [7]. One model was even trained to handle low-light images and reconstruct a better lit image from it [8].

In particular, diffusion models are another type of machine learning model that is actively being researched for solving inverse problems. This project focuses on using a provided pretrained diffusion model to estimate noise in input (”measured”) images, which can then be used in sampling steps to eventually output an estimated clean image.

• *S. Zhu is with the Department of Electrical Engineering, Stanford University, Stanford, CA 94305.*
E-mail: srzhu3@stanford.edu

2 Related Work

To begin, we explain how diffusion models are used in this project. We begin with a measured image x_0. We iteratively add noise to the image in a forward noising process, which can be simplified to a closed form solution [9]. The image can be partially noised [10], or completely noised. Then, from the noisy image, we can perform either one step denoising, or multi-step sampling to iteratively denoise an image. In the case of the latter, some reverse diffusion algorithms depend on the posterior to iteratively denoise [10], [11], [12], while others do not [9]. Further details will be discussed in the Proposed Method section.

During both the forward and reverse diffusion process, variance-preserving and variance-exploding formulae can be used [9]. For this project, we primarily focus on the variance-preserving formulae. The forward noise model is given as:

$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}z_{t-1} \tag{1}$$

where t represents the timestep, x_i is the image at timestep i, β is a noise scheduling parameter, and z is a i.i.d. Gaussan random variable $\mathcal{N}(0, I)$. As mentioned before, this can be compressed down into a closed form solution that relies only on the initial clean image:

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}z \tag{2}$$

where

$$\alpha_t = 1-\beta \tag{3}$$

$$\bar{\alpha}_t = \Pi_{i=1}^{t}\alpha_i \tag{4}$$

and z is still an i.i.d Gaussian random variable $\mathcal{N}(0, I)$.

The variance-preserving reverse diffusion algorithm is described by DDPM as follows [9]:

$$\hat{x}_0 = \frac{1}{\sqrt{\bar{\alpha}}}(x_t + (1-\bar{\alpha}_t)s_\theta(x_t, t)) \tag{5}$$

$$x_{t-1} = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t + \frac{\sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)}{1-\bar{\alpha}_t}\hat{x}_0 \tag{6}$$

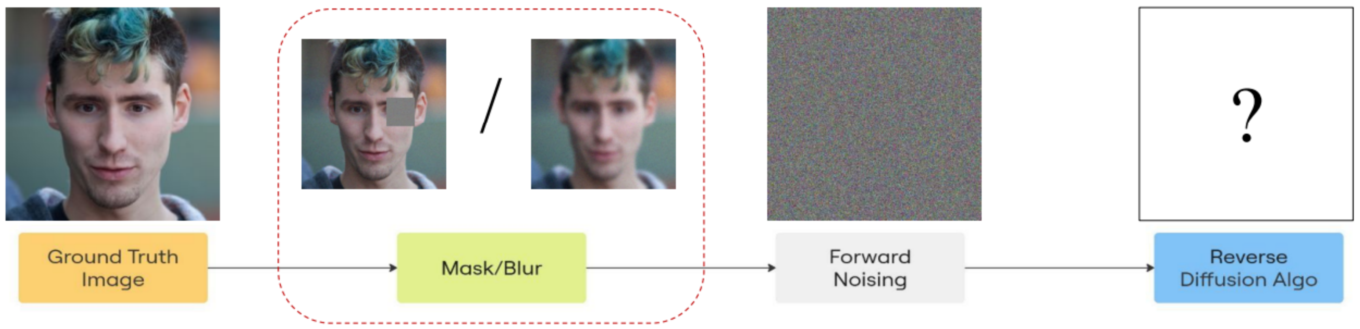

Fig. 1. A flowchart of the proposed methodology. We begin with a fixed ground truth image to better compare each of the different algorithms. This ground truth image is passed through a forward model (either masking or blurring) before adding some noise to produce a ”measured” image y. This process simulates taking a picture in the real world. The measured image is then sent through a forward noising algorithm before being run through each of the different reverse diffusion algorithms.

where $\hat{x}_0$ is the estimated clean image (posterior) and

$$s_\theta(x_t, t) = -\frac{\epsilon_\theta(x_t, t)}{\sqrt{1-\bar{\alpha}_t}} \tag{7}$$

is the score function, which can be calculated from the estimated noise $\epsilon_\theta(x_t, t)$ computed by a denoising neural network.

Score-Distillation Editing (SDEdit) is one of the reverse diffusion algorithms that utilizes posterior sampling. The algorithm utilizes Equations (5) and (6) and a hyperparameter t, which represents how many timesteps out of the total number of steps used to produce the noise schedule β_t are used in the forward and reverse models. Essentially, it describes the level of noise we want to work with. The lower the t, the less noise is added, and fewer sampling steps are done in the reverse diffusion process.

The second reverse diffusion algorithm explored is Score-Based Annealed Langevin Dynamics (ScoreALD). This algorithm requires knowledge of the forward measurement model (A, from $y = Ax + b$, where y is the measured image, x is the original image, and b is (Gaussian) noise) and builds on top of SDEdit. During each reverse diffusion step, Equations (5) and (6) are also used. However, a final step is added:

$$x_{t-1} = x_{t-1} - \frac{1}{\sigma^2 + \gamma_t^2} \nabla_{x_t} ||A(x_t) - y||^2 \tag{8}$$

where σ represents the standard deviation of the measurement noise b and γ_t represents the annealing variable at timestep t. This enables a form of gradient descent to converge towards the maximum likelihood clean image $\hat{x}_0$. This gradient term is annealed to aid in convergence, so as more time steps occur, the gradient term grows smaller over time.

The last reverse diffusion algorithm is Diffusion Posterior Sampling (DPS), which is similar to ScoreALD. The only difference is the last step, where Equation (8) is replaced with:

$$x_{t-1} = x_{t-1} - \frac{\zeta_t}{2\sigma^2} \nabla_{x_t} ||A(\hat{x}_0) - y||^2 \tag{9}$$

$$\zeta_t = \frac{\zeta \times 2\sigma^2}{||\nabla_{x_t} ||A(\hat{x}_0) - y||^2||} \tag{10}$$

where ζ is a scaling hyperparameter and ζ_t is the computed scaling parameter at timestep t. We can see two main differences. The first is that instead of computing the norm of $A(x_t)-y$, we compute the norm of $A(\hat{x}_0)-y$, which relies on the estimated clean image computed using Equation (5). The second difference is that instead of annealing the gradient term over time, the gradient term is normalized and scaled with the same factor for each timestep.

3 Proposed Method

To begin, we will implement a forward diffusion algorithm based on the closed-form solution [9], and test the denoising procedure in a single step. Once we confirm that these processes work, we will implement a posterior sampling loop to iteratively denoise a noisy image. A total of n=1000 timesteps will be used to generate the noise schedule across all tasks.

To compare each of the three reverse diffusion algorithms, we will fix a test image. This ground truth image is either masked or blurred before adding a bit of noise. This output image represents a ”measured” image, which would we obtain in the real world. We will then forward noise the image (stopping at a predetermined timestep in the case of SDEdit), then perform each reverse diffusion algorithm. The overall process is shown in Figure 1

We will use two metrics to compare each of the outputs from the reverse diffusion algorithms against the ground truth image: Peak Signal to Noise Ratio (PSNR) and Learned Perceptual Image Patch Similarity (LPIPS) [13]. The higher the PSNR value, the better, and the lower the LPIPS value, the better. We will run each task 5 times to obtain the average and standard deviation for each metric.

4 Experimental Results

We begin with showing the explicit results of our single step forward and reverse denoising process. We set $t = 300$ to ensure that enough noise is added, while retaining the majority of information from the initial image (see Figure 2). The resulting PSNR is 27.357 ± 0.177, and the resulting LPIPS is 0.199 ± 0.009. The code (implemented in Python 3.13.2) for each function is shown below:

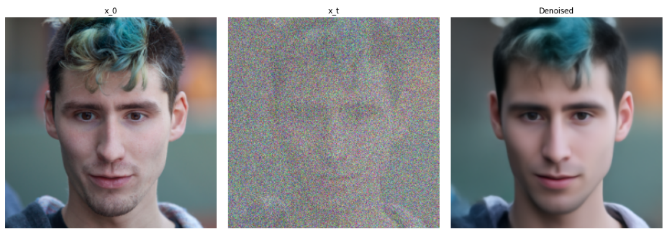

Fig. 2. Visual results of the single step forward process and denoising. The leftmost image represents the clean image, the image in the middle depicts the image after a single step of forward noising, and the rightmost image depicts the output of the denoising algorithm.

```
def forward_process(x_0, t, alphas_cumprod=
    alphas_cumprod):
    curr_alpha = alphas_cumprod[t]
    x0_coeff = np.sqrt(curr_alpha)
    noise = torch.normal(0.0, np.sqrt(1-
        curr_alpha), size=x_0.shape, device=
        device)
    x_t = x0_coeff * x_0 + noise
    return x_t
```

```
def denoise(x_t, t, eps):
    score = - eps / np.sqrt(1.0 -
        alphas_cumprod[t])
    x0_hat = sqrt_recip_alphas_cumprod[t] *
        x_t + sqrt_recipm1_alphas_cumprod[t]
         * score
    return x0_hat
```

We then implement the posterior sampling loop with the following functions:

```
def q_posterior_mean(x_start, x_t, t):
    assert x_start.shape == x_t.shape

    xt_coeff = posterior_mean_coef2[t]
    x0_coeff = posterior_mean_coef1[t]

    q = xt_coeff * x_t + x0_coeff * x_start
    return q
```

```
def p_sample(model, x, t):
    model_output = model(x, t)
    model_output, model_var_values = torch.
        split(model_output, x.shape[1], dim
        =1)

    x_0_hat = process_xstart(denoise(x, t,
        model_output))
    x_t = q_posterior_mean(x_0_hat, x, t)

    noise = torch.randn_like(x)
    model_log_variance = get_variance(
        model_var_values, t)
    if t != 0:  # no noise when t == 0
        x_t += torch.exp(0.5 *
            model_log_variance) * noise

    return {'sample': x_t, 'pred_xstart':
        x_0_hat}
```

Within the sampling loop itself, we have:

```
y = p_sample(model, img, time)
img = y['sample']
```

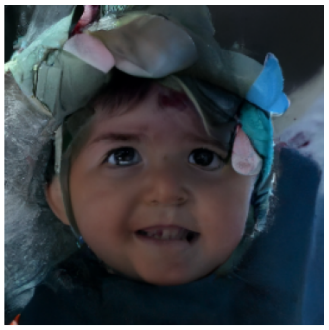

Fig. 3. Result of running posterior sampling on a completely random input image.

We run this for all 1000 timesteps against a randomly generated noisy image and obtain the result in Figure 3. As there is no ground truth image to be compared against, we cannot meaningfully use our PSNR and LPIPS metrics.

With the groundwork laid out, we move onto our implementation of the three reverse diffusion algorithms. We begin with our implementation of SDEdit, where we first define our hyperparameter $t = 500$ (implemented as init_time) and the resulting noisy image as a result of DDPM forward noising:

```
init_time = 500
guide_coeff = np.sqrt(alphas_cumprod[
    init_time])
noise_coeff = np.sqrt(1 - alphas_cumprod[
    init_time])
img = guide_coeff * measurement +
    noise_coeff * x_start
```

The remainder of the algorithm relies on the same sampling loop mentioned in the previous task. Some resulting images are depicted in Figure 4.

For our implementation of ScoreALD we add a few functions for computing the additional step mentioned in Equation (8):

```
def grad_likelihood(x_prev, x_0_hat,
    measurement, task, mask_type):

    Ax = forward(x_prev, task, mask_type)
    norm = torch.square(torch.linalg.norm(Ax
         - measurement))
    grad = torch.autograd.grad(outputs=norm,
         inputs=x_prev)[0]
    return grad
```

```
def scoreald_conditioning(x_prev, x_t,
    x_0_hat, measurement, task, mask_type,
    anneal_factor=1):

    grad_term = grad_likelihood(x_prev,
        x_0_hat, measurement, task,
        mask_type)
    x_t = x_t - grad_term / (np.power(sigma,
         2) + np.power(anneal_factor, 2)) /
        2
    return x_t
```

Lastly, we include the annealing hyperparameter γ_t, which depends on the task we are trying to accomplish:

```
if task == 'inpaint': # mask
    anneal_params = [15, 20]
else: # blur
    anneal_params = [10, 15]
annealing = torch.linspace(anneal_params[0],
     anneal_params[1], steps=num_timesteps)
```

For our DPS implementation, we modify the grad_likelihood function and slightly modify the update step to match Equation (9):

```
def grad_likelihood(x_prev, x_0_hat,
    measurement, task, mask_type):
    Ax = forward(x_0_hat, task, mask_type)
    norm = torch.square(torch.linalg.norm(Ax
         - measurement))
    grad = torch.autograd.grad(outputs=norm,
         inputs=x_prev)[0]
```

```
def dps_conditioning(x_prev, x_t, x_0_hat,
    measurement, task, mask_type):

    if task == 'deconv': # blur
        scale = 0.3
    elif mask_type == 'random':
        scale = 0.1
    else: # mask
        scale = 1.0
    grad_term = grad_likelihood(x_prev,
        x_0_hat, measurement, task,
        mask_type)
    zeta_t = scale * 2 * np.power(sigma, 2)
        / torch.norm(grad_term)
    x_t = x_t - zeta_t / (2 * np.power(sigma
        , 2)) * grad_term
    return x_t
```

5 Discussion

From Table 1, in terms of PSNR, DPS outperformed the other two reverse diffusion algorithms by a significant margin. The LPIPS score for DPS was typically lower than the other two as well, though ScoreALD was able to sometimes acheive a lower score than DPS.

Qualitatively, we can see that from SDEdit, 500 timesteps out of 1000 was not enough to get rid of the mask in the masked image, and it appears as a gray blotch that covers up the left upper cheek of the subject. Between both SDEdit images, both images have lost a lot of detail when compared to the ground truth image, and the outputs look like completely different people. When experimenting with a lower number of timesteps, the mask retention grew stronger, and the blurring effect from the blur input also grew stronger. When increasing the number of timesteps, the masking and blurring grew weaker, but the images began diverging further from the ground truth image.

We can clearly see that ScoreALD performed much better than SDEdit, with both output images looking somewhat similar to the ground truth image. The mask in the masked input resulted in something similar to a glasses frame, showing that the model wasn't able to fully remove the effects of masking. Additionally, the image appears to be much more saturated in terms of coloring, and the hair of the subject almost appears blurred. The blurred input

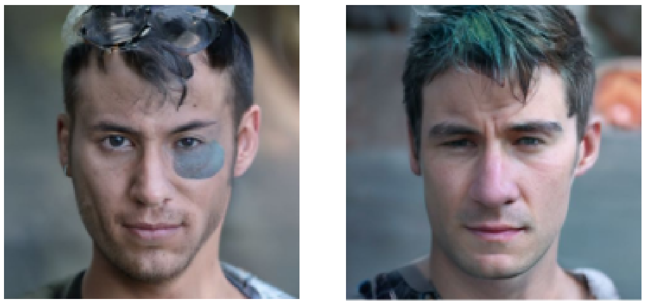
Fig. 4. SDEdit example output images. The image on the left is the result of the masked input image, while the image on the right is the result of the blurred input image.

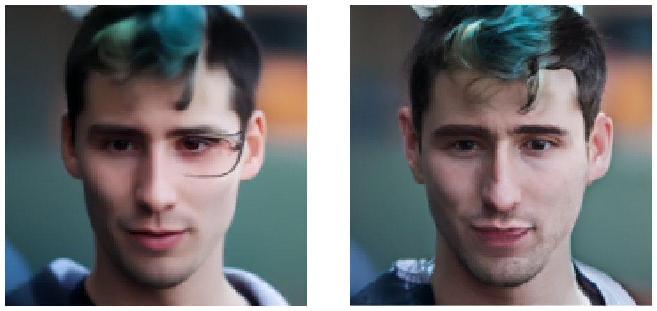
Fig. 5. ScoreALD example output images. The image on the left is the result of the masked input image, while the image on the right is the result of the blurred input image.

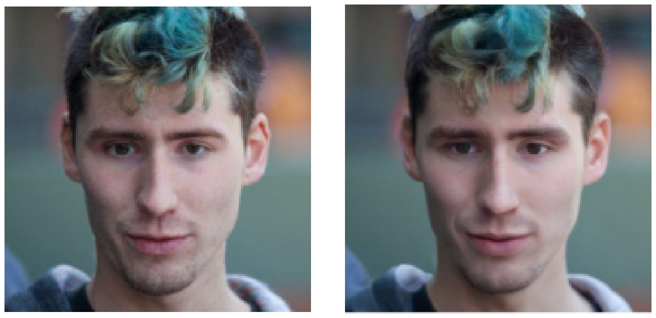
Fig. 6. DPS example output images. The image on the left is the result of the masked input image, while the image on the right is the result of the blurred input image.

appears to be closer to the ground truth image, though the expression of the subject is completely changed.

Lastly, with DPS, we can see that both output images are fairly close to the ground truth image. The mask appears to have been fully removed in the masked input, and there is no sign of blurring or saturation in either of the images. The expression remains the same and a lot of the details about the color and shape of the hair remains the same, though some differences such as smoothed skin appear.

5.1 Challenges and Limitations

When working on this project, we noticed that the diffusion model performed extremely poorly on non-human subjects. For example, we tried reconstructing a masked

	Mask PSNR	Mask LPIPS	Blur PSNR	Blur LPIPS
SDEdit	19.852 ± 0.502	0.288 ± 0.024	20.146 ± 0.077	0.270 ± 0.016
ScoreALD	22.445 ± 0.262	0.218 ± 0.012	22.813 ± 0.455	0.178 ± 0.014
DPS	35.428 ± 0.413	0.022 ± 0.001	29.149 ± 0.295	0.077 ± 0.008

TABLE 1
PSNR and LPIPS scores for each different reverse diffusion algorithm.

or blurred image of a red panda using each of the different reverse diffusion algorithms, and the resulting output looked nothing like the original image. This clearly was a result of the diffusion model being trained only human images, so the resulting output is to be expected. However, this highlights the fact that without an extremely large and diverse dataset, it is hard to produce a generalizable model over a domain-specific one.

Additionally, as seen in the output images, diffusion models have a tendency to hallucinate high frequency details. This is an obvious fact, since hallucinating details from noise is what diffusion models are made for. However, this may appear as a bigger issue for fields like medical imaging, where small details are critically important in diagnosing diseases or potentially life-threatening issues.

On the surface, it seems as though SDEdit is useless, as it performs significantly worse than both ScoreALD and DPS. However, both ScoreALD and DPS require knowledge of the forward model A (as mentioned in the Related Work section). In the real world, having no knowledge of the forward model is common, which makes ScoreALD and DPS unusable in those situations. Additionally, we noticed that performing ScoreALD and DPS were much more computationally expensive than SDEdit, so limitations in resources could determine which process is better, depending on the use case.

6 Conclusion

Utilizing diffusion models for solving inverse problems is still a very active area in the field of signal processing. As seen from the results of this project, they are not perfect and have many limitations. However, depending on the amount of information that we have in terms of forward models and the tolerance we need for hallucinations, they could be a very useful tool.

Acknowledgments

The author would like to thank PhD candidate Axel Levy for providing mentorship throughout the duration of this project and Professor Gordon Wetzstein for teaching EE 367.

References

[1] A. Buades, B. Coll, and J.-M. Morel, "A non-local algorithm for image denoising," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, 2005, pp. 60–65 vol. 2.

[2] S. Paris, P. Kornprobst, J. Tumblin, and F. Durand, "A gentle introduction to bilateral filtering and its applications," *ACM SIGGRAPH 2007 Papers - International Conference on Computer Graphics and Interactive Techniques*, p. 1, 08 2008.

[3] S. Kuthirummal, H. Nagahara, C. Zhou, and S. K. Nayar, "Flexible depth of field photography," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 58–71, 2011.

[4] R. Raskar, A. Agrawal, and J. Tumblin, "Coded exposure photography: motion deblurring using fluttered shutter," vol. 25, no. 3, p. 795–804, Jul. 2006. [Online]. Available: https://doi.org/10.1145/1141911.1141957

[5] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising," *Trans. Img. Proc.*, vol. 26, no. 7, p. 3142–3155, Jul. 2017. [Online]. Available: https://doi.org/10.1109/TIP.2017.2662206

[6] J. Lehtinen, J. Munkberg, J. Hasselgren, S. Laine, T. Karras, M. Aittala, and T. Aila, "Noise2noise: Learning image restoration without clean data," in *International Conference on Machine Learning (ICML)*, vol. 80, March 2018, pp. 2971–2980.

[7] S. Nah, T. Kim, and K. M. Lee, "Deep multi-scale convolutional neural network for dynamic scene deblurring," 07 2017, pp. 257–265.

[8] C. Chen, Q. Chen, J. Xu, and V. Koltun, "Learning to see in the dark," 2018. [Online]. Available: https://arxiv.org/abs/1805.01934

[9] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *CoRR*, vol. abs/2006.11239, 2020. [Online]. Available: https://arxiv.org/abs/2006.11239

[10] C. Meng, Y. Song, J. Song, J. Wu, J. Zhu, and S. Ermon, "Sdedit: Image synthesis and editing with stochastic differential equations," *CoRR*, vol. abs/2108.01073, 2021. [Online]. Available: https://arxiv.org/abs/2108.01073

[11] A. Jalal, M. Arvinte, G. Daras, E. Price, A. G. Dimakis, and J. I. Tamir, "Robust compressed sensing MRI with deep generative priors," *CoRR*, vol. abs/2108.01368, 2021. [Online]. Available: https://arxiv.org/abs/2108.01368

[12] C. Lu, Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu, "Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models," 2023. [Online]. Available: https://arxiv.org/abs/2211.01095

[13] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," 2018. [Online]. Available: https://arxiv.org/abs/1801.03924