Appendix

A Further Technical Details

A.1 Algorithm

Algorithm 1 Reverse Diffusion Process in KernelFusion

Input:
  Pretrained velocity model: PD_ψ
  Number of noise/denoise timesteps: T_{nd}
  Noise schedule: $\{\beta_t\}_{t=1}^{T}$ with $\alpha_t = 1 - \beta_t$
  Initial guess: $\hat{x}_{0,T_{nd}} = I_{LR} \uparrow_{bic,s}$
  Networks: U-Net R_θ, SIREN INR_ϕ with grid g
  Center of mass function: COM
  Number of optimization steps per t: n_{iter}
  Learning rate: γ
Output: Generated sample x_0
for $t = T_{nd}$ to 1 **do**
  $\xi \sim \mathcal{N}(0, I)$ → Sample noise
  for $i = 1$ to n_{iter} **do**
    $\hat{k}_\phi \leftarrow \mathrm{INR}_\phi(g)$ → Get kernel
    $\hat{x}_{0,t+1,\theta} \leftarrow \mathrm{R}_\theta(\hat{x}_{0,t+1})$ → Optimize x_0
    $\hat{x}_{0,t+1} \downarrow_s = (\hat{x}_{0,t+1,\theta} * \hat{k}_\phi) \downarrow_s$ → Downscale step
    $\hat{x}_{t,\theta} = \mu_{t+1}(\hat{x}_{0,t+1,\theta}, x_{t+1}) + \sigma_{t+1}\xi$
    $\hat{v}_\theta \leftarrow \mathrm{PD}_\psi(\hat{x}_{t,\theta}, t)$ → Denoising step
    $\hat{x}_{0,t,\theta} = \sqrt{\bar{\alpha}_t} \cdot \hat{x}_{t,\theta} - \sqrt{1 - \bar{\alpha}_t} \cdot \hat{v}_\theta$
    $\hat{x}_{0,t,\theta} \leftarrow \mathrm{R}_\theta(\hat{x}_{0,t,\theta})$ → Optimize x_0
    $\hat{x}_{0,t} \downarrow_s = (\hat{x}_{0,t,\theta} * \hat{k}_\phi) \downarrow_s$ → Downscale step
    $\mathcal{L} = \| I_{LR} - \hat{x}_{0,t} \downarrow_s \|_2^2 + \| I_{LR} - \hat{x}_{0,t+1} \downarrow_s \|_2^2 + \mathrm{COM}(\hat{k}_\phi)$
    $\theta \leftarrow \theta - \gamma \nabla_\theta(\mathcal{L})$ → Step for U-Net
    $\phi \leftarrow \phi - \gamma \nabla_\phi(\mathcal{L})$ → Step for INR
  end for
  $\mu_t = \beta_t \frac{\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_t} \hat{x}_{0,t,\theta} + (1 - \bar{\alpha}_{t-1}) \frac{\sqrt{\alpha_t}}{1-\bar{\alpha}_t} \hat{x}_{t,\theta}$
  $\zeta \sim \mathcal{N}(0, I)$ → Sample noise
  $x_{t-1} = \mu_t + \sigma_t \zeta$ → Add stochasticity, $\sigma_0 = 0$
end for
return x_0

A.2 Patch-Diffusion

Architecture The backbone model is a convolutional network, that inputs an image tensor x and a timestep t. We have a total of 6 blocks, each block conditioned on t. We use one block of two 3×3 filters, followed by five blocks of $3 \times 3 + 1 \times 1$ filters. We use 128 filters for the hidden layers.

Training Details The patch diffusion model is trained using random crops of size 64 pixels. The model is trained for $600'000$ steps, using Adam as optimizer, with a learning rate of $lr = 1 \times 10^{-4}$ and cosine annealing. Network weights are initiated using the default configurations of Pytorch.

A.3 U-Net

Architecture The refinement UNET consists of 5 blocks, with 32 filters on the input level and 512 filters on the bottom level. Each block consists of 2 convolutional layers with a 3×3 kernel, ReLU activations and batch norm. The final layer uses a tanh activation function to ensure that the predicted x_0 output is in the expected -1 to 1 range. The levels are down respectively upsampled by a scale factor of 2.

Training Details The U-NET is trained at each time step t during the reverse diffusion process. We use Adam optimizer, a learning rate of $lr = 1 \times 10^{-4}$. We apply cosine annealing, reducing the learning rate at each t to a final $lr = 1 \times 10^{-5}$. The U-NET is initialized at the first T_{start} of the reverse diffusion process and then finetuned along the different timesteps t. With the exception of the initial T_{start} where we apply $n_{iter} = 100$ iterations, the model is then finetune for $n_{iter} = 20$ iterations during each t. Network weights are initiated using the default configurations of Pytorch.

A.4 Implicit Neural Representation for Kernel Estimation

Architecture As described in Sec. Kernel Estimation, we took inspiration from SIREN Sitzmann et al. (2020) for our implicit neural representation. The network consists of 5 fully connected layers, with 256 nodes each. In contrast to the original paper, we reduced ω from 30 to 5 and apply it across all layers. Our last layer has an activation function which we call *leaky sigmoid*, a sigmoid function also allowing for slight negative values: $\sigma_{leaky}(x) = (1 + 10^{-4}) \cdot \sigma(x) - 10^{-4}$. The kernel is normalized such that its sum equals to 1. Additionally, as shown in Algorithm 1, a center of mass loss is introduced, encouraging the resulting kernel to be mass centered.

Training Details As we train our INR along with the U-NET, we use the same training setup as described in Sec. A.3.

B Additional Visual Examples

B.1 Additional Kernel Evaluation Results

Fig. 8 shows the results of the estimated kernels for 4 additional images providing additional evidence of the effectiveness of KernelFusion compared to current competitors. Fig. 10 provides a complete overview over all estimated kernels of the Blind144 dataset.

B.2 Additional Real Life Examples

Fig. 11 to 18 show additional visual comparisons of Super-Resolution applied on a sample of diverse *real-world* images, taken from a variety of different sources (the source of each real LR image is mentioned in each example). Our recovered SR-Kernel is also provided for each image. Since these are real images, there are NO ground-truth HR images nor ground-truth kernels.

C Additional Ablations

C.1 Kernel Network

As described in Sec. 4, KernelFusion leverages an Implicit Neural Representation (INR), specifically a SIREN Sitzmann et al. (2020) architecture. Alternative architectural choices are possible. In the following, two alternative choices are discussed: Directly optimizing the kernel and using a linear CNN as e.g. used in KernelGAN Bell-Kligler et al. (2019).

Direct Kernel Optimization: Each entry of the 24×24 kernel grid is treated as an individual parameter that is directly optimized. Note that this means that no additional network is applied. Additionally, the optimized kernel positivity (e.g. using a ReLU Agarap (2018) activation function) and a normalization such that its sum equals to 1 is applied.

Linear CNN: KernelGAN Bell-Kligler et al. (2019) leveraged a deep linear CNN for kernel estimation. This network consists solely of convolutional layers without any additional activations functions. The network is used to downscale the image and implicitly captures the kernel as such. It can be explicitly extracted by convolving all layers sequentially (see Bell-Kligler et al. (2019) for more details). Our network consists of 6 convolutional layers with kernel sizes, $[7, 7, 5, 3, 1, 4]$, channels $[1, 32, 32, 32, 32, 32, 1]$ and strides $[1, 1, 1, 1, 1, 4]$. Not that the last stride entry indicates the scale factor of 4 used in our experimental setup.

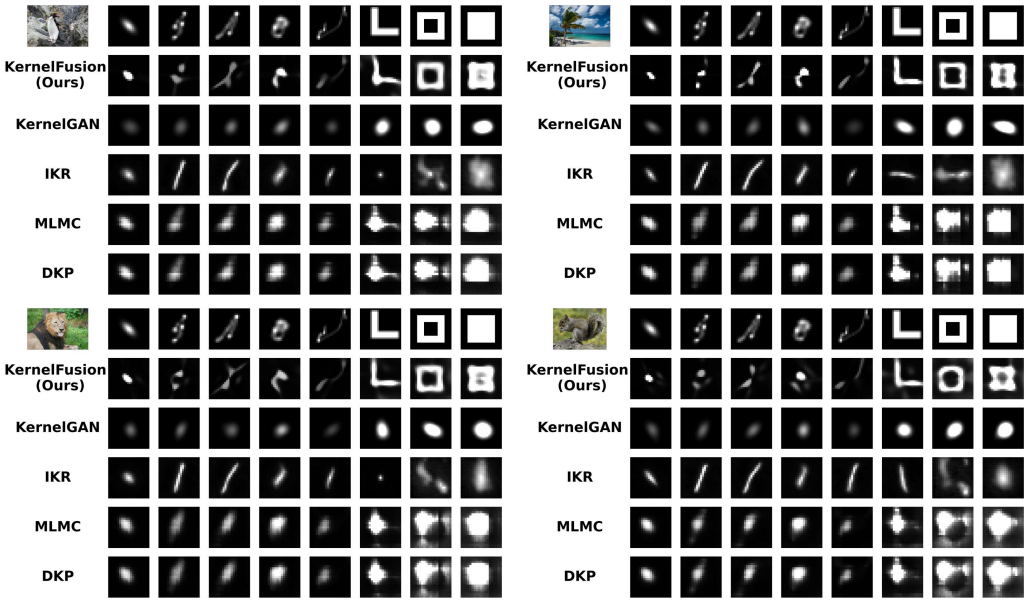

Figure 8: **Comparison of estimated kernels from different Blind-SR methods.** The top row represents the ground-truth (GT) degradation kernels, while each subsequent row corresponds to the estimated kernels from different SR methods, including our approach, KernelGAN, IKR, MLMC, and DKP. Our method demonstrates superior flexibility in recovering complex, non-Gaussian degradations, accurately capturing kernel structures across a diverse range of degradations.

Table 4: Kernel Network ablations on K0,K1,K2 of Blind144

Method	**KernelFusion PSNR↑**	**DIP PSNR↑**
Direct Sigmoid	16.366	15.529
Direct Relu	26.541	24.266
Linear CNN	23.494	23.098
INR	26.995	23.638

Experimental Setup: As shown in Table 4, we compare four different architectural choices for the kernel estimation: A direct kernel optimization (one with a Sigmoid and one with a ReLU activation function), a KernelGAN like linear CNN, and the SIREN INR applied in KernelFusion. To better show the power of our diffusion setup, we additionally evaluated the architectures in a DIP setup (as used in the ablations in Sec. 5).

Results: Table 4 shows the resulting SR PSNRs on the synthetic kernels k_0, k_1, and k_2 of Blind144 (see Fig. 19 and Fig. 20 for reference). KernelFusion with the INR shows superior results over the alternatives. Nevertheless, the direct optimization using a ReLU activation shows suprisingly good results. However, when comparing the reconstructed kernels (see Fig. 19 and Fig. 20) it becomes clear that the direct optimization leads to noisy, non-consistent kernels. Using an INR, much more accurate kernels can be recovered.

C.2 Applying the U-Net twice

As discussed in Sec. 4 and in Algorithm 1, the same U-Net is applied twice, once before and once after PD. An alternative design choice could consist of two separate, identical U-Nets that do not share weights and are optimized individually with an identical parameter setup. Table 5 shows the result of the described ablation. With a PSNR of 26.498 dB, optimizing two separate U-Nets is significantly worse than optimizing the a single U-Net that has been applied twice. The intuition

Table 5: Two-UNet ablation: Weight sharing across both stages improves performance over using two independently trained U-Nets.

Method	**Blind144 PSNR↑**
KernelFusion (Two separate U-Nets)	26.498
KernelFusion (same U-Net applied twice)	27.191

behind applying the same U-Net twice can be described as follows: Once the target prediction is sufficiently good at small t's, the U-Net output difference between x_0 at $t+1$ (before PD) and t (after PD) should be marginal as the same images will have the same patch distribution (also compare Fig. 21). On the other side, we initialize Phase 2 with a bicubic guess of for x_0. Due to the high noise level, PD destroys global structure while adjusting the patch distribution and the second application of the U-Net receives a heavily corrupted input. During this early phase, the first application of the U-Net effectively acts as a regularizer that leverages the bicubic guess to provide the global structure information.

D Receptive Field of Patch Diffusion Network

As discussed in Sec. 4 and Sec. 5, PD operates with a receptive field of 15×15 pixels. Consequently, PD learns local structure at the patch level and has no explicit notion of global image layout. When predicting $\hat{x}_0$ at large t (high noise levels), this can cause the reconstruction to lose the exact global structure of the image. Fig. 21 illustrates this: for small t, the noised image still preserves the global structure, and PD produces a reasonable reconstruction. However, for large t (e.g., $t = 800$), the input to PD is heavily corrupted, and PD reconstructs an image whose local patches match the learned LR patch distribution, but the global structure becomes strongly blurred and inconsistent, yielding a mosaic of plausible local patches. This effect diminishes as t decreases and more of the original structure remains visible. Introducing a U-Net as a global image prior compensates for this lack of global awareness, allowing us to both adjust the patch distribution and recover an accurate reconstruction.

E Runtime Improvements

As pointed out in *Limitations* in Sec. 5, KernelFusion has to be trained from scratch for each image individually and hence comes with a training cost that is reflected in runtimes that are longer than other SOTA SR algorithms. While KernelFusion is not yet fast enough for commercial deployment, it serves a different purpose: establishing the scientific feasibility of a task previously considered impossible; Blind-SR with unrestricted kernels.

That said, optimization is a logical next step. Table 6 shows the SR PSNRs using a patch diffusion model trained for 100k steps as well as the 600k setup used for our main evaluation. With just 100k steps, patch diffusion can capture much of the necessary statistics. This yields a $4\times$ speedup (~ 5 minutes compared to ~ 20 minutes) with negligible performance loss ($\sim$0.07 dB).

As runtimes for KernelFusion are above commercially deployed algorithms, two points shall be additonally stressed:

(i) Breaking Assumptions: As shown in Fig. 1, SOTA methods rely on fixed priors (Gaussian/Motion) and collapse when these assumptions break. Our paper points out this general limitation and brings it to the attention of the community. KernelFusion succeeds in this "impossible" regime by extracting the prior directly from the LR test image. We note that other single-image / internal-learning methods (e.g., KernelGAN Bell-Kligler et al. (2019), MLMC Xia et al. (2024), DKP Yang et al. (2024b)) also require per-image optimization and typically take several minutes per image to produce their SR output. Yet, none of these can handle arbitrary downscaling kernels.

(ii) Feasibility: We see a parallel to early foundational works like NeRF Mildenhall et al. (2021) or Deep Image Prior Ulyanov et al. (2018), which required significant computation to establish feasibility in new domains. By proving that unrestricted kernel recovery is solvable, KernelFusion

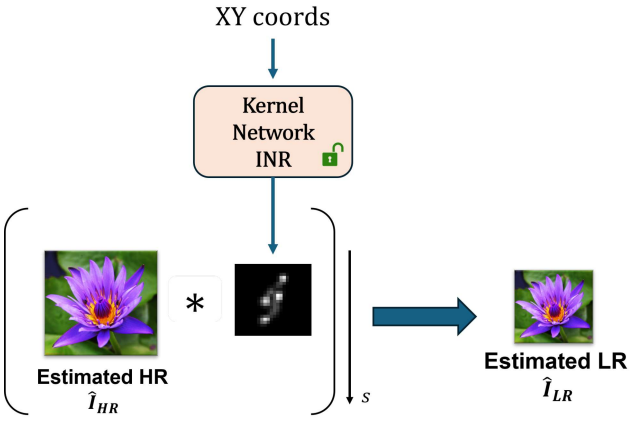

Figure 9: **Illustration of the kernel estimation in Phase 2.** The INR-based kernel network maps normalized (x, y) coordinates to kernel values, producing a downscaling kernel that is convolved with the current HR estimate and downsampled by a factor s to yield the predicted LR image $\hat{I}_{\text{LR}}$.

Table 6: Runtime improvements: Training PD for 100k steps instead of 600k steps leads to marginal losses in PSNR.

Method	**Blind144 PSNR↑**	**Runtime min↓**
KernelFusion (PD: 100K)	27.117	~ 5
KernelFusion (PD: 600K)	27.191	~ 20

opens a new, free from predefined kernel assumptions paradigm. We hope that the community will build on these foundations to bridge the gap to real-time efficiency.

F Details on Kernel Estimation in Phase 2.

Given the current HR estimate $\hat{I}_{\text{HR}}$ and the INR-based kernel network INR_θ, we represent the downscaling kernel on a $K \times K$ grid (in our experiments $K = 24$). For each normalized kernel coordinate g we evaluate

$$k_\theta = \text{INR}_\theta(g), \tag{3}$$

To obtain a valid Kernel, we enforce non–negativity and unit sum. Using this kernel, the predicted LR image $\hat{I}_{\text{LR}}$ is obtained by convolving the current HR estimate with $\hat{k}_\theta$ and then downsample by the scale factor s (here $s = 4$):

$$\hat{I}_{\text{LR}} = (\hat{I}_{\text{HR}} * \hat{k}_\theta) \downarrow_s, \tag{4}$$

where $\downarrow_s$ denotes subsampling by s and $*$ represents a discrete convolution.

The patch-diffusion model in Phase 1 was trained on the observed LR image I_{LR}. Therefore, in Phase 2 we enforce consistency between the predicted LR image and the original observed LR image via an MSE loss,

$$\mathcal{L}_{\text{cons}} = \left\| \hat{I}_{\text{LR}} - I_{\text{LR}} \right\|_2^2 \tag{5}$$

which couples the HR estimate $\hat{I}_{\text{HR}}$ and the kernel parameters θ through the forward model. Figure 9 illustrates the kernel estimation process.

G Use of Large Language Models

Large Language Models (LLMs) have been used while writing the paper i.e. to identify typos, improve formulations or shorten text snippets.

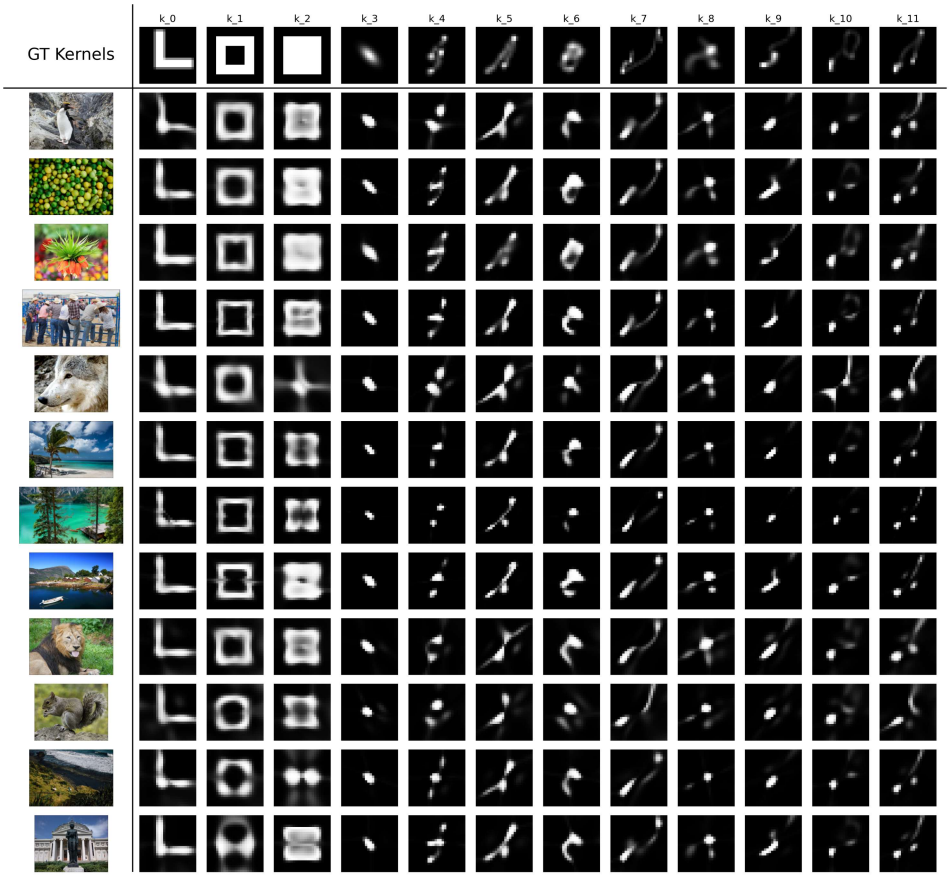

Figure 10: **Estimated kernels of KernelFusion on Blind144:** Complete overview of all 144 estimated kernels.

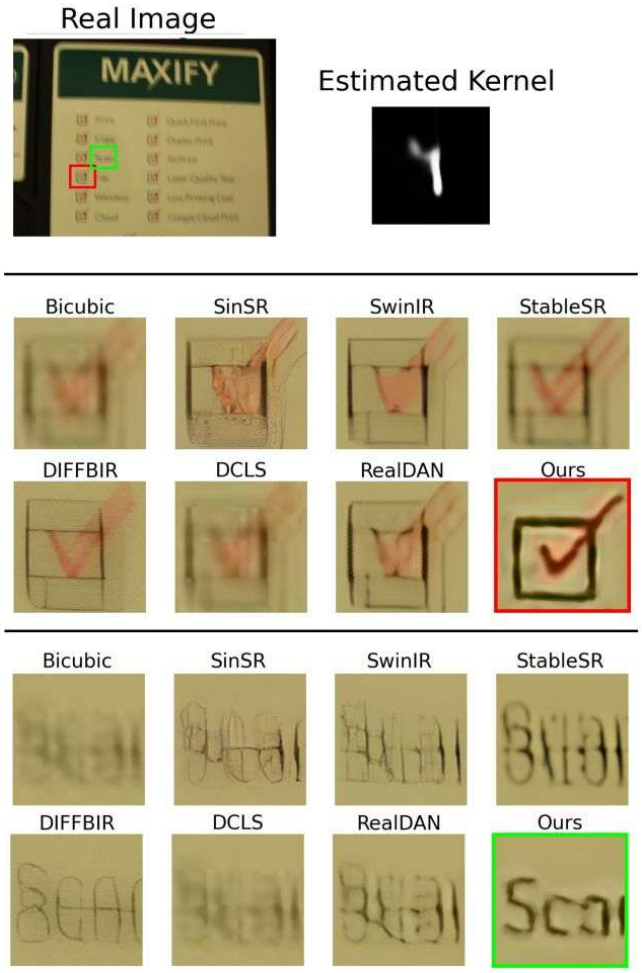

Figure 11: **Super-Resolution of a REAL image (taken from Lai et al. (2016)).**

(*Top*) The real image (used as the LR input) & its estimated SR-kernel (using our method).

(*Middle* & *Bottom*) A visual comparison of SR×4 results of leading SR methods. Zooms of the red and green image regions are displayed, respectively. *(There is NO ground-truth HR image)*

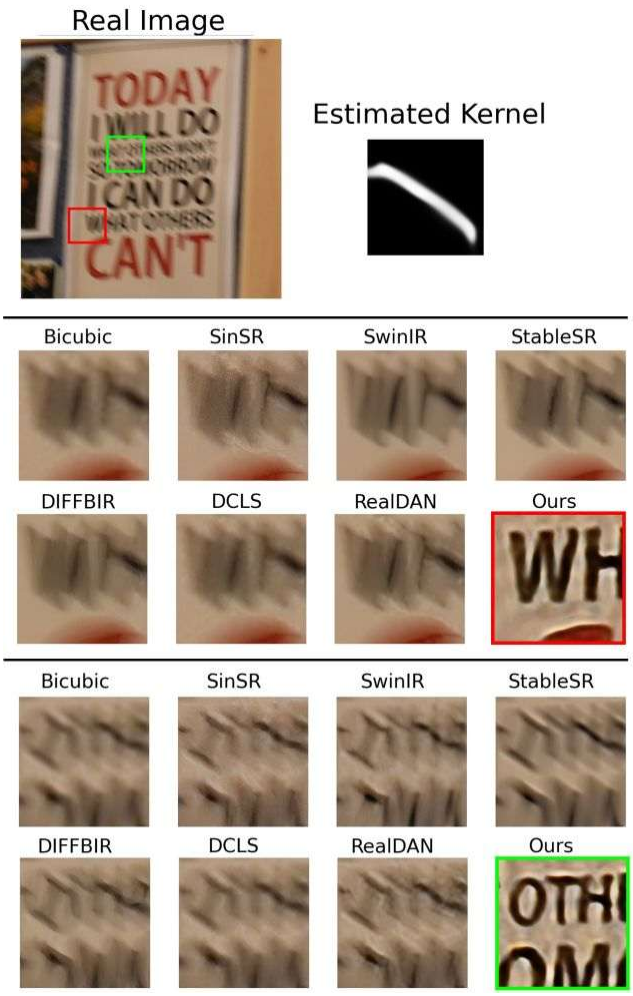

Figure 12: **SR of a REAL image (captured using a DSLR camera *without* optical stabilization).**

(*Top*) The real image (used as the LR input) & its estimated SR-kernel (using our method).

(*Middle* & *Bottom*) A visual comparison of SR×4 results of leading SR methods. Zooms of the red and green image regions are displayed, respectively. *(There is NO ground-truth HR image)*

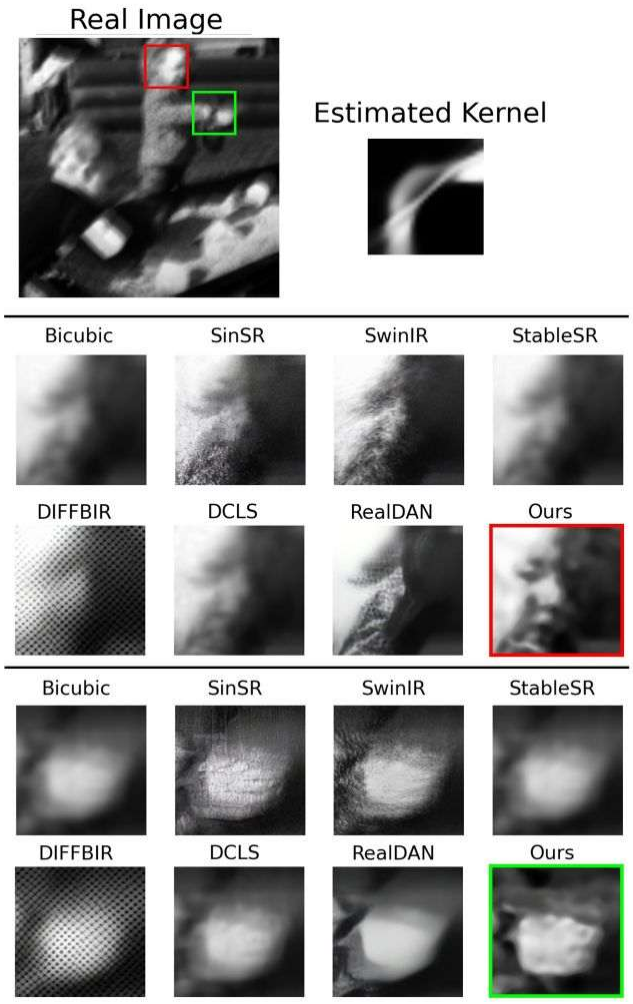

Figure 13: **Super-Resolution of a REAL image (taken from Levin et al. (2009)).**

(*Top*) The real image (used as the LR input) & its estimated SR-kernel (using our method).

(*Middle* & *Bottom*) A visual comparison of SR×4 results of leading SR methods. Zooms of the red and green image regions are displayed, respectively. *(There is NO ground-truth HR image)*

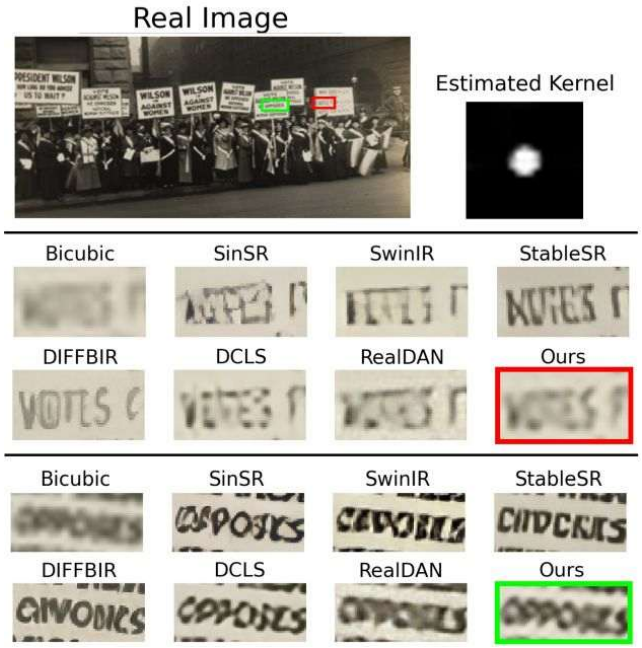

Figure 14: **SR of a REAL old (poor-quality) historic photo downloaded from the internet.**

(*Top*) The real image (used as the LR input) & its estimated SR-kernel (using our method).

(*Middle* & *Bottom*) A visual comparison of SR×4 results of leading SR methods. Zooms of the red and green image regions are displayed, respectively. *(There is NO ground-truth HR image)*

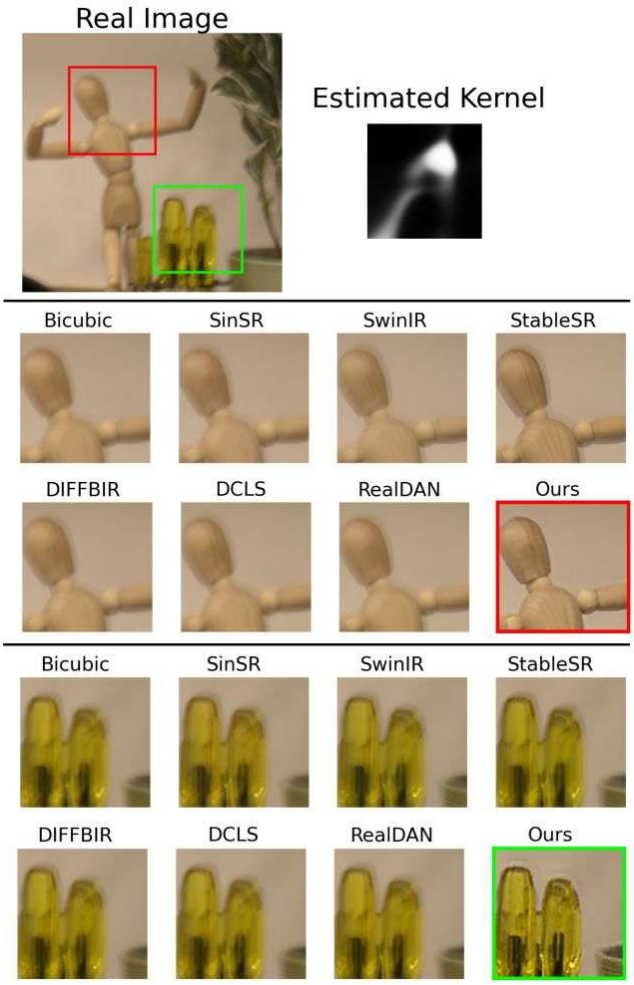

Figure 15: **SR of a REAL image (captured using a DSLR camera *without* optical stabilization).**

(*Top*) The real image (used as the LR input) & its estimated SR-kernel (using our method).

(*Middle* & *Bottom*) A visual comparison of SR×4 results of leading SR methods. Zooms of the red and green image regions are displayed, respectively. *(There is NO ground-truth HR image)*

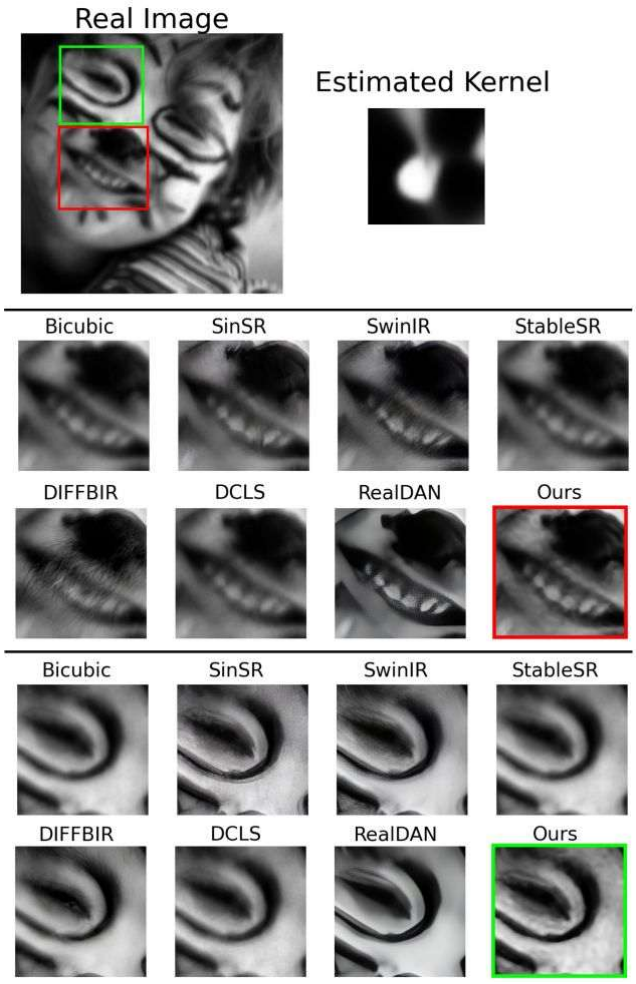

Figure 16: **Super-Resolution of a REAL image (taken from Levin et al. (2009)).**

(*Top*) The real image (used as the LR input) & its estimated SR-kernel (using our method).

(*Middle* & *Bottom*) A visual comparison of SR×4 results of leading SR methods. Zooms of the red and green image regions are displayed, respectively. *(There is NO ground-truth HR image)*

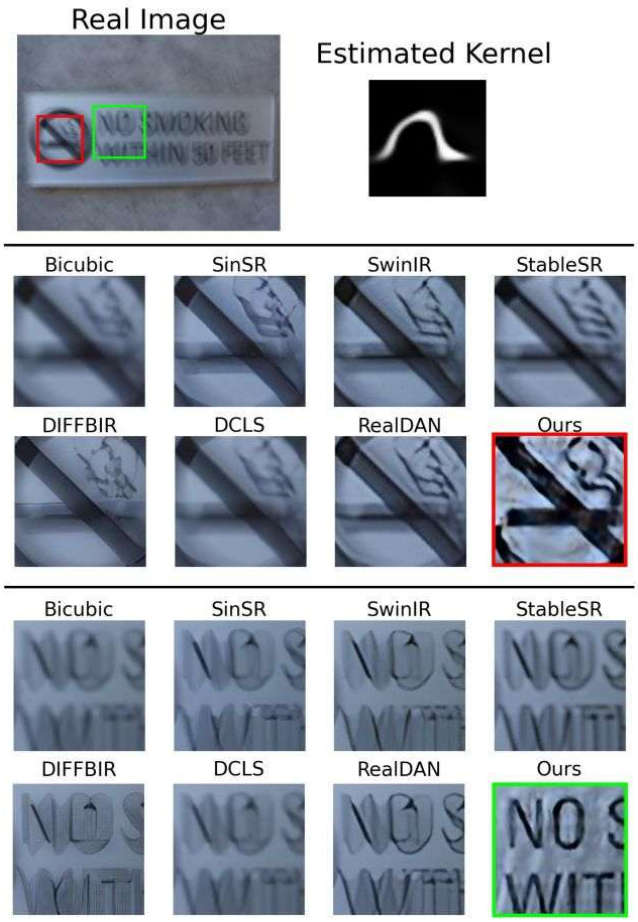

Figure 17: **Super-Resolution of a REAL image (taken from Lai et al. (2016)).**

(*Top*) The real image (used as the LR input) & its estimated SR-kernel (using our method).

(*Middle & Bottom*) A visual comparison of SR×4 results of leading SR methods. Zooms of the red and green image regions are displayed, respectively. *(There is NO ground-truth HR image)*

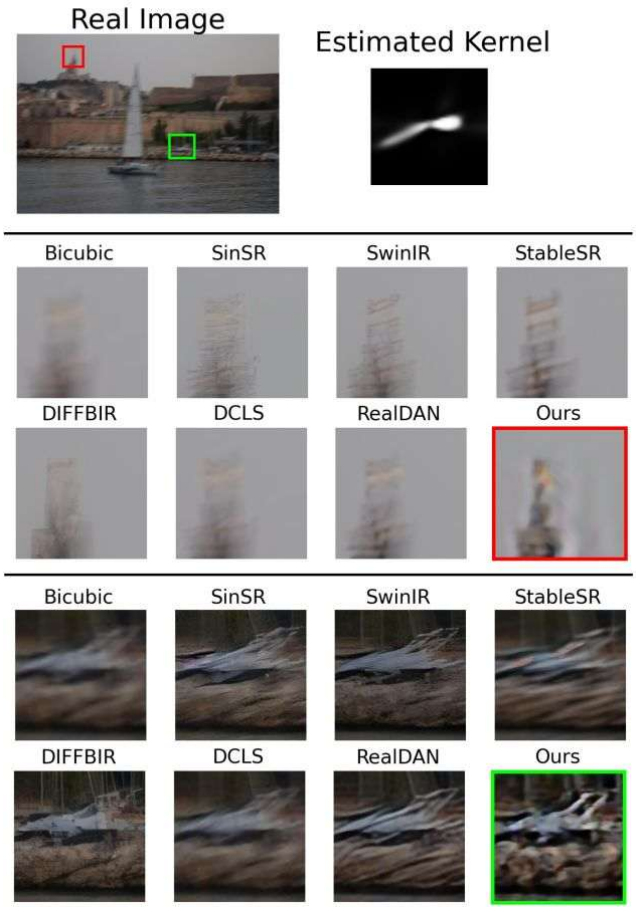

Figure 18: **Super-Resolution of a REAL image (taken from Lai et al. (2016)).**

(*Top*) The real image (used as the LR input) & its estimated SR-kernel (using our method).

(*Middle* & *Bottom*) A visual comparison of SR×4 results of leading SR methods. Zooms of the red and green image regions are displayed, respectively. *(There is NO ground-truth HR image)*

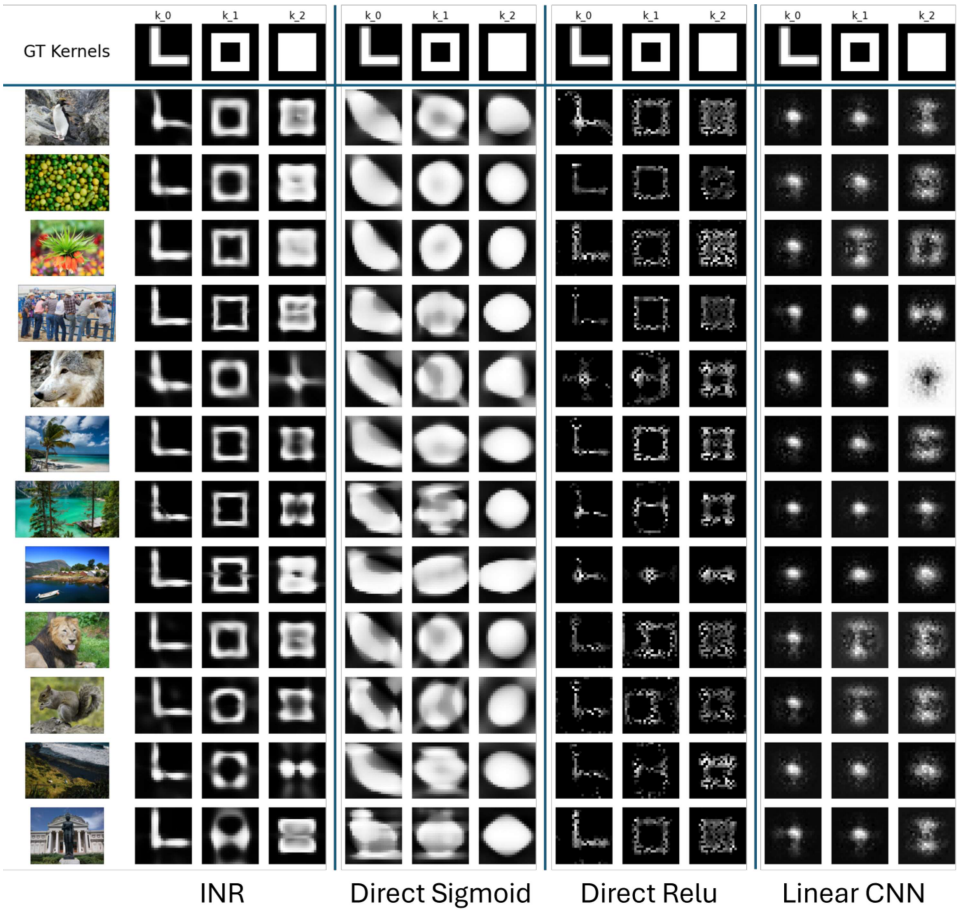

Figure 19: **Estimated kernels for KernelFusion setup with 4 different kernel estimation approaches:** INR, direct kernel estimation with sigmoid activation, direct kernel estimation with ReLu activation, and linear CNN inspired by KernelGAN.

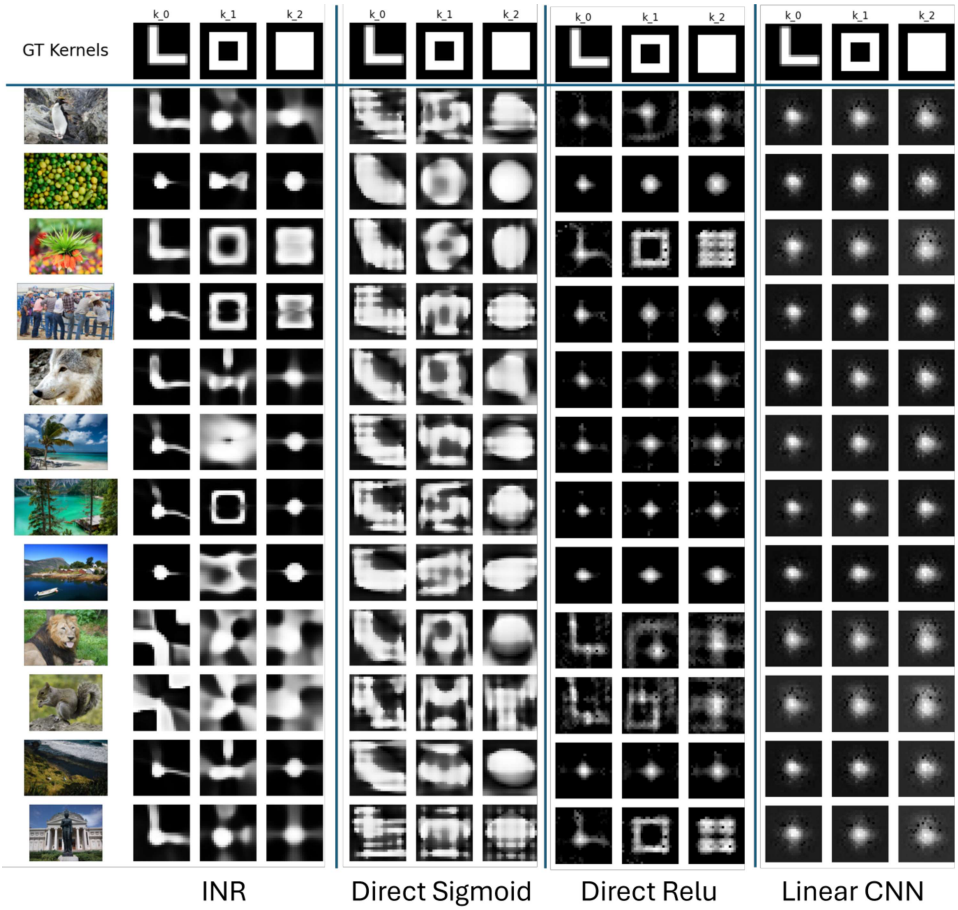

Figure 20: **Estimated kernels for the DIP setup with 4 different kernel estimation approaches:** INR, direct kernel estimation with sigmoid activation, direct kernel estimation with ReLu activation, and linear CNN inspired by KernelGAN.

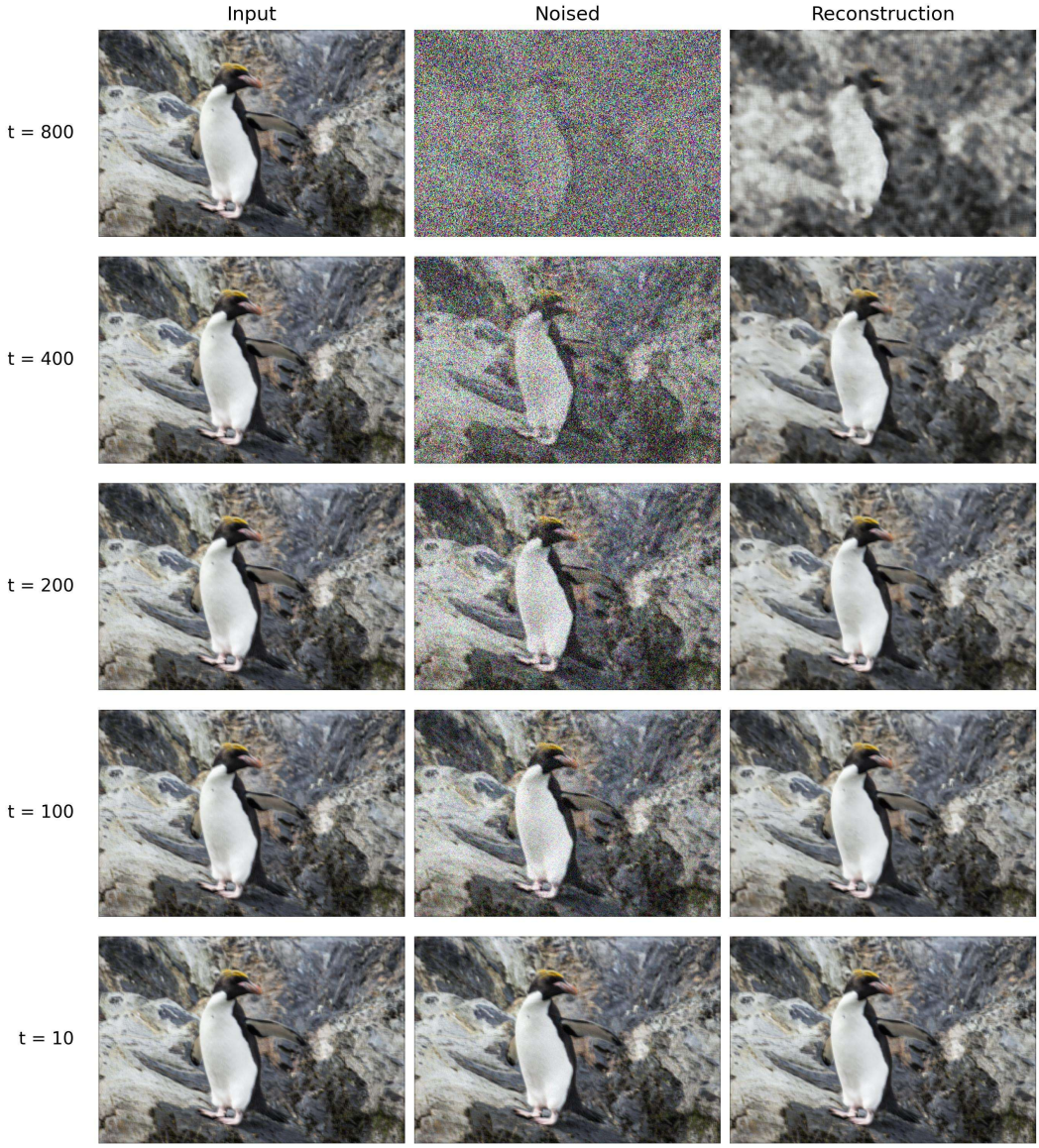

Figure 21: **Prediction of $\hat{x}_0$ using Patch Diffusion:** At high noise levels - corresponding to large t's - patch diffusion struggles to maintain global structure, and produces a mosaic-like reconstruction of local patches (best viewed zoomed in).