Appendix

A Importance of Upsampling

In addition to image denoising, we demonstrate that the unlearnt upsampling operation is key to other image restoration tasks as well, as shown in Fig.12. The experiments are conducted using MLP-Decoder, same as the setup for Fig.3 (a).

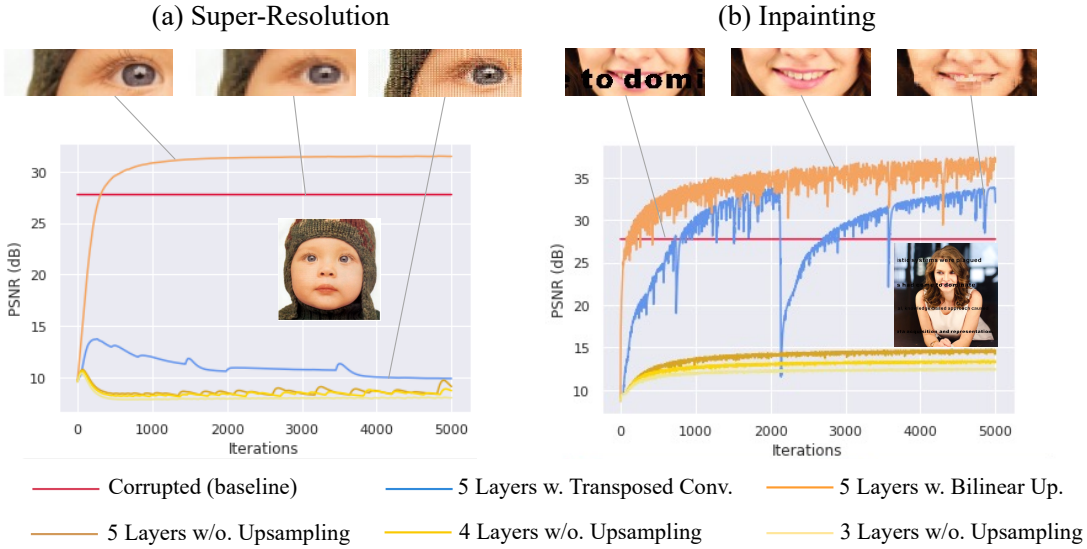


Fig. 12: Influences of the architecture components on **super-resolution** and **image inpainting**, evaluated on MLP-Decoder. The baseline method for super-resolution is nearest-neighbor interpolation.

B Additional Comparisons

To illustrate that our architectural design is also advantageous in avoiding the early-stopping oracle, here we additionally compare with a recent method proposed by **Shi et al. (2022)** [7] that explicitly detects the timing for stopping by regularizing the norms of the weights and using the ratio of no-reference blurriness and sharpness as the stopping criterion. Their method introduces new hyperparameters such as the regularization granularity, the sigmas of Gaussian upsampling varied across layers and the ratio eplison. We followed the default choices for these hyper-parameters. The optimization stops at around 5000th iteration, where the criterion is met. For NAS [2] and ISNAS [1] methods, the training is early-stopped at the 1200th iteration. For all the other methods including ours, a fixed iteration number of 3000 is adopted.

Table 7: Quantitative results on Set9 [3] with varying noise levels.

Method	$\sigma = 25$		$\sigma = 50$	
	PSNR	SSIM	PSNR	SSIM
Shi et al. [7]	28.45	0.851	24.28	0.706
Ours	**30.26**	**0.900**	**26.13**	**0.833**

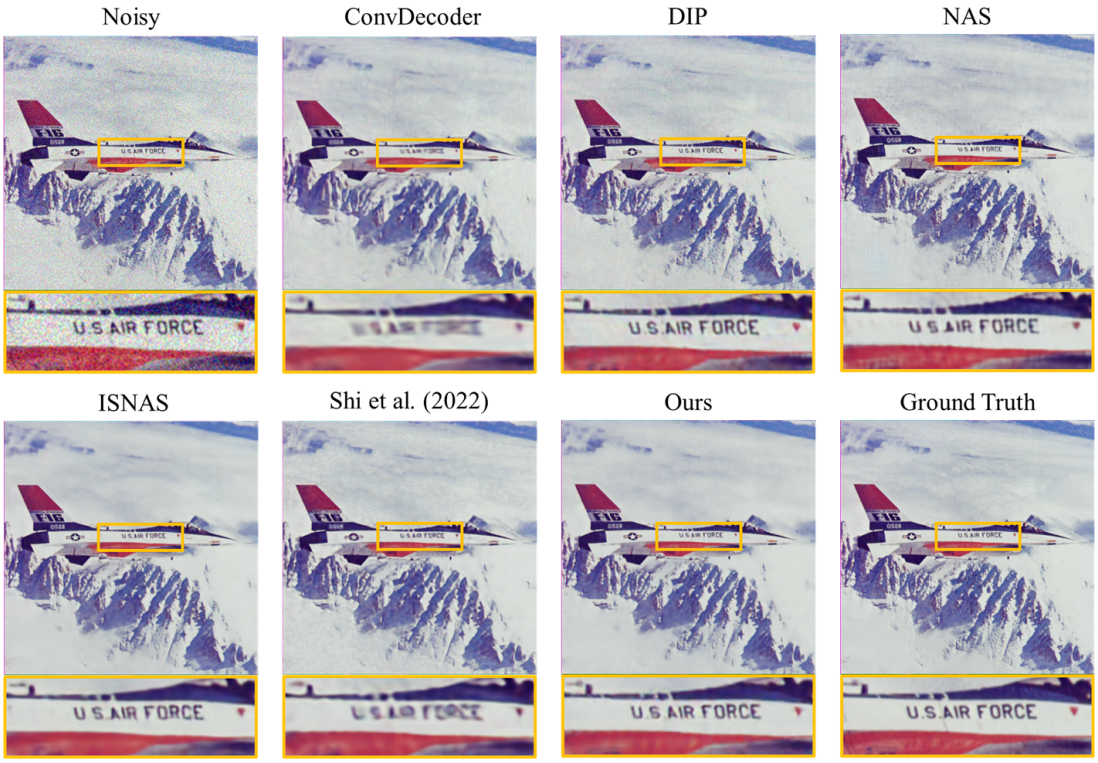


Fig. 13: Qualitative comparisons on 'F16' from Set9 [3] under **Gaussian noise with** $\sigma = 25$

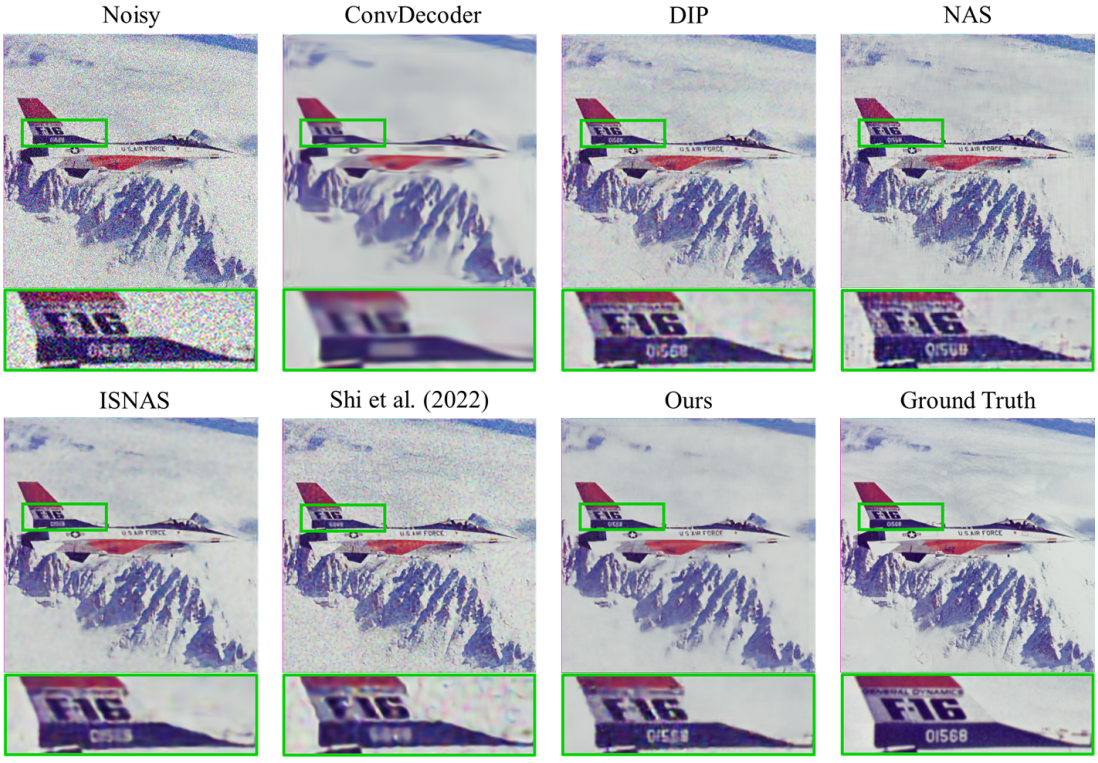


Fig. 14: Qualitative comparisons on 'F16' from Set9 [3] under **Gaussian noise with** $\sigma = 50$

C The Customized Upsamplers

C.1 Filter Construction

As stated in the main text, upsampling can be seen as zero insertion followed by filtering. We therefore insert zeros into the input in an interleaved fashion, and then perform filtering with the following filter coefficients respectively:

Nearest neighbor (NN): [0.5, 0.5]

Bilinear: [0.25, 0.5, 0.25]

$\mathcal{L}_{14}$: [-0.0054, -0.0518, 0.2554, 0.6036, 0.2554, -0.0518, -0.0054]

$\mathcal{L}_{15}$: [0.0105, -0.0263, -0.0518, 0.2763, 0.5826, 0.2763, -0.0518, -0.0263, 0.0105]

$\mathcal{L}_{-60}$: [-0.001921, -0.004291, 0.009947, 0.021970, -0.022680, -0.073998, 0.034907, 0.306100, 0.459933, 0.306100, 0.034907, -0.073998, -0.022680, 0.021970, 0.009947, -0.004291, -0.001921]

$\mathcal{L}_{-100}$: [0.000015, 0.000541, 0.003707, 0.014130, 0.037396, 0.075367, 0.121291, 0.159962, 0.175182, 0.159962, 0.121291, 0.075367, 0.037396, 0.014130, 0.003707, 0.000541, 0.000015]

$\mathcal{L}_{16}$: [-0.0006, 0.0129, -0.0294, -0.0518, 0.2800, 0.5777, 0.2800, -0.0518, -0.0294, 0.0129, -0.0006]

$\mathcal{L}_{17}$: [0.0003, -0.0001, -0.0043, 0.0065, 0.0205, -0.0499, -0.0475, 0.2936, 0.5621, 0.2936, -0.0475, -0.0499, 0.0205, 0.0065, -0.0043, -0.0001, 0.0003]

Particularly, $\mathcal{L}_{14}$, $\mathcal{L}_{15}$, $\mathcal{L}_{16}$ and $\mathcal{L}_{17}$ are Butterworth filters which have a maximally flat frequency response in the passband. They are designed to match the passband frequency response of the Nearest Neighbor. They have the same bandwidth of 0.5, but with a different order that determines the steepness of the transition from the passband to the stopband. $\mathcal{L}_{14}$ is a 6-order 1-D Butterworth low pass filter (LPF) corresponding to a 7×7 kernel in 2-D. Its coefficients can be obtained in matlab via: maxflat$(6, \text{'sym'}, 0.5)$. $\mathcal{L}_{15}$ is an 8-order (9×9 kernel in 2-D) Butterworth LPF and can be obtained via maxflat$(8, \text{'sym'}, 0.5)$. $\mathcal{L}_{-60}$ and $\mathcal{L}_{-100}$ are designed using the Kaiser window. For $\mathcal{L}_{-60}$, we set the cutoff frequency to 0.46 and the beta of the Kaiser window to 4.5. For $\mathcal{L}_{-100}$, we set the cutoff frequency to 0.1 and the Beta of the Kaiser window to 10. With these coefficients, each filter is constructed as below and can then be used as a plug-in module for any network architecture.

Algorithm 1: PyTorch-style pseudocode for customized upsampling

```
# upx: the upsampling scaling factor in the x direction
# upy: the upsampling scaling factor in the y direction
# x: the input to be upsampled
def InsertZeros(x, upx, upy, gain=1.0):
   b,c,h,w = x.size()
   x = x.reshape([b, c, h, 1, w, 1])
   x = F.pad(x, [0, upx - 1, 0, 0, 0, upy - 1])
   x = x.reshape([b, c, h * upx, w * upy])
   x = x * gain
   return x
# LPF construction
# w: the coefficients
def lowpass_conv(num_chns, w, pad_size='same', pad_mode='zeros', gain=1.0):
   # filter size
   k_size = len(w)
   # Convert 1D LPF coefficients to 2D
   f_2d_coeff = torch.outer(w,w)
   f_weights = torch.broadcast_to(f_2d_coeff, [num_chns, 1, k_size, k_size])
   conv = nn.Conv2d(num_chns, num_chns, kernel_size=k_size, stride=1, padding=pad_size,
    padding_mode=pad_mode, bias=False, groups=chs) # Initialize a filter
   f_weights = f_weights * gain
   conv.weight.data = f_weights
   conv.weight.requires_grad = False
   return conv
```

C.2 Evaluation on an hourglass architecture

To supplement the table in Fig.4, we repeated the evaluation of the customized upsamplers on the original DIP model [8], i.e., a 5-level hourglass architecture with full skip connections and 128 channels per layer.

Similarly, we also observe that NN, $\mathcal{L}_{14}$ and $\mathcal{L}_{15}$ are the fastest to reach the Peak PSNR for both kinds of images (Table 10), and since DIP [8] is more heavily-parameterized than ConvDecoder [4], they lead to performance drop in both cases sooner due to over-fitting (Fig. 9, Fig. 10). By contrast, Bilinear, $\mathcal{L}_{-60}$ and $\mathcal{L}_{-100}$ perform relatively more stably throughout the training, and benefiting from the enhanced model capacity, they achieve better results than they do on ConvDecoder. This again suggests that the final performance depends on the balance between the increased depth/width and upsampling operations.

Table 8: **Peak PSNR** values.

	NN	$\mathcal{L}14$	$\mathcal{L}15$	$\mathcal{L}16$	$\mathcal{L}17$	Bilinear	$\mathcal{L}_{-60}$	$\mathcal{L}_{-100}$
Coarse-grained	32.05	32.28	32.38	32.19	32.48	32.39	32.22	31.35
Fine-grained	25.14	25.15	25.10	25.02	25.01	24.64	24.60	21.85

Table 9: **Final PSNR** values at the 3000^{th} iteration.

	NN	$\mathcal{L}14$	$\mathcal{L}15$	$\mathcal{L}16$	$\mathcal{L}17$	Bilinear	$\mathcal{L}_{-60}$	$\mathcal{L}_{-100}$
Coarse-grained	22.67 $\downarrow$	25.64 $\downarrow$	26.90 $\downarrow$	27.42 $\downarrow$	29.40 $\downarrow$	31.67	30.43	31.35
Fine-grained	22.65 $\downarrow$	23.92 $\downarrow$	24.52 $\downarrow$	24.64 $\downarrow$	24.90	24.65	24.60	21.85

Table 10: **The iteration number [iter./3000] where the peak PSNR is reached with different upsamplers.** As the strength of attenuation increases (from left to right), the slower the peak PSNR is reached.

	NN	$\mathcal{L}14$	$\mathcal{L}15$	$\mathcal{L}16$	$\mathcal{L}17$	Bilinear	$\mathcal{L}_{-60}$	$\mathcal{L}_{-100}$
Coarse-grained	926	1197	1153	1275	1953	2151	3000	3000
Fine-grained	1497	1935	2250	2426	2730	3000	3000	3000

D Large-Scale Validation Experiments

D.1 Depth and Width

To further illustrate the influences of depth and width, we test the hourglass architectures with $1 \sim 8$ encoder-decoder layers (i.e., $1 \sim 8$-level) and the following width choices $\{32, 48, 64, 96, 128, 256, 512\}$ and ran all $8 \times 7 = 56$ combinations. Note that skip connections are not involved and every decoder layer is assumted to be followed by a $2\times$ bilinear upsampling operation.

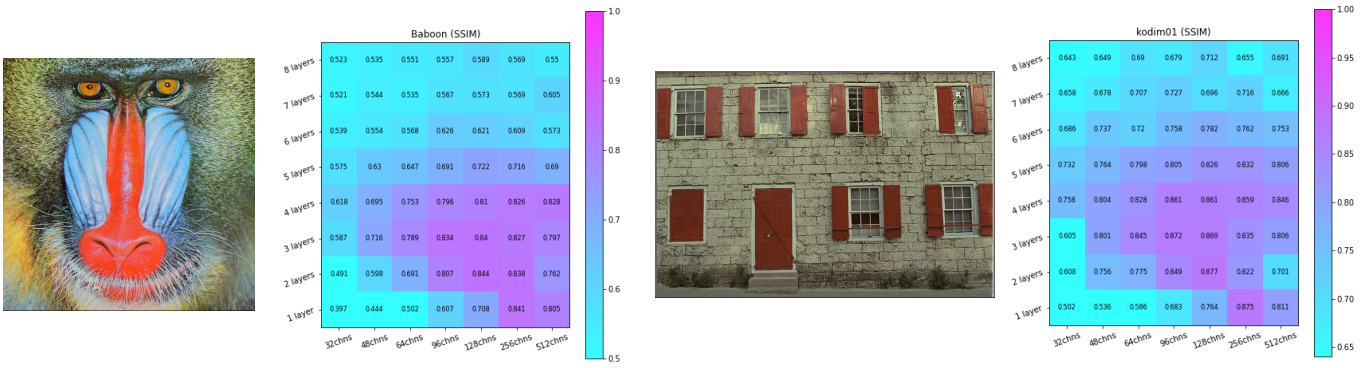


Fig. 15: SSIM (↑) values of different depth and width combinations for **fine-grained images**. These images are sensitive to the increased depth where the low-pass filtering effects caused by upsampling are stronger, since they have a wider bandwidth and are more susceptible to losses of high-frequency details. Thus, the high-performing architectures tend to be **shallow** ($1 <$ depth ≤ 4) and **wide** (128/256 channels). They are also robust to width choices $96 \sim 256$ channels. No skip connections are involved.

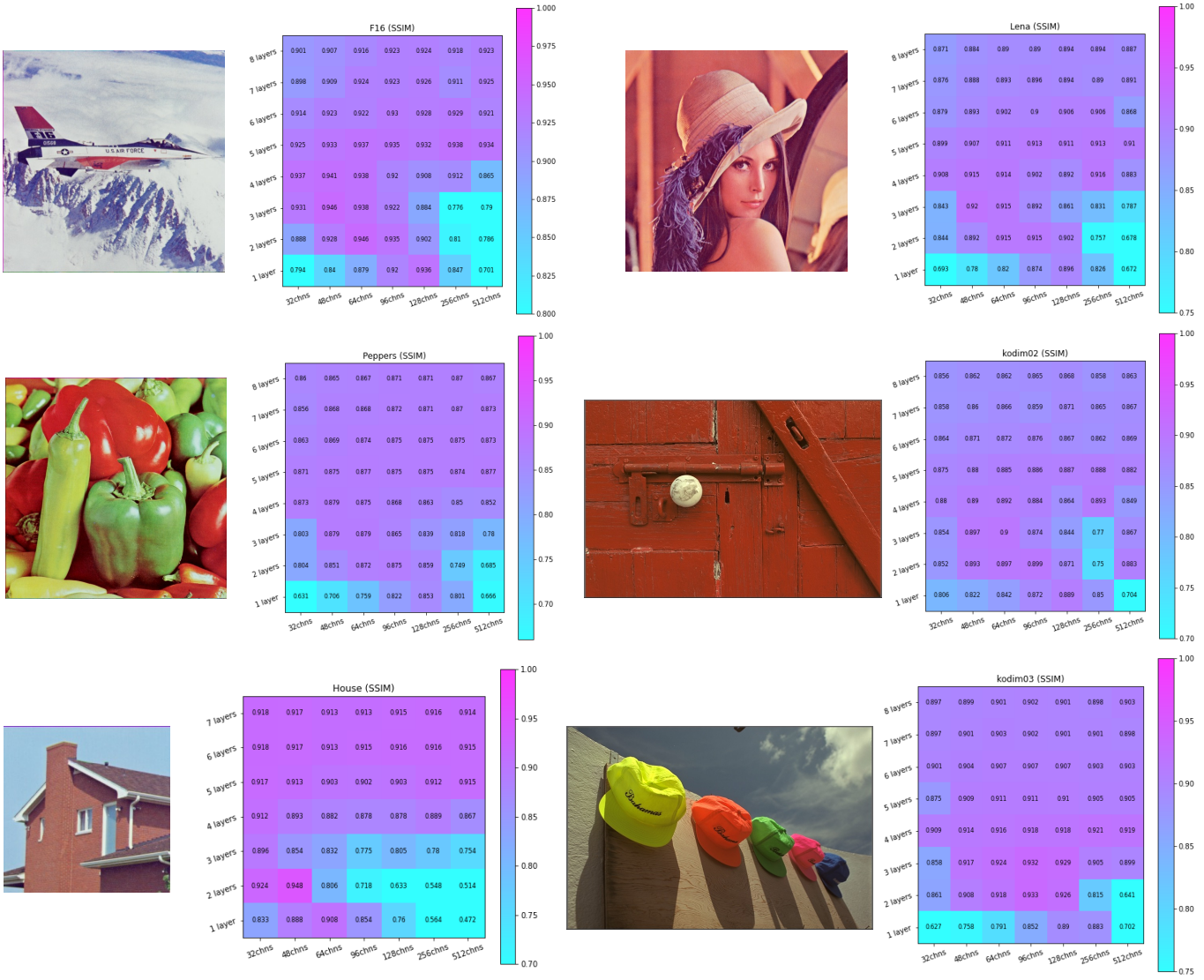


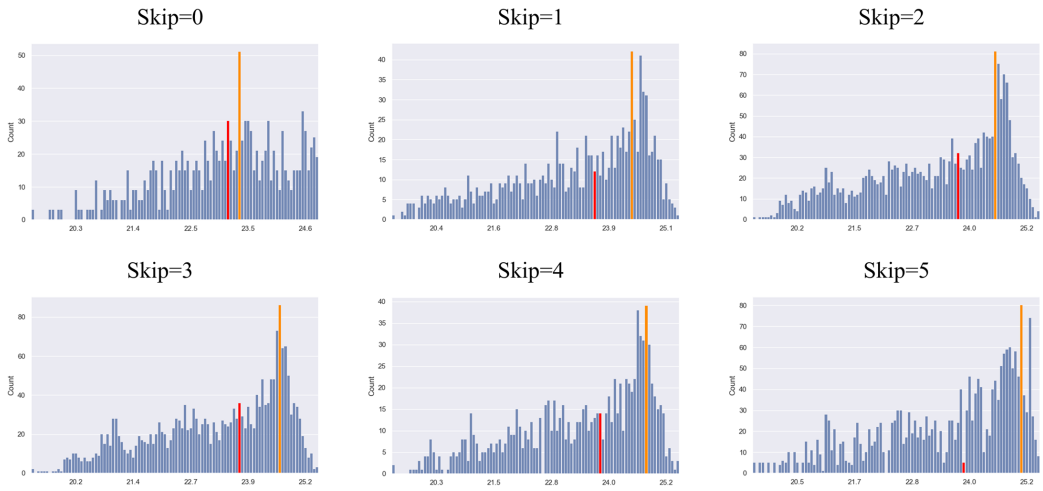
Fig. 16: SSIM (↑) values of different depth and width combinations for **coarse-grained images**. The low-pass filtering effects of the upsampling do not affect them much since they are dominated by low frequencies and most of the energy is concentrated on the center of the spectrum. Thus, they are robust to depth choices: $2 \sim 8$ layers. However, they become sensitive to width when the network is shallow (less attenuation) where over-fitting can easily occur with larger width. Thus, the high-performing architectures tend to be **narrow**. No skip connections are involved.

As shown in Fig. 15 and Fig. 16, the architectural characteristics of fine-grained and coarse-grained images are dramatically different: the high-frequency abundant images necessitate shallow and wide networks to preserve the fine details; the low-frequency dominated images are robust to a wider range of depth choices but necessitate narrower width to prevent over-fitting when the depth is low. Meanwhile, although we only have three pre-defined width choices (i.e., 32, 64, 128) for this study, it can be seen that the images are quite robust within a certain range – this means that simple averaging (e.g., 48, 96) would work well for ambiguous cases in practice.

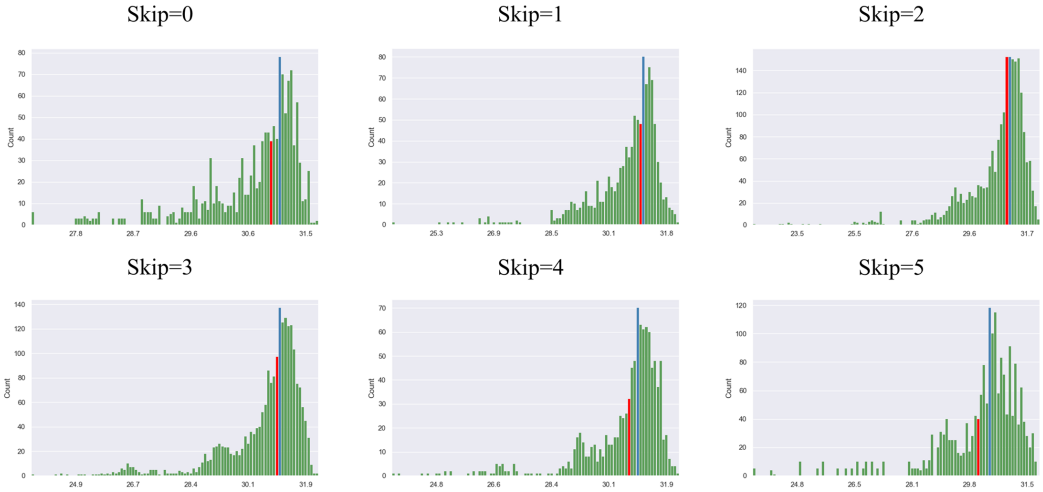
D.2 Skip Connections

Our base model is an encoder-decoder architecture with 5 layers for each. Initially, it is free of upsampling/downsampling and skip connections. We then add skip connection and upsampling (downsampling at the corresponding position on encoder) to the base model and test the performance.

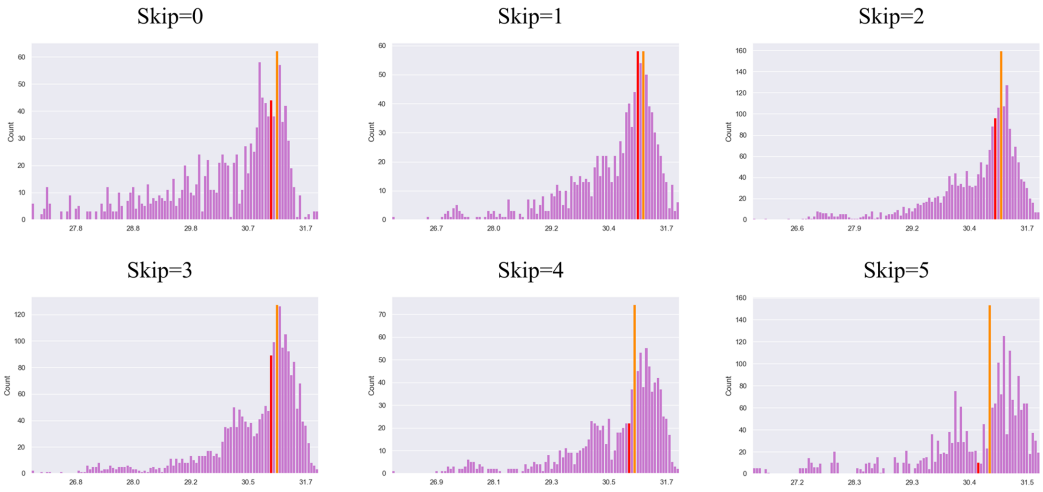
Since the skip connections are freely added to the model, there are $C_5^0+C_5^1+C_5^2+C_5^3+C_5^4+C_5^5 = 32$ diffrent skip connection patterns. The rescale (downsampling-upsampling pairs) pattern is more complicated because we not only consider $\times 2$ rescaling, but also $\times 4$, $\times 8$, $\times 16$, and $\times 32$ rescaling for one downsampling/upsampling layer. The number of possible arrangements for



(a) 'Baboon' (128-channel, fine-grained) from Set9 [3] dataset. The red line denotes the median number of models, and the orange line denotes the maximum. As more skip connections are added, an increasing number of models perform better, which may imply that skip connections reduce the actual down-/up-sampling rate and thus the low-pass filtering effects.



(b) 'F-16' (64-channel, coarse-grained) from Set9 [3] dataset, The red line denotes the median number of models, and the blue line denotes the maximum. More skip connections do not substantially benefit the coarse-grained images and may even increase the risks of over-fitting.



(c) 'House' (32-channel, coarse-grained) from Set9 [3] dataset. The red line denotes the median number of models, and the orange line denotes the maximum. On this case, more skip connections incur over-fitting more easily.

Fig. 17: The distribution of the final PSNR scores of the 7424 models

downsampling-upsampling pairs is $1 + 5 + 15 + 35 + 70 + 106 = 232$. Therefore, our experimental results are comprised of $32 \times 232 = 7424$ **combinations in total**. We ran these models on a 128-channel ("Baboon"), a 64-channel ("F16") and a 32-channel image ("House") respectively. The results are summarized in Fig. 17, Fig. 18 and 19. Overall, skip connections help enhance the performance on fine-grained images but can cause over-fitting on the coarse-grained ones, *likely* because the actual down-/up-sampling rate is reduced.

E Texture-DIP Dataset

E.1 Texture Feature Extraction

Spatial features. Texture coarseness can also be characterized by the number of neighboring pixel pairs having the same grey values, which is the basis of the Grey Level Co-occurrence Matrix (GLCM) haralick1973textural, a classic tool for texture analysis. We derive four kinds of statistics from the GLCM of each image, i.e., dissimilarity ($\sum_i \sum_j c_{ij}|i-j|$), correlation ($-\sum_i \sum_j \frac{(i-u_i)(j-u_j)c_{ij}}{\sigma_i \sigma_j}$), homogeneity ($\sum_i \sum_j \frac{c_{ij}}{1+|i-j|}$) and contrast ($\sum_i \sum_j (i-j)^2 c_{ij}$), where c_{ij} denotes the entry of the GLCM representing the co-occurrences of the pixel values i and j. The co-occurrences are measured by each metric from four angles $\{0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}\}$, leading to a total of 16 features for each image. The features are then sent to Decision Tree for further selection.

Frequency feature. Coarse texture dominated by large homogeneous spatial structures has its Power Spectral Density (PSD) concentrated on low frequencies, which drops off rapidly towards higher frequencies, while finer texture exhibits a more moderate decay. To characterize the decay trend, a straightforward way is to use the ratio of the magnitudes at lower frequencies and the magnitudes at higher frequencies. To do so, we first convert the 2D power spectrum of the image from its Cartesian coordinates $S(k,l)$ to polar coordinates, and compute its 1D representation via azimuthally averaging over θ, defined as $\hat{S}(r) = \frac{1}{2\pi}\int_0^{2\pi} S(r,\theta)d\theta$ with $r = \sqrt{k^2+l^2}$ and $\theta = \text{atan2}(k,l)$, which represents the mean magnitude of the frequencies with respect to the radial distance r. The averaging of the frequency components along circles also smooths the noise components. For the obtained spectral vector, we drop the DC power and normalize it to $[0,1]$.

E.2 Decision Tree and SVM for Width Classification

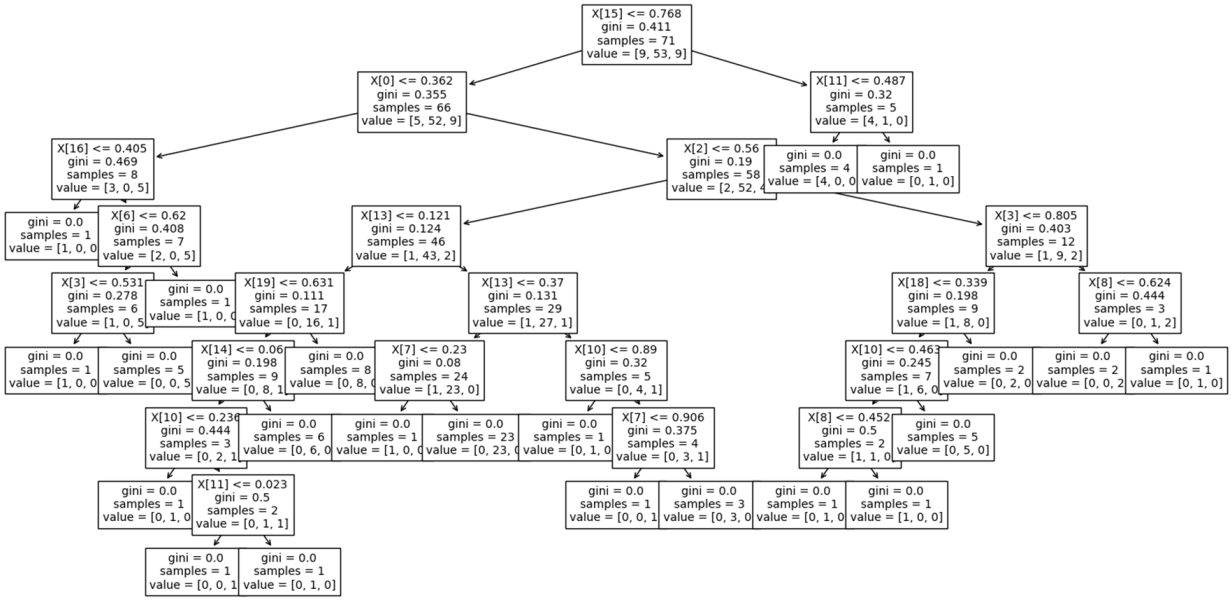


Fig. 18: A sample decision tree from a group (5) of randomized decision trees for feature selection. The input is denoted as x. The nodes starting with $x[n] \leq threshold$ are decision nodes, which splits the data most efficiently based on the features. The nodes starting with $gini = 0.0$ are end nodes, indicating a decision is reached. The construction of the decision tree is based on the Gini impurity values, which in turn determines the importance of the features.

To illustrate the correlations between width and image texture, we perform a width classification task with texture features as the inputs. We first pre-process the inputs by performing feature selection. Specifically, we used the Extra Trees Classifier

[5]. Similar to Random Forests, Extra Trees is an ensemble machine learning approach that trains a group of decision trees and aggregates the results of them to make the final prediction. Compared to deep learning, the Extra Trees Classifier is more explainable, has lower computational costs and can reduce bias because the Extra Trees Classifier samples the whole dataset when constructing the trees while the Random Forests Classifier subsamples the inputs at a time.

Implementation details. The feature selection is performed in the form of 5-fold cross-validation. In each fold, the Extra Trees Classifier fits the training set and tests on the held-out set. For dividing a decision tree, it selects the best features based on Gini impurity, which measures the purity of the sub-split and can be computed as below:

For a classification problem with K outcomes and n samples, $p_i = \frac{1}{n}\sum_{y \in Q_n} I(y = k)$,

$$GI = 1 - \sum_{i=1}^{n} p_i^2 \tag{1}$$

We then compute the normalized total reduction of the Gini impurity values as the Gini Importance of the feature. The higher the Gini Importance value, the more important the feature. We selected the top 5 features for classification.

Since the width classification is multi-class, we employ the one-vs-all classification strategy. Specifically, we use three SVMs with SGD optimization to classify 32-chns images against the rest, 64-chns images against the rest, and 128-chns images against the rest. We fed the 5 selected features into the SVMs. Each SVM outputs a binary prediction indicating whether or not the input belongs to the specific width choice. We repeat 10 times of the 5-fold cross-validation to rule out the contingency of the training-testing set partition.

E.3 Overview of Our Dataset

We re-classify the images from the popular Set9 [3], Set12 [9] and CBSD68 [6] datasets into three width choices $\{32, 64, 128\}$, and show the correlations between the width and the complexity of the image texture by performing the width classification task as described above. The AUC scores are shown in the main text. As some images may be robust to more than one width choice, there are two versions of our dataset, one with repeated images and one without.

Fig. 19: **Example images from our Texture-DIP dataset.** Images from the **1st** and **2nd** rows correspond to the width choices of 32 and 64, respectively. Images from the **3rd** row correspond to the width choice of 128 and are considered to be **fine-grained**. The images are originally from Set9 [3], Set12 [9] and CBSD68 [6].

F More Qualitative Results

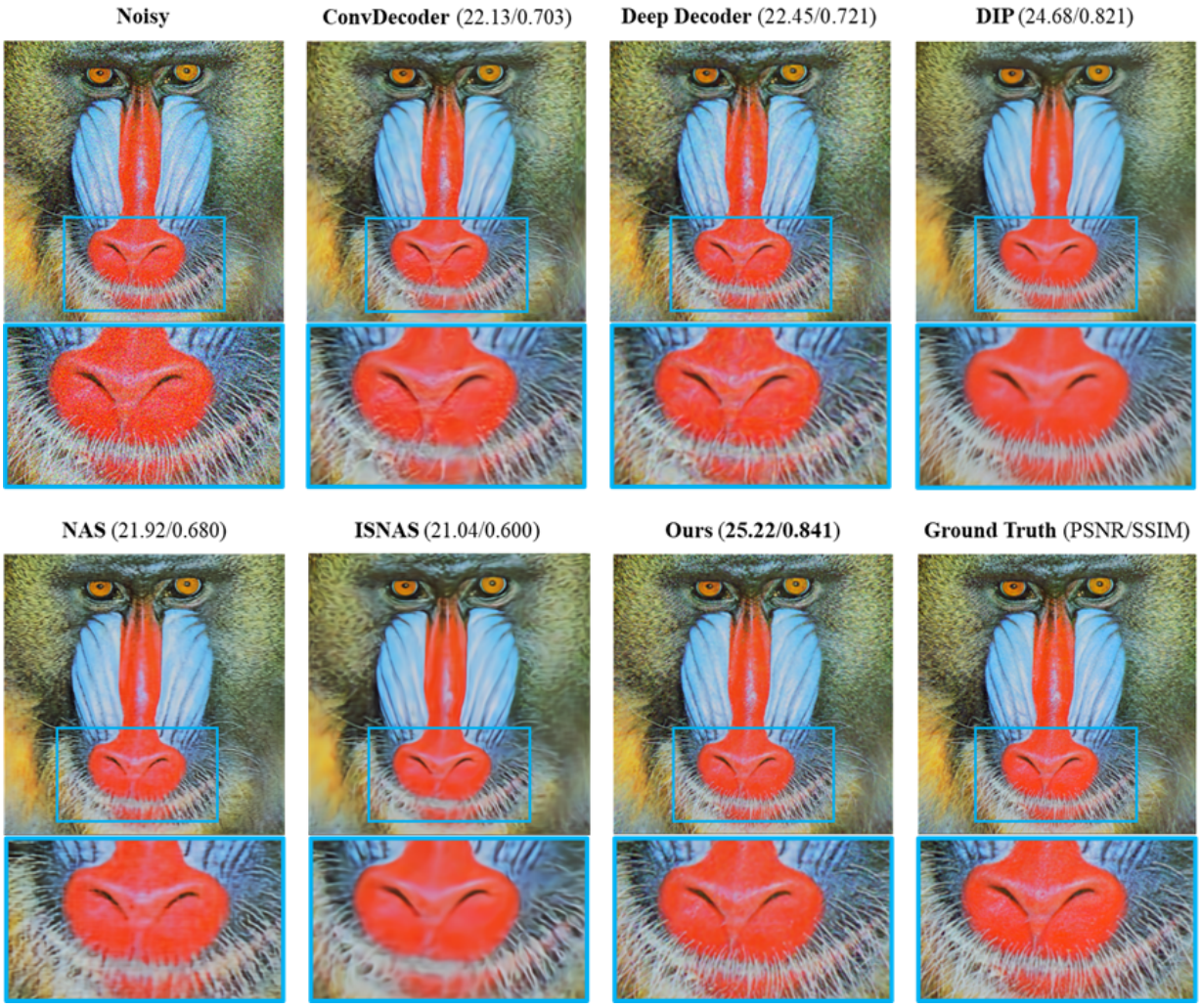


Fig. 20: Qualitative results for 'Baboon' (an **extremely fine-grained** image where current methods typically fall short) from Set9 [3] dataset.

References

1. Metin Ersin Arican, Ozgur Kara, Gustav Bredell, and Ender Konukoglu. Isnas-dip: Image-specific neural architecture search for deep image prior. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1960–1968, 2022.
2. Yun-Chun Chen, Chen Gao, Esther Robb, and Jia-Bin Huang. Nas-dip: Learning deep image prior with neural architecture search. In *European Conference on Computer Vision*, pages 442–459. Springer, 2020.
3. Kostadin Dabov, Alessandro Foi, and Karen Egiazarian. Video denoising by sparse 3d transform-domain collaborative filtering. In *2007 15th European Signal Processing Conference*, pages 145–149. IEEE, 2007.
4. Mohammad Zalbagi Darestani and Reinhard Heckel. Accelerated mri with un-trained neural networks. *IEEE Transactions on Computational Imaging*, 7:724–733, 2021.
5. Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63:3–42, 2006.
6. Stefan Roth and Michael J Black. Fields of experts. *International Journal of Computer Vision*, 82(2):205–229, 2009.
7. Zenglin Shi, Pascal Mettes, Subhransu Maji, and Cees GM Snoek. On measuring and controlling the spectral bias of the deep image prior. *International Journal of Computer Vision*, 130(4):885–908, 2022.
8. Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9446–9454, 2018.
9. Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE transactions on image processing*, 26(7):3142–3155, 2017.