Deep Image Prior
Supplementary Material

In this supplemental material we provide more details about the setup in each experiment. We also extend the figures from the main paper with more qualitative comparisons. The code for all experiments is published at `https://github.com/DmitryUlyanov/deep-image-prior`. Interactive comparison is available at project page `https://dmitryulyanov.github.io/deep_image_prior`.

1. Architectures

In general we found that deep architectures *without* too many short paths from input to output define very good deep priors. Having too many short paths prevent learning regular image patterns, resulting in memorization at the pixel level and deficient image priors.

While other options are possible, we mainly experimented with fully-convolutional architectures, where the input $z \in \mathbb{R}^{C' \times W \times H}$ has the same spatial resolution as the the output of the network $f_\theta(z) \in \mathbb{R}^{3 \times W \times H}$.

We use encoder-decoder ("hourglass") architecture (possibly with skip-connections) for f_θ in all our experiments (fig. 1), varying small number of hyperparameters. Although the best results can be achieved by carefully tuning an architecture for each task (and potentially for each image) independently, we found wide range of hyperparameters and architectures to give acceptable results.

We use LeakyReLU [4] as a non-linearity. As a downsampling technique we simply use strides implemented in convolution module. We also tried average/max pooling and downsampling with Lanczos kernel, but did not find a consistent difference between any of them. As an upsampling operation we choose between bilinear upsampling and nearest neighbour upsampling. An alternative upsampling method could be to use transposed convolutions, but the results we obtained using them were worse. We use reflection padding instead of zero padding in convolution layers everywhere except for feature inversion experiment.

We considered two ways to create the input z: 1. *random*, where the z is filled with uniform noise between zero and 0.1, 2. *meshgrid*, where we initialize $z \in \mathbb{R}^{2 \times W \times H}$ using `np.meshgrid` (see fig. 2). Such initialization serves as an additional smoothness prior to the one imposed by the structure of f_θ itself. We found such input to be beneficial for large-hole inpainting.

During fitting of the networks we sometimes use a noise-based regularization. I.e. at each iteration we perturb the input z with an additive normal noise. While we have found such regularization to impede optimization process, we also observed that the network was able to optimize it's objective to zero no matter the variance of the additive noise (i.e. the network was always able to adapt to any reasonable variance for sufficiently large number of optimization steps). Finally, we use *ADAM* optimizer [5] in all our experiments. We implement everything in PyTorch.

Below, we provide the remaining details of the used architectures that were application-specific. We use the notation introduced in fig. 1.

After that, a large number of additional experimental results are provided. *Electronic zoom-in is recommended for almost all figures.*

Super-resolution (fig. 1 and 5 of main paper).

$z \in \mathbb{R}^{32 \times W \times H} \sim U(0, \frac{1}{10})$
$n_u = n_d =$ `[128, 128, 128, 128, 128]`
$k_u = k_d =$ `[3, 3, 3, 3, 3]`
$n_s =$ `[4, 4, 4, 4, 4]`
$k_s =$ `[1, 1, 1, 1, 1]`
$\sigma_p = \frac{1}{30}$
`num_iter` $= 2000$
`LR` $= 0.01$
`upsampling` $=$ `bilinear`

The decimation operator d is composed of low pass filtering operation using Lanczos2 kernel (see [9]) and resampling, all implemented as a single (fixed) convolutional layer.

For `8x` super-resolution (9) we only changed to std. of input noise to $\sigma_p = \frac{1}{20}$ and number of iterations to 4000.

Text inpainting (fig. 7 top row of main paper).

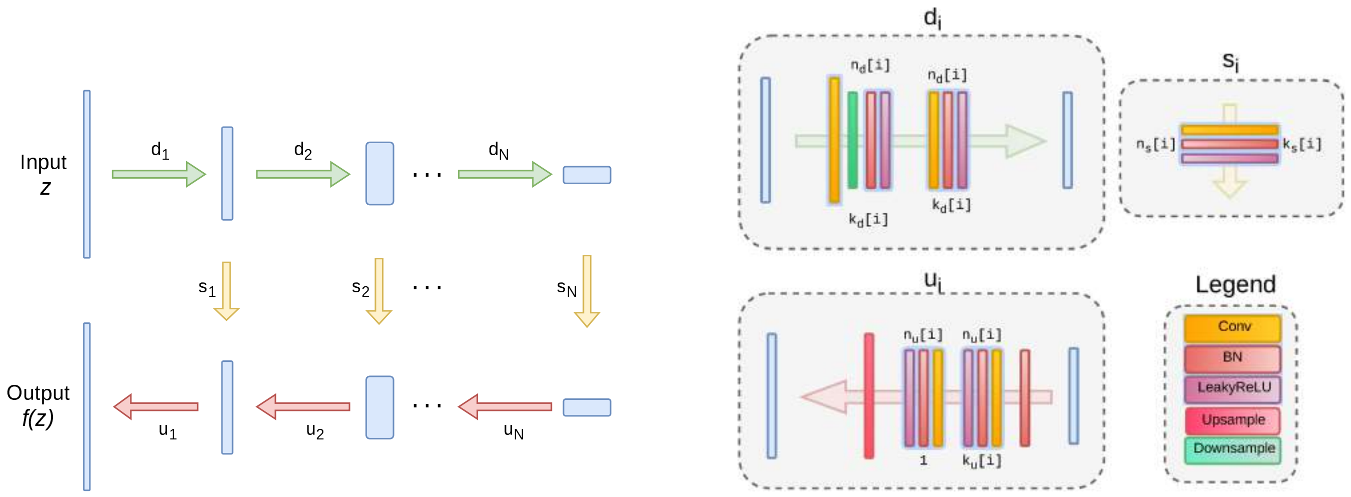


Figure 1: **The architecture used in the experiments.** We use "hourglass" (also known as "decoder-encoder") architecture. We sometimes add skip connections (yellow arrows). $n_u[i]$, $n_d[i]$, $n_s[i]$ correspond to the number of filters at depth i for the upsampling, downsampling and skip-connections respectively. The values $k_u[i]$, $k_d[i]$, $k_s[i]$ correspond to the respective kernel sizes.

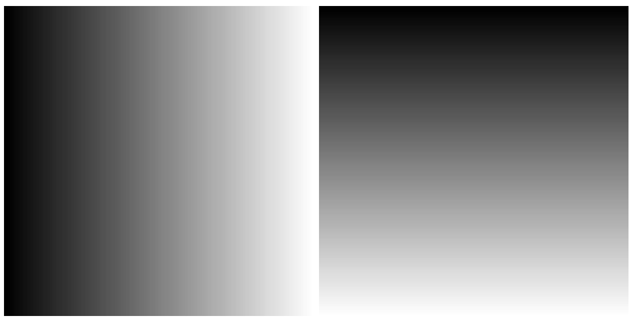


Figure 2: **"Meshgrid" input** z used in some inpainting experiments. The intensity encodes the value: from zero (black) to one (white).

$z \in \mathbb{R}^{32 \times W \times H} \sim U(0, \frac{1}{10})$
$n_u = n_d =$ `[16, 32, 64, 128, 128]`
$k_u = k_d =$ `[7, 7, 7, 7, 7]`
$n_s =$ `[0, 0, 0, 0, 4]`
$k_s =$ `[NA, NA, NA, NA, 1]`
$\sigma_p = 0$
`num_iter` $= 3000$
`LR` $= 0.01$
`upsampling` $=$ `nearest`

Large hole inpainting (fig. 6 of main paper).

$z =$ `np.meshgrid(W,H)`
$n_u = n_d =$ `[16, 32, 64, 128, 128]`
$k_u = k_d =$ `[7, 7, 7, 7, 7]`
$n_s =$ `[0, 0, 0, 0, 0]`
$k_s =$ `[NA, NA, NA, NA, NA]`
$\sigma_p = 0$
`num_iter = 5000`
`LR = 0.1`
`upsampling` $=$ `nearest`

Large hole inpainting (fig. 8 of main paper).

$z \in \mathbb{R}^{32 \times W \times H} \sim U(0, \frac{1}{10})$
$n_u = n_d =$ `[16, 32, 64, 128, 128, 128]`
$k_d =$ `[3, 3, 3, 3, 3, 3]`
$k_u =$ `[5, 5, 5, 5, 5, 5]`
$n_s =$ `[0, 0, 0, 0, 0, 0]`
$k_s =$ `[NA, NA, NA, NA, NA, NA]`
$\sigma_p = 0$
`num_iter = 5000`
`LR = 0.1`
`upsampling` $=$ `nearest`

We simply sliced off last layers to get smaller depth.

Denoising.

$z \in \mathbb{R}^{32 \times W \times H} \sim U(0, \frac{1}{10})$
$n_u = n_d =$ `[128, 128, 128, 128, 128]`
$k_u = k_d =$ `[3, 3, 3, 3, 3]`
$n_s =$ `[4, 4, 4, 4, 4]`
$k_s =$ `[1, 1, 1, 1, 1]`
$\sigma_p = \frac{1}{30}$
`num_iter = 1800`
`LR = 0.01`
`upsampling = bilinear`

We used the following implementations of referenced denoising methods: [6] for CBM3D and [1] for NLM.

We use exponential sliding window with weight $\gamma = 0.99$.

JPEG artifacts removal (fig. 3) $C' = 3$, $LR = 0.01$.

$z \in \mathbb{R}^{3 \times W \times H} \sim U(0, \frac{1}{10})$
$n_u = n_d =$ `[8, 16, 32, 64, 128]`
$k_u = k_d =$ `[3, 3, 3, 3, 3]`
$n_s =$ `[0, 0, 0, 4, 4]`
$k_s =$ `[NA, NA, NA, 1, 1]`
$\sigma_p = \frac{1}{30}$
`num_iter = 2400`
`LR = 0.01`
`upsampling = bilinear`

Image reconstruction (fig. 7 bottom) $C' = 3$, $LR = 0.01$.

$z \in \mathbb{R}^{3 \times W \times H} \sim U(0, \frac{1}{10})$
$n_u =$ `[4]`
$n_u =$ `[128]`
$k_u = k_d =$ `[5]`
$n_s =$ `[32]`
$k_s =$ `[7]`
$\sigma_p = 0$
`num_iter = 3000`
`LR = 0.01`
`upsampling = bilinear`

We optimized both w.r.t. input z and parameters θ reinitializing θ every 300 iterations.

Alexnet inversion.

$z \in \mathbb{R}^{32 \times W \times H} \sim U(0, \frac{1}{10})$
$n_u = n_d =$ `[16, 32, 64, 128, 128, 128]`
$k_u = k_d =$ `[7, 7, 5, 5, 3, 3]`
$n_s =$ `[4, 4, 4, 4, 4]`
$k_s =$ `[1, 1, 1, 1, 1]`
`num_iter = 3100`
`LR = 0.001`
`upsampling = nearest`

We used `num_iter = 10000` for VGG inversion experiment (16)

References

[1] A. Buades. NLM demo. http://demo.ipol.im/demo/bcm_non_local_means_denoising/. 3

[2] A. Dosovitskiy and T. Brox. Inverting convolutional networks with convolutional networks. In *Proc. CVPR*, 2016. 13

[3] D. Glasner, S. Bagon, and M. Irani. Super-resolution from a single image. In *ICCV*, pages 349–356. IEEE Computer Society, 2009. 5, 6, 7

[4] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 1

[5] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. 1

[6] M. Lebrun. BM3D code. https://github.com/gfacciol/bm3d. 3

[7] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *Proc. CVPR*, 2015. 13

[8] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 13

[9] K. Turkowski. Filters for common resampling-tasks. *Graphics gems*, pages 147–165, 1990. 1

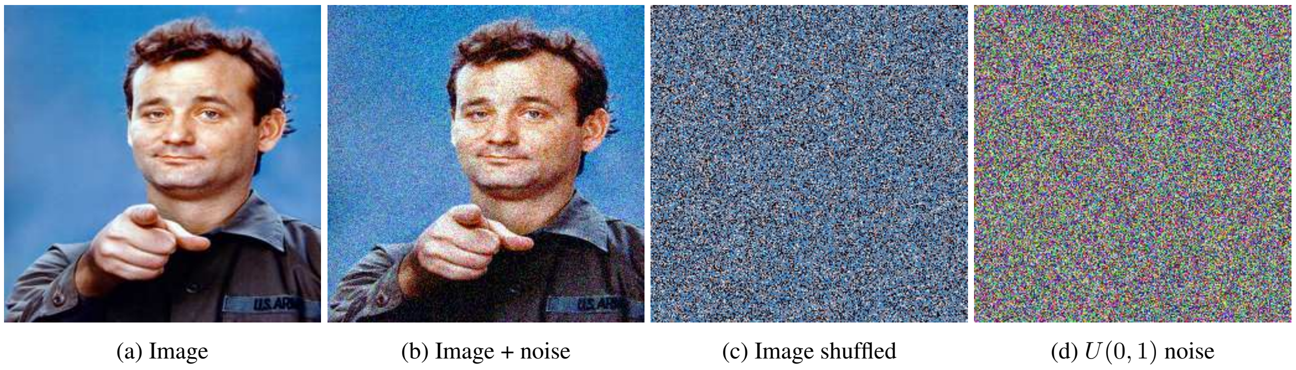

(a) Image (b) Image + noise (c) Image shuffled (d) $U(0, 1)$ noise

Figure 3: Images used in fig. 2 of the main paper to demonstrate the noise impedance effect.

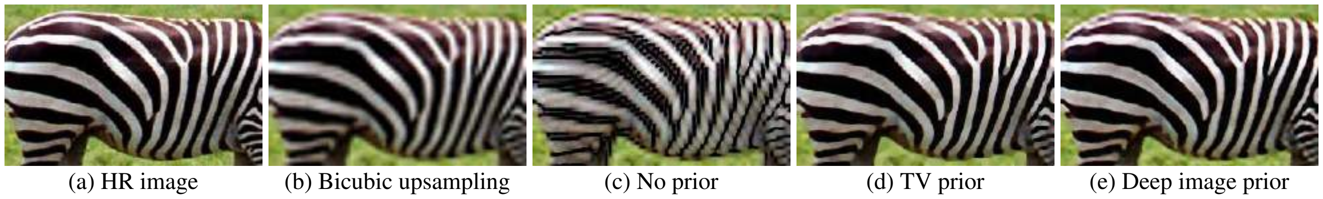

(a) HR image (b) Bicubic upsampling (c) No prior (d) TV prior (e) Deep image prior

Figure 4: **Prior effect in super-resolution. (c):**Direct optimization of data term $E(x; x_0)$ with respect to the pixels leads to ringing artifacts. TV prior removes ringing artifacts but introduces cartoon effect. Deep prior leads to the result that is both clean and sharp.

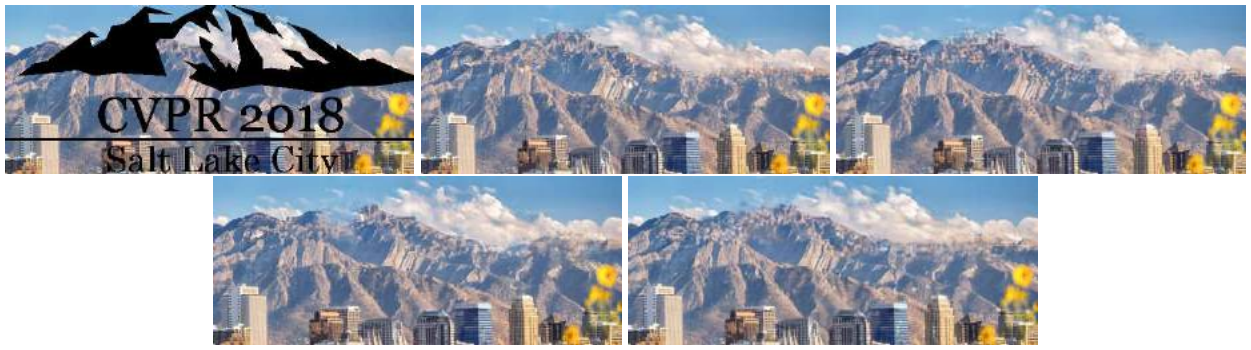
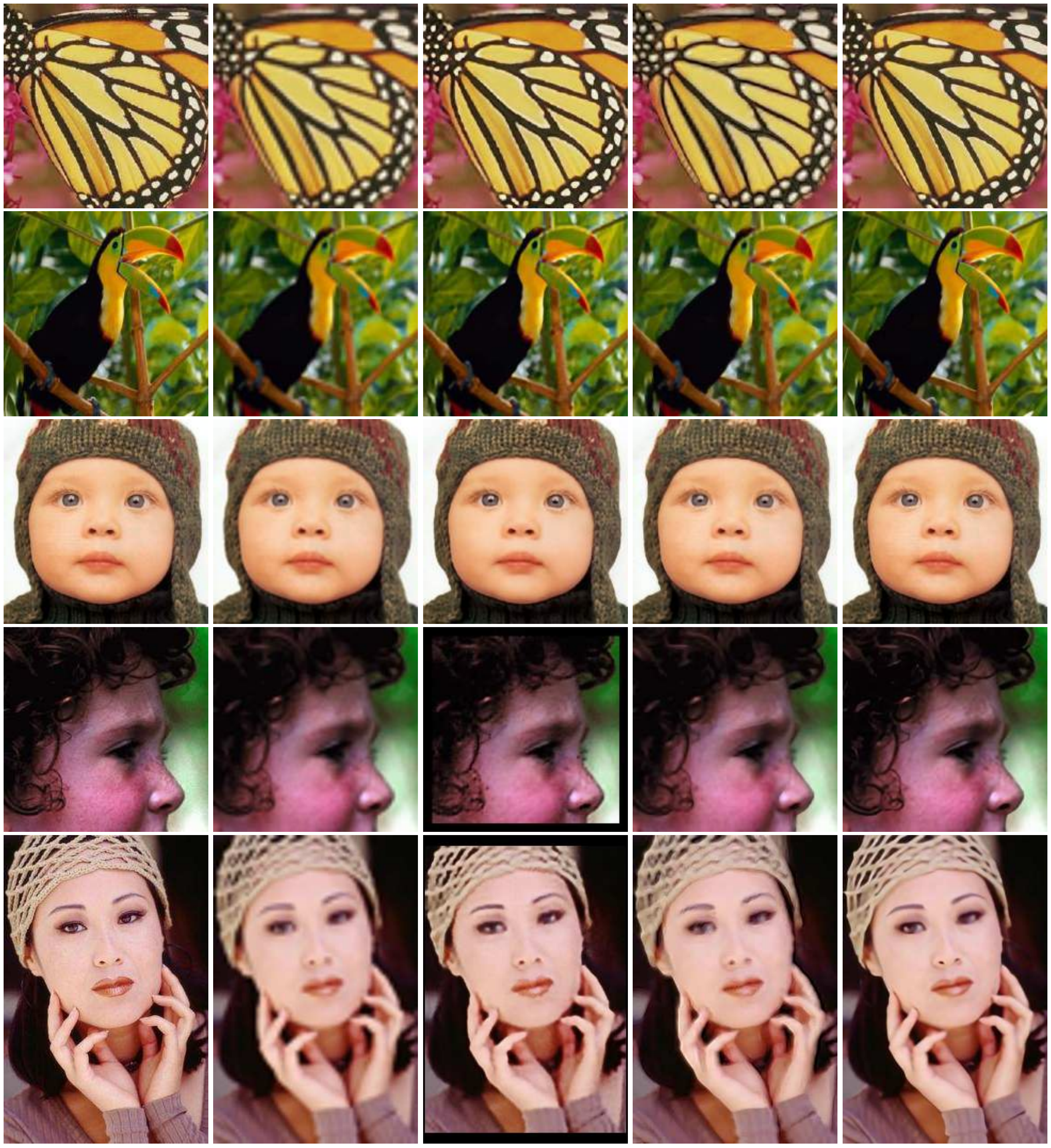


Figure 5: **Inpainting diversity.** Left: original image (black pixels indicate holes). The remaining four images show results obtained using deep prior corresponding to different input vector z.



(a) Original image (b) Bicubic, **Not trained** (c) Ours, **Not trained** (d) Glasner [3], **Not trained** (e) SRResNet, **Trained**

Figure 6: **4x image super-resolution.** Comparison on the Set5 dataset. In our experiments we center cropped the images to make spatial dimensions divisible by 32 (resulting in the black frames in the plots that can be removed with more sophisticated padding schemes).

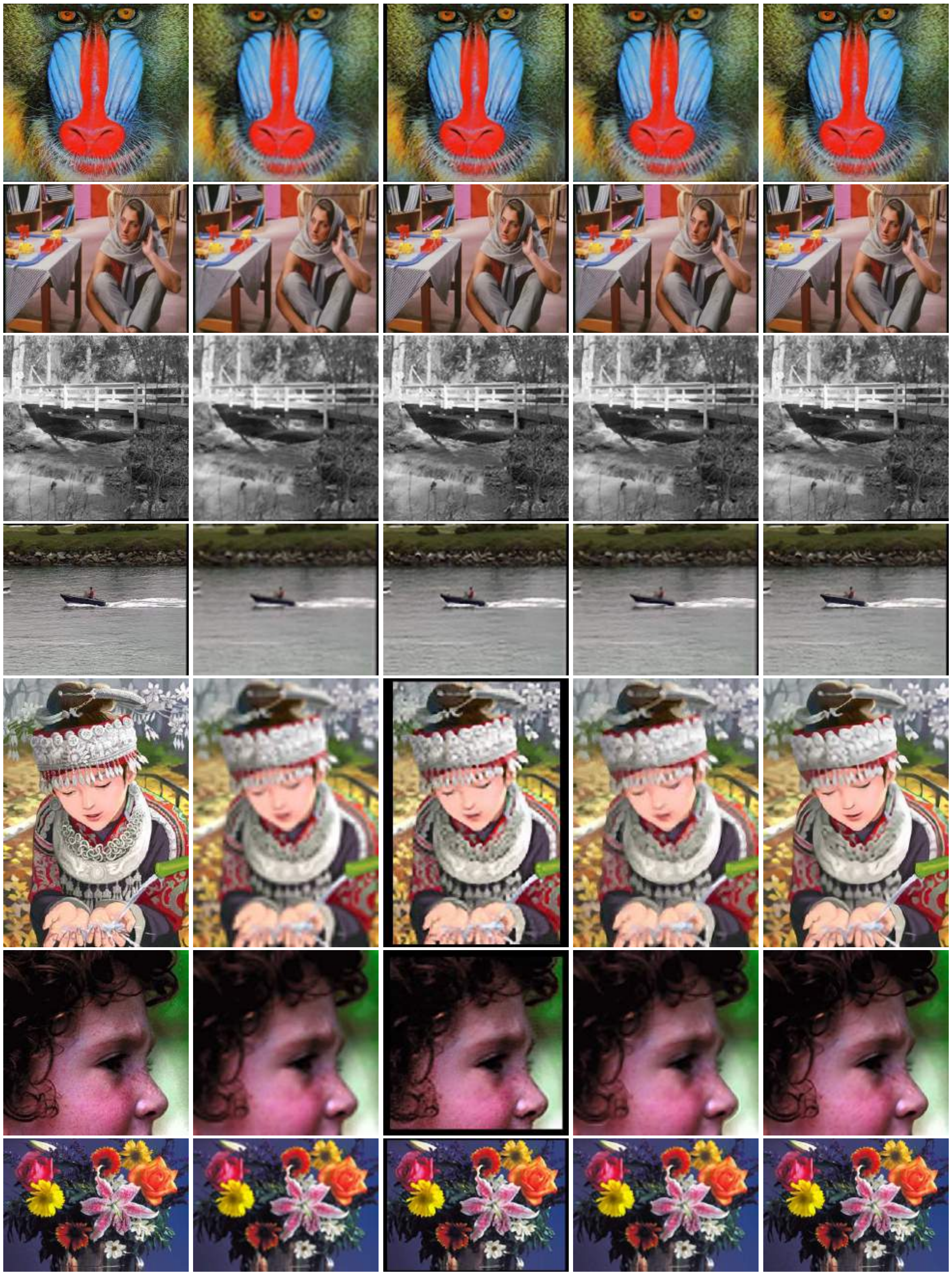

(a) Original image
(b) Bicubic, **Not trained**
(c) Ours, **Not trained**
(d) Glasner [3], **Not trained**
(e) SRResNet, **Trained**

Figure 7: **4x image super-resolution.** Comparison on the Set14 dataset. Part 1.

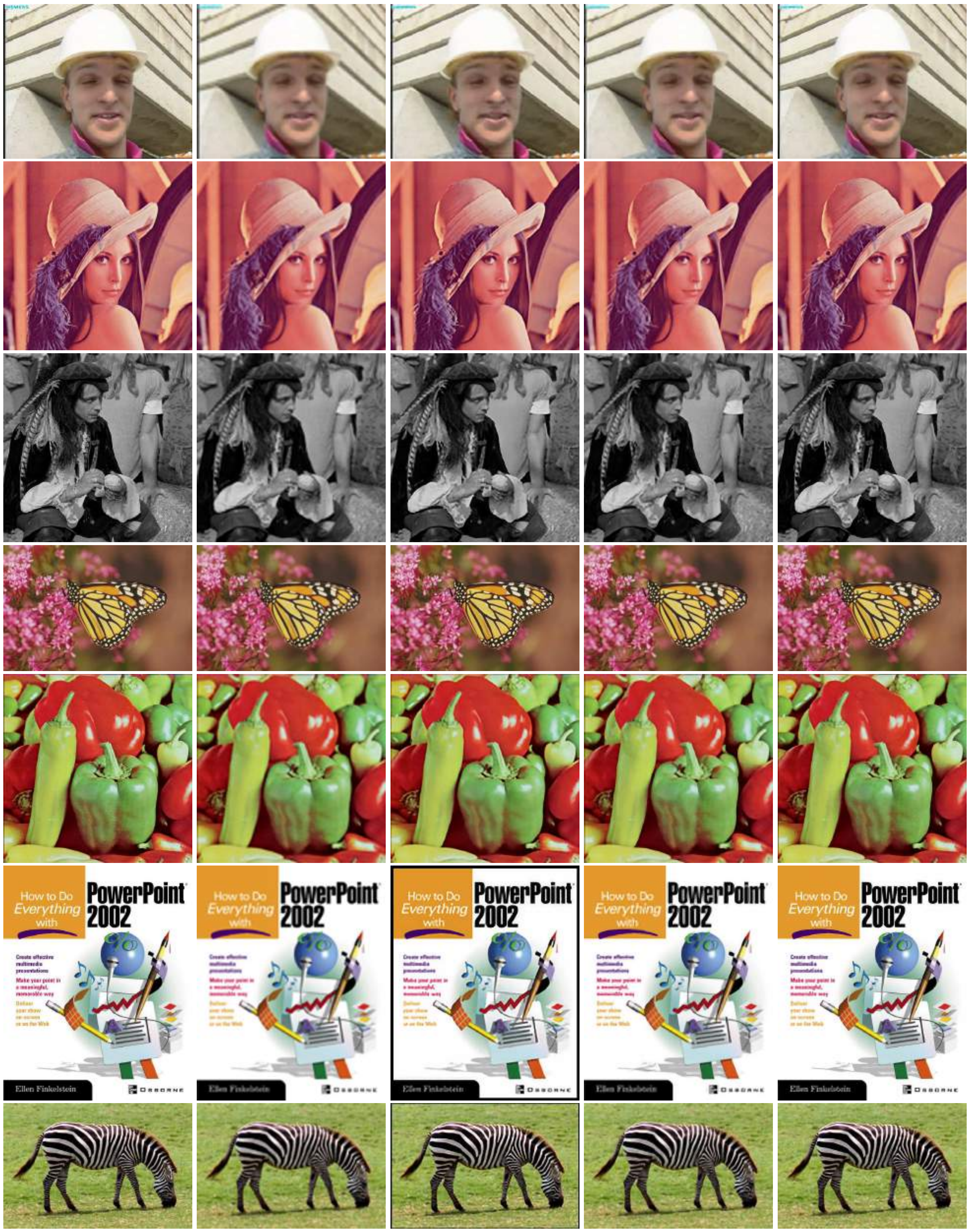

(a) Original image (b) Bicubic, **Not trained** (c) Ours, **Not trained** (d) Glasner [3], **Not trained** (e) SRResNet, **Trained**

Figure 8: **4x image super-resolution.** Comparison on the Set14 dataset. Part 2.

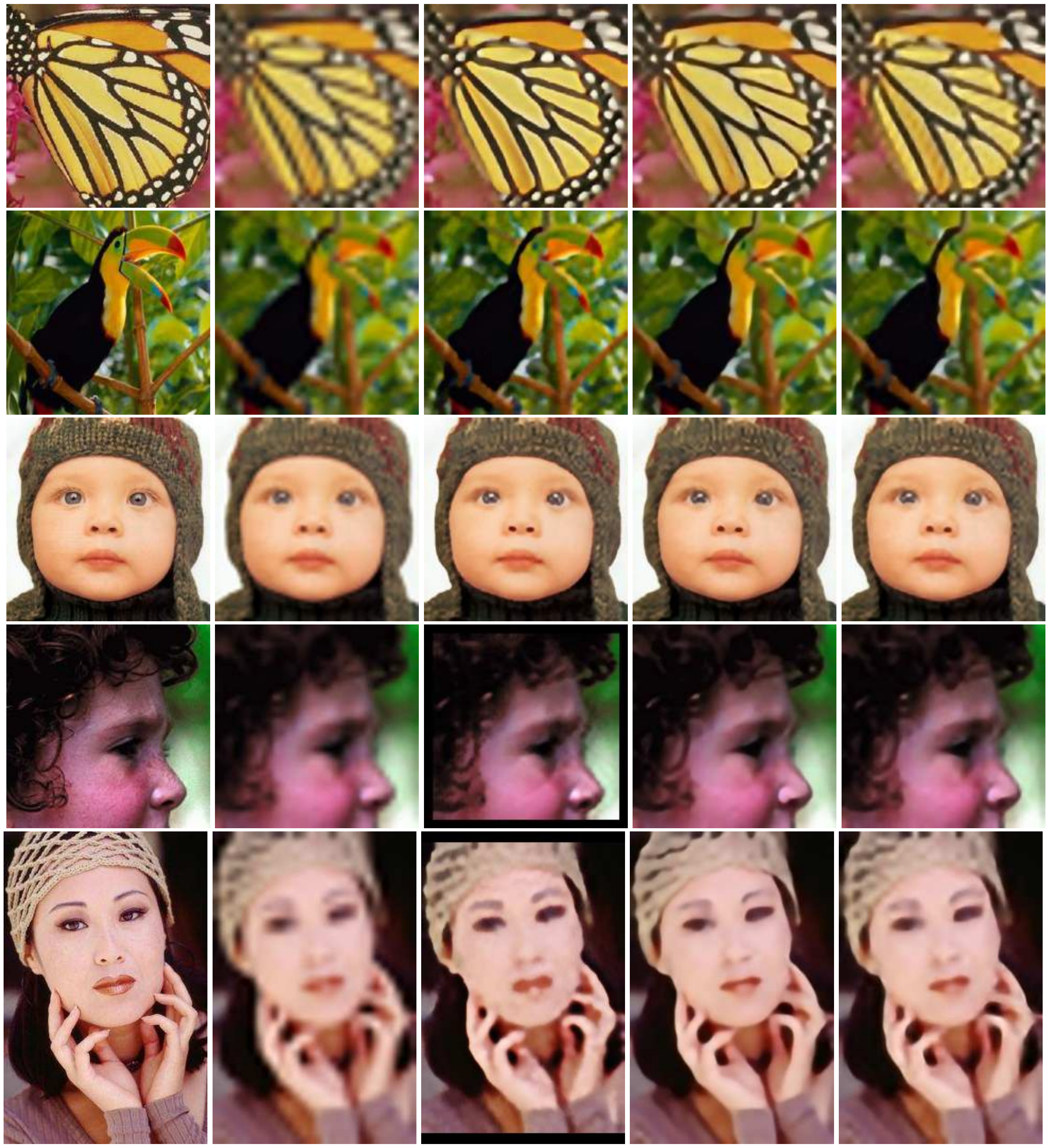
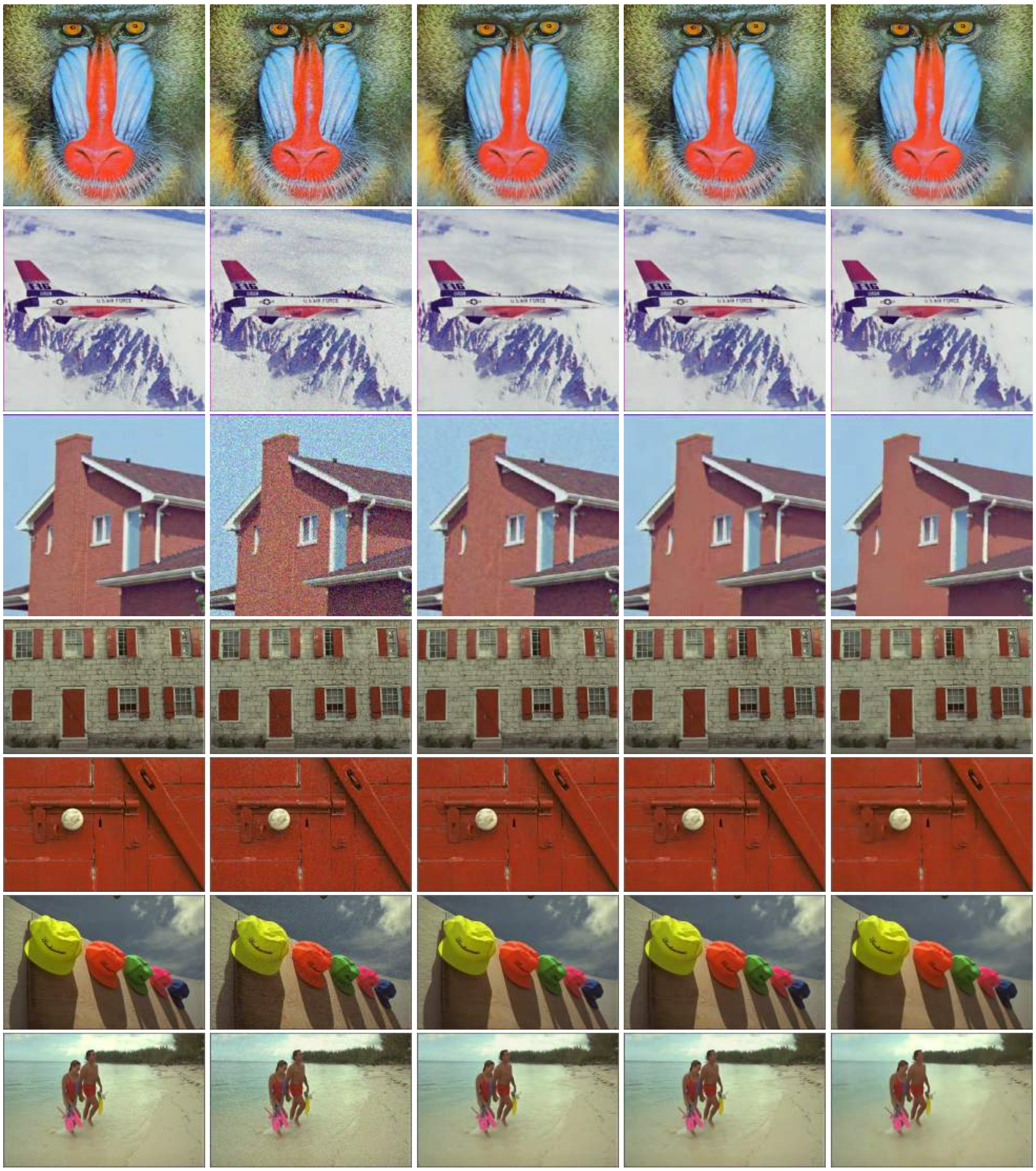

(a) Original image (b) Bicubic, **Not trained** (c) Ours, **Not trained** (d) LapSRN, **trained** (e) VDSR, **Trained**

Figure 9: **8x image super-resolution.** Comparison on the Set5 dataset.



(a) Original image (b) Noisy image (c) Ours (d) CBM3D (e) Non-local means

Figure 10: **Denoising,** $\sigma = 25$**.** Part 1.

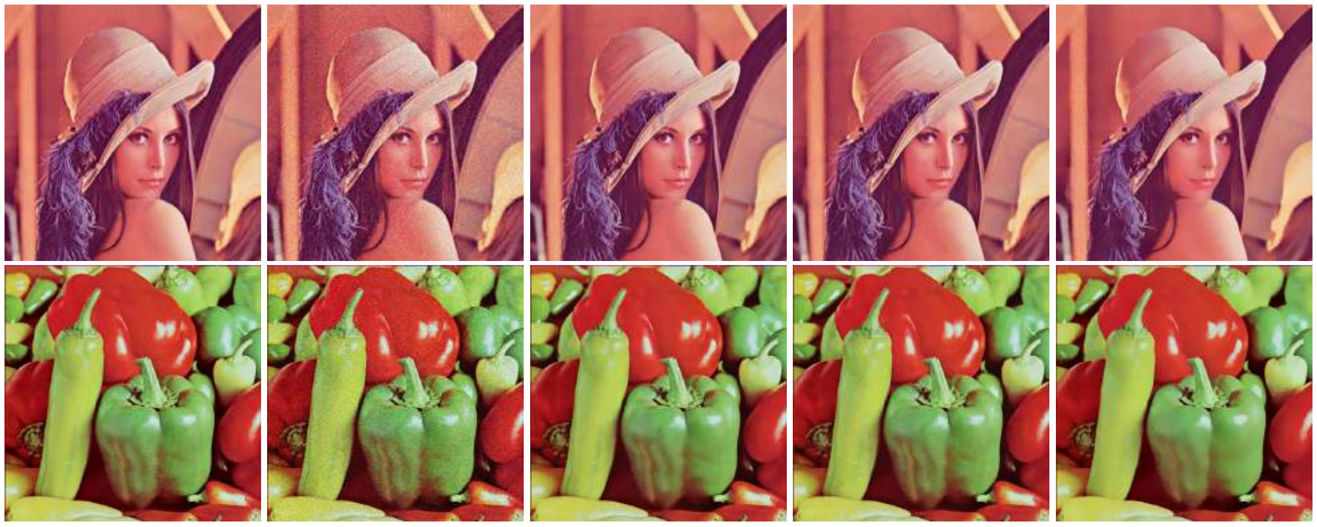

(a) Original image (b) Noisy image (c) Ours (d) CBM3D (e) Non-local means

Figure 11: **Denoising,** $\sigma = 25$**.** Part 2.

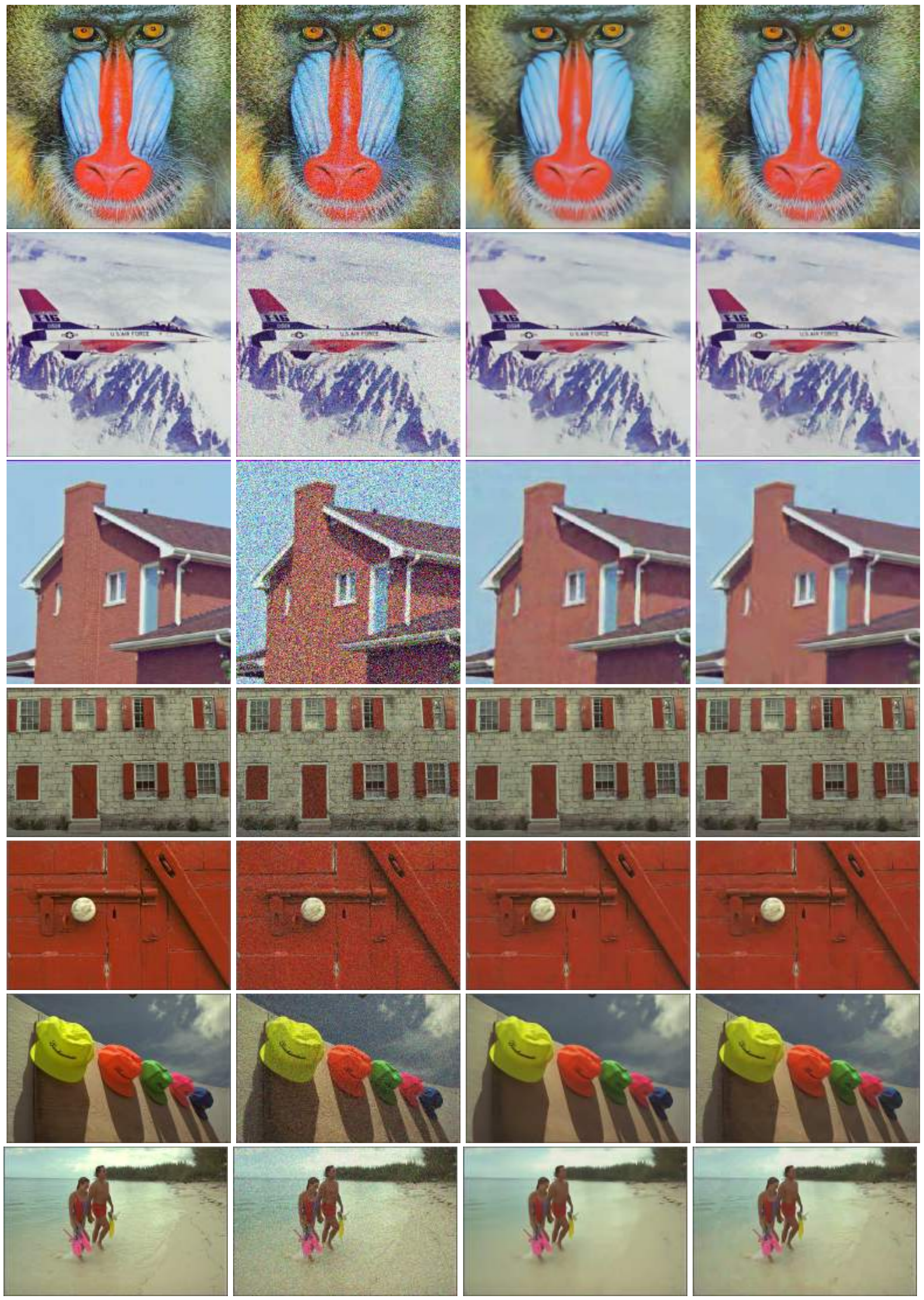
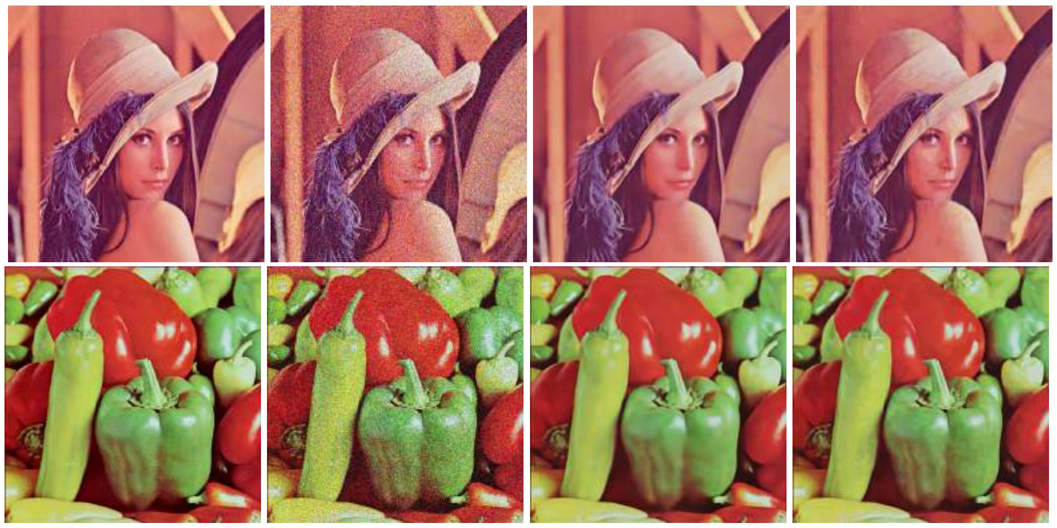
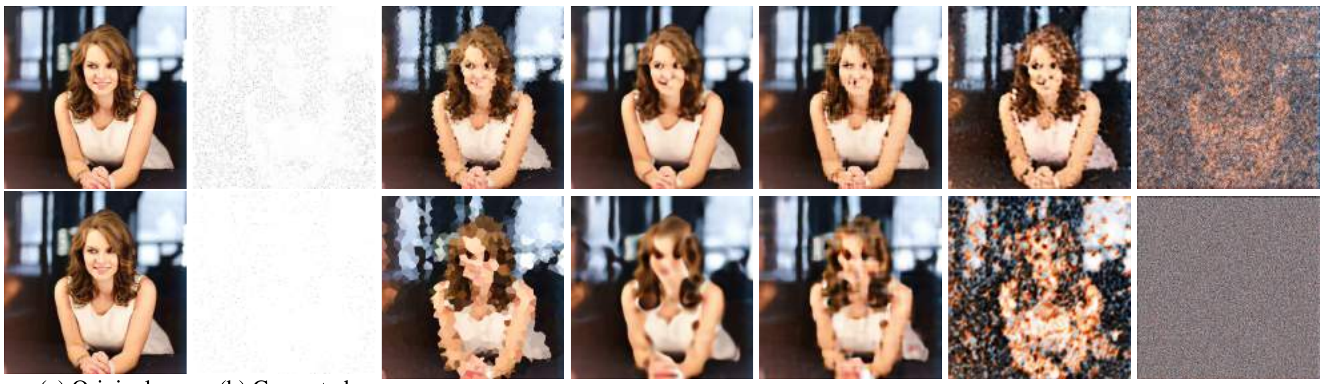

(a) Original image (b) Noisy image (c) Ours (d) CBM3D

Figure 12: **Denoising results on the standard benchmark set,** $\sigma = 50$**.** Part 1.

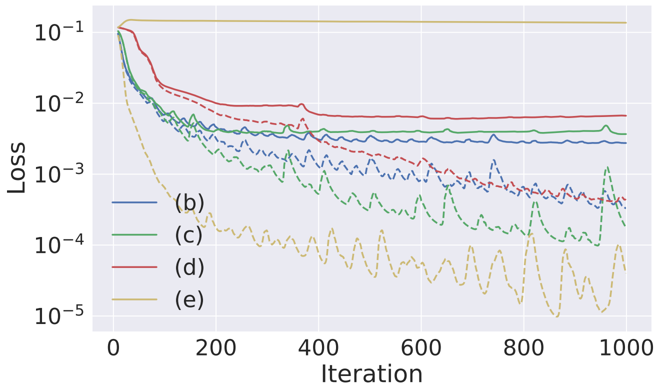


(a) Original image (b) Noisy image (c) Ours (d) CBM3D

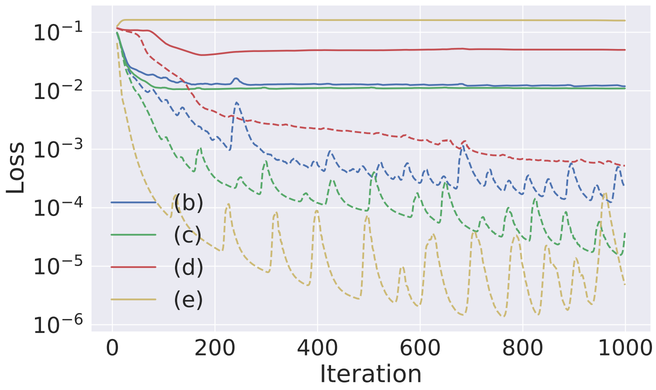
Figure 13: **Denoising results on the standard benchmark set, $\sigma = 50$.** Part 2.



(a) Original (b) Corrupted (c) Voronoi diagram (d) Hourglass bilinear upsampling (e) Hourglass nearest upsampling (f) U-Net (g) ResNet



(h) Loss/MSE curves for the top row (2% pixels known)



(i) Loss/MSE curves for the bottom row (0.5% pixels known)

Figure 14: **Reconstruction using small number of pixels.** We reconstruct 'Kate' image from 2% and 0.05% of pixels in the first and the second rows respectively. Different architectures lead to different inpainting results, yet each converges to a local minima of the data term (dashed line). Solid line shows MSE error of the generated image compared to the original image.

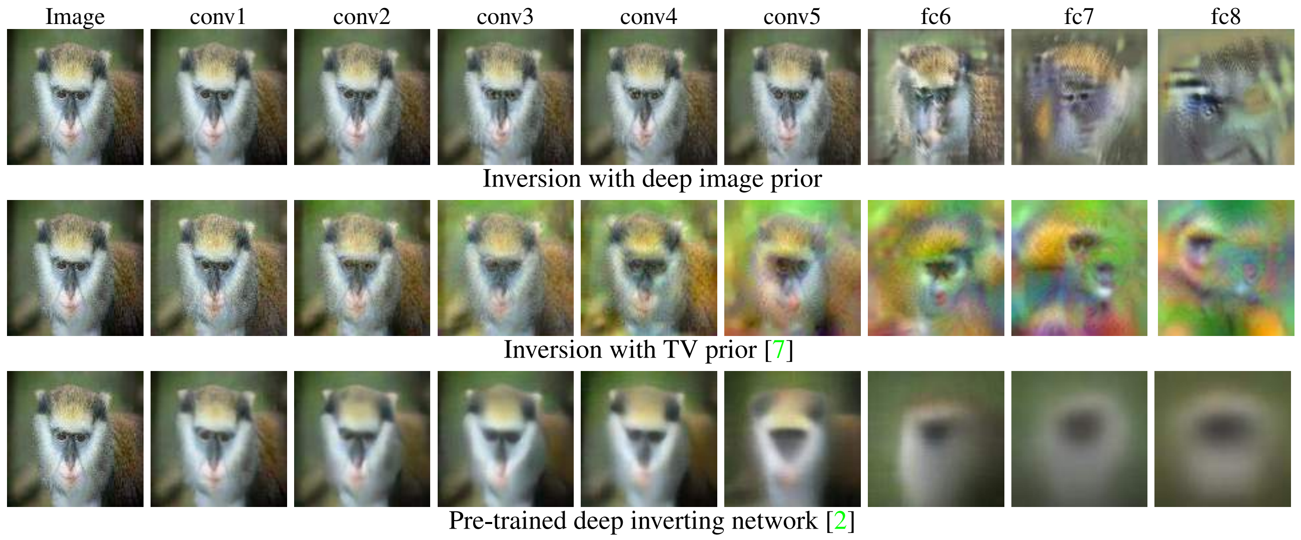


Image
conv1
conv2
conv3
conv4
conv5
fc6
fc7
fc8
Inversion with deep image prior
Inversion with TV prior [7]
Pre-trained deep inverting network [2]

Figure 15: **Inversion of AlexNet activations at different layers with different priors (see main text for the discussion of the problem).**

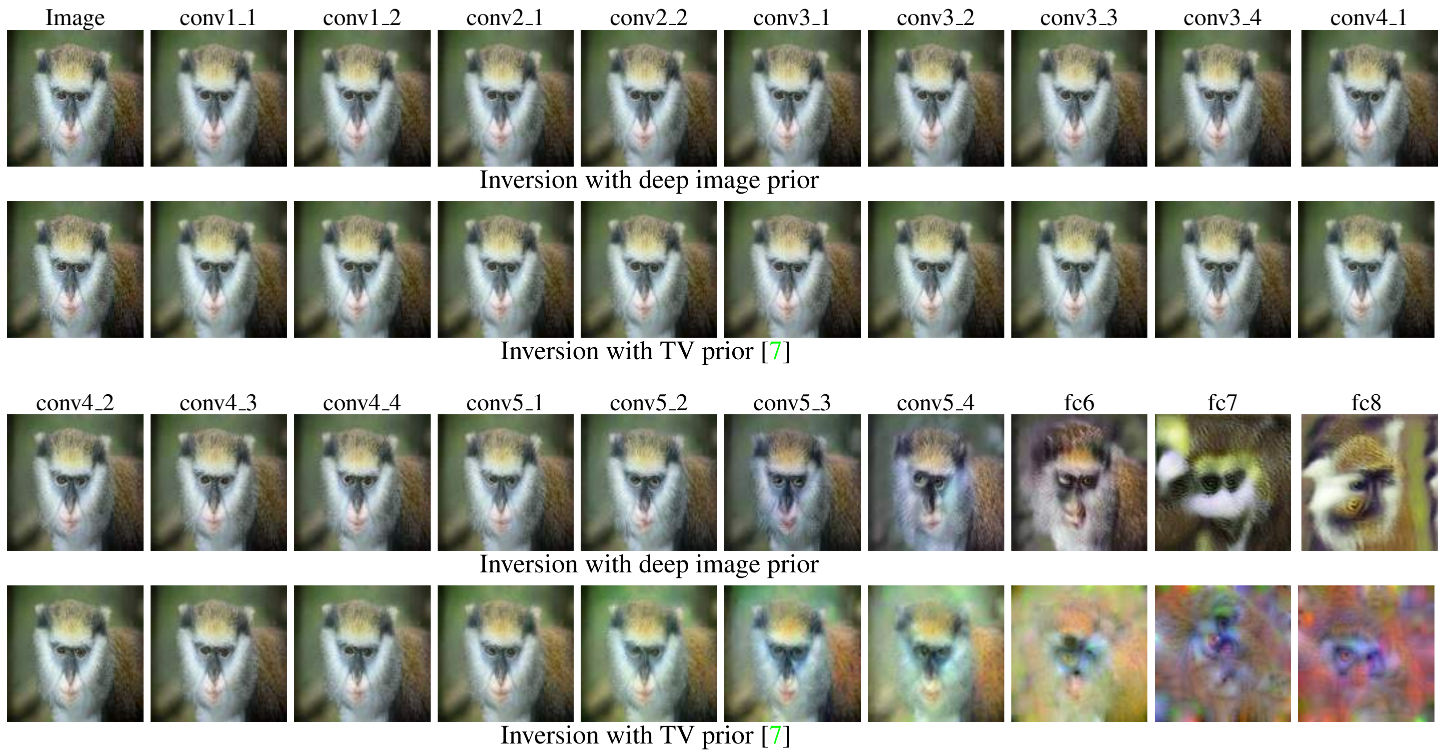


Figure 16: Inversion of VGG-19 [8] network activations at different layers with different priors (see main text for the discussion of the problem).