One Network to Solve Them All — Solving Linear Inverse Problems using Deep Projection Models

[Supplemental Materials]

1. Network Architecture

We now describe the architecture of the networks used in the paper. We use exponential linear unit (elu) [1] as activation function. We also use virtual batch normalization [6], where the reference batch size b_{ref} is equal to the batch size used for stochastic gradient descent. We weight the reference batch with $\frac{b_{\text{ref}}}{b_{\text{ref}}+1}$. We define some shorthands for the basic components used in the networks.

- *conv(w, c, s)*: convolution with $w \times w$ window size, c output channels and s stride.
- *dconv(w, c, s)* deconvolution (transpose of the convolution operation) with $w \times w$ window size, c output channels and s stride.
- *vbn*: virtual batch normalization.
- *bottleneck(same/half/quarter)*: bottleneck residual units [5] having the same, half, or one-fourth of the dimensionality of the input. Their block diagrams are shown in Figure 1.
- *cfc*: a channel-wise fully connected layer, whose output dimension is with same size as the input dimension.
- *fc(s)*: a fully-connected layer with the output size s.

To simply the notation, we use the subscript *ve* on a component to indicate that it is followed by *vbn* and *elu*.

Projection network $\mathcal{P}$. The projection network $\mathcal{P}$ is composed of one encoder network $\mathcal{E}$ and one decoder network, like a typical autoencoder. The encoder $\mathcal{E}$ projects an input to a 1024-dimensional latent space, and the decoder projects the latent representation back to the image space. The architecture of $\mathcal{E}$ is as follows.

$$\begin{array}{lcccc} Input & \rightarrow & conv(4,64,1)_{ve} & \rightarrow & conv(4,128,1)_{ve} \\ & \rightarrow & conv(4,256,2)_{ve} & \rightarrow & conv(4,512,2)_{ve} \\ & \rightarrow & conv(4,1024,2)_{ve} & \rightarrow & cfc \\ & \rightarrow & conv(2,1024,1)_{ve} & & (latent) \end{array} \tag{1}$$

The decoder is a symmetric counter part of the encoder:

$$\begin{array}{lcccc} latent & \rightarrow & dconv(4,512,2)_{ve} & \rightarrow & dconv(4,256,2)_{ve} \\ & \rightarrow & dconv(4,128,1)_{ve} & \rightarrow & dconv(4,64,1)_{ve} \\ & \rightarrow & dconv(4,3,1) & & (Output) \end{array} \tag{2}$$

Image-space classifier $\mathcal{D}$. As shown in Figure 3 of the paper, we use two classifiers — one operates in the image space $\mathbb{R}^d$ and discriminates natural images from the projection outputs, the other operates in the latent space of $\mathcal{P}$ based on the hypothesis that after encoded by $\mathcal{E}$, a perturbed image and a natural image should already lie in the same set.

For the image-space classifier $\mathcal{D}$, we use the 50-layer architecture of [4] but use the bottleneck blocks suggested in [5]. The detailed architecture is as follows.

$$\begin{array}{lcccc} Input & \rightarrow & conv(4,64,1) & & \\ & \rightarrow & bottleneck(half) & \rightarrow & \{bottleneck(same)\}_{\times 3} \\ & \rightarrow & bottleneck(half) & \rightarrow & \{bottleneck(same)\}_{\times 4} \\ & \rightarrow & bottleneck(half) & \rightarrow & \{bottleneck(same)\}_{\times 6} \\ & \rightarrow & bottleneck(half) & \rightarrow & \{bottleneck(same)\}_{\times 3} \\ & \rightarrow & vbn \ \& \ elu & \rightarrow & fc(1) \ (output), \end{array} \tag{3}$$

where $\{\}_{\times n}$ means we repeat the building block n times.

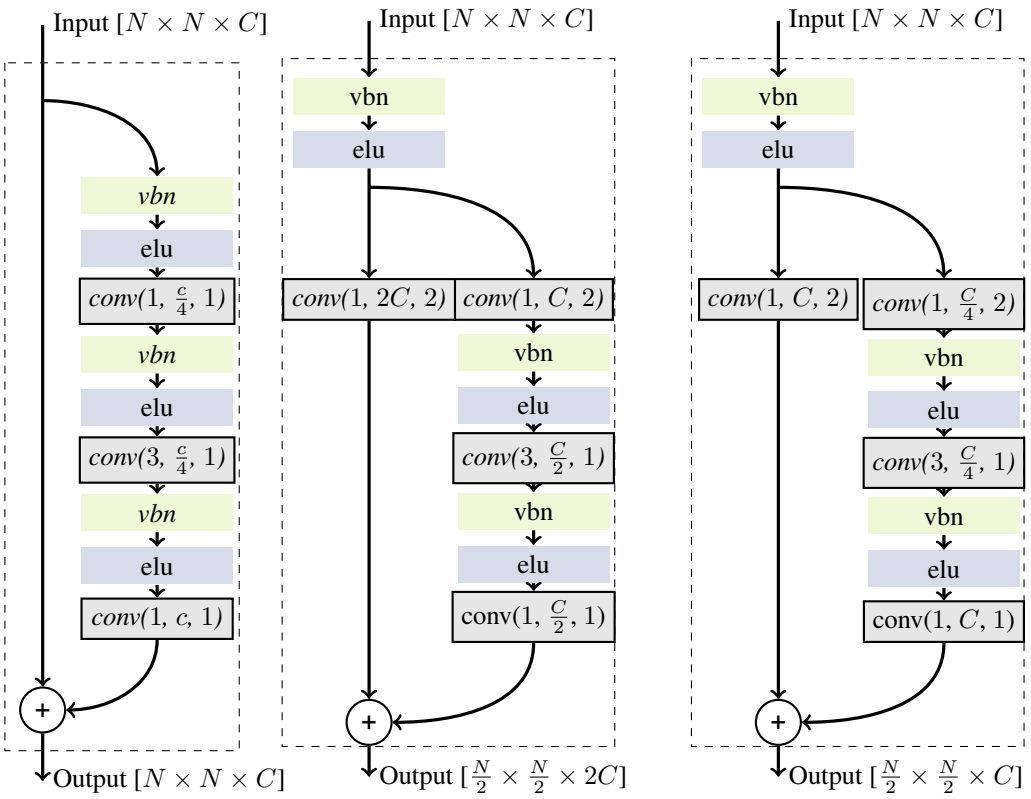
Latent-space classifier $\mathcal{D}_\ell$. The latent space classifier $\mathcal{D}_\ell$ operates on the output of the encoder $\mathcal{E}$. Since the input dimension is smaller than that of $\mathcal{D}$, we use fewer *bottleneck* blocks than we did in $\mathcal{D}$.

$$\begin{array}{lcc} Input & \rightarrow & bottleneck(same)_{\times 3} \\ & \rightarrow & bottleneck(quarter) \\ & \rightarrow & \{bottleneck(same)\}_{\times 2} \\ & \rightarrow & vbn \ \& \ elu \\ & \rightarrow & fc(1) \ (output) \end{array} \tag{4}$$

2. More Implementation Details

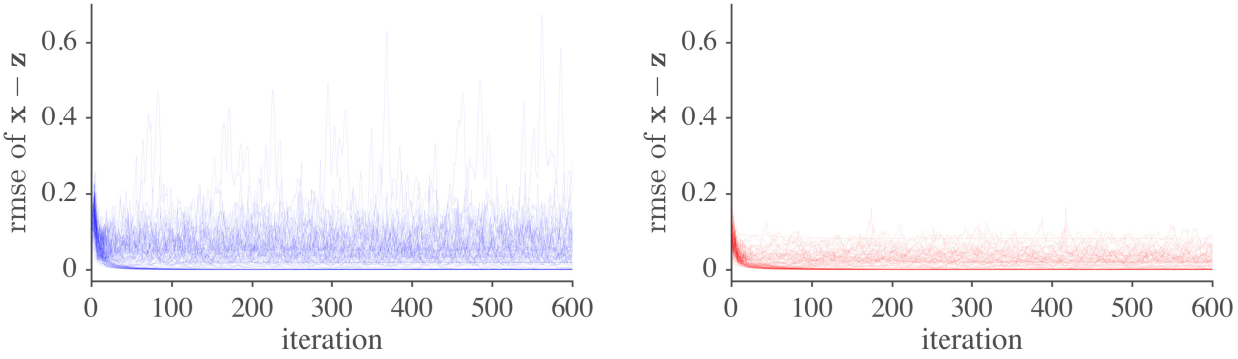
We now describe the details of the training procedure on each dataset.

(i) *MNIST dataset.* The images in the dataset are 28×28 and grayscale. We train the projector and the classifier



(a) bottleneck(same) (b) bottleneck(half) (c) bottleneck(quarter)

Figure 1: Block diagrams of the bottleneck components used in the paper. (a) *bottleneck(same)* preserves the dimensionality of the input by maintaining the same output spatial dimension and the numger of channels. (b) *bottleneck(half)* reduces dimensionality by 2 via halving each spatial dimension and doubling the number of channels. (c) *bottleneck(quarter)* reduces dimensionality by 4 via halving each spatial dimension.



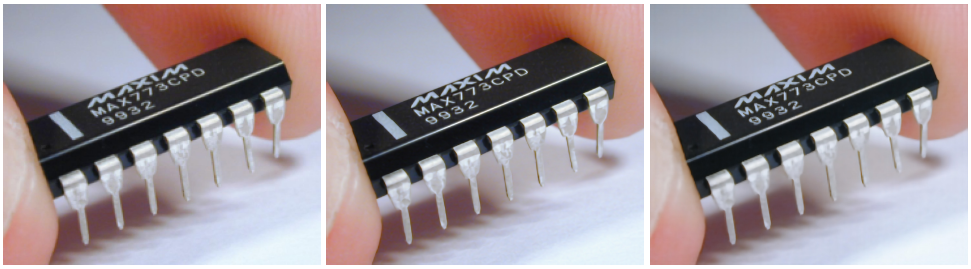
(a) Non-smooth $\mathcal{D}$ and $\mathcal{D}_\ell$ (b) Smooth $\mathcal{D}$ and $\mathcal{D}_\ell$

Figure 2: Comparison of ADMM convergence between (a) a projection network trained with indifferentiable $\mathcal{D}$ and $\mathcal{D}_\ell$ and (b) the proposed architecture, in which the gradient of $\mathcal{D}$ and $\mathcal{D}_\ell$ are Lipschitz continuous. We perform scattered inpainting with box size equal to 6 and a total of 10 boxes on 100 random images in ImageNet dataset. For both cases, we set $\rho = 0.05$. We use transparent lines ($\alpha = 0.1$) in order to show densities.

input

ground truth [3] (PSNR=20.50) proposed (PSNR=23.20)

input

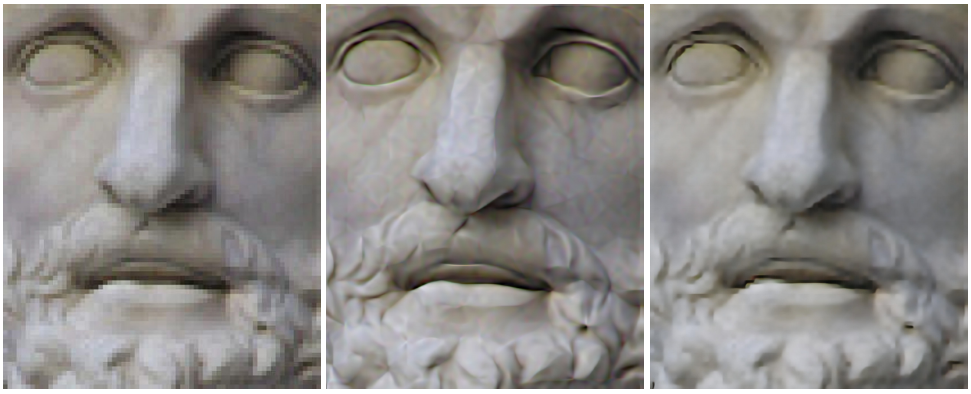


bicubic interpolation [3] proposed

input

bicubic interpolation [3] proposed

input



bicubic interpolation [3] proposed

Figure 3: Results of $4\times$ (on the first three rows) and $8\times$ (on the last row) super-resolution of Freeman and Fattal [3] (on the third column) and the proposed method (on the last column). All the input images are from [3]. Note that all the images, except for the one in the first row, do not have ground truth. For the proposed method, we use the projection network trained on ImageNet dataset and set $\rho = 1.0$.

networks on the training set and test the results on the test set. Since the dataset is relatively simpler, we remove the upper three layers from both $\mathcal{D}$ and $\mathcal{D}_\ell$, and we do not perturb the images by smoothing. We use batches with 32 instances and train the networks for $80,000$ iterations.

(ii) *MS-Celeb-1M dataset.* The dataset contains a total of 8 million aligned and cropped face images of 10 thousand people from different viewing angles. We randomly select images of $73,678$ people as the training set and those of $25,923$ people as the test set. We resize the images into 64×64. We use batches with 25 instances and train the network for $10,000$ iterations.

(iii) *ImageNet dataset.* ImageNet contains 1.2 million training images and 100 thousand test images on the Internet. We randomly crop a square image based on the shorter side of the images and resize the cropped image into 64×64. We use batches with 25 instances and train the network for $68,000$ iterations.

(iv) *LabelMe dataset.* The dataset also contains images from the Internet. We do not train a projection network on the dataset. We use the test set to quantitatively evaluative the performance of the projection network trained on ImageNet dataset. Since the images in the dataset have very high resolution, we first resize the images to 469×387, which is the average resolution of the images in ImageNet dataset. We then follow the same procedure as that used with ImageNet to generate test images.

Image perturbation. We generate perturbed images with two methods — adding Gaussian noise with spatially varying standard deviations and smoothing the input images. We generate the noise by multiplying a randomly sampled standard Gaussian noise with a weighted mask upsampled from a low-dimensional mask with bicubic algorithm. The weighted mask is randomly sampled from a uniform distribution ranging from $[0.05, 0.5]$. Note that the images are ranging from $[-1, 1]$. To smooth the input images, we first downsample the input images and then use nearest-neighbor method to upsample the results. The ratio to the downsample is uniformly sampled from $[0.2, 0.95]$. After smoothing the images, we add the noise described above. We only use the smoothed images on ImageNet and MS-Cele-1M datasets.

3. Convergence of ADMM

Theorem 1 states a sufficient condition for the nonconvex ADMM to converge. Based on Theorem 1, we use exponential linear units as the activation functions in $\mathcal{D}$ and $\mathcal{D}_\ell$ and truncate their weights after each training iteration, in order for the gradient of $\mathcal{D}$ and $\mathcal{D}_\ell$ to be Lipschitz continuous. Even though Theorem 1 is just a sufficient condition, in practice, we observe improvement in terms of convergence. We conduct experiments on scattered inpainting on ImageNet dataset using two projection networks — one trained with $\mathcal{D}$ and $\mathcal{D}_\ell$ using the smooth exponential linear units, and the other trained with $\mathcal{D}$ and $\mathcal{D}_\ell$ using the non-smooth leaky rectified linear units. Note that leaky rectified linear units are indifferentiable and thus violate the sufficient condition provided by Theorem 1. Figure 2 shows the root mean square error of $\mathbf{x} - \mathbf{z}$, which is a good indicator of the convergence of ADMM, of the two networks. As can be seen, using leaky rectified linear units results in higher and spikier root mean square error of $\mathbf{x} - \mathbf{z}$ than using exponential linear units. This indicates a less stable ADMM process. It shows that following Theorem 1 can help the convergence of ADMM.

4. Super-resolution results

We compare the proposed method with that of Freeman and Fattal [3] on $4\times$- and $8\times$-super resolution. The results are shown in Figure 3.

5. Denoising results

We compare the proposed method with the state-of-the-art denoising algorithm, BM3D [2]. We add Gaussian random noise with different standard deviation σ (out of 255) to the test images, which were taken by the author with a cell phone camera. The value of σ of each image is provided to BM3D. For the proposed method, we let $A = I$, the identity matrix, and set $\rho = \frac{3}{255}\sigma$. To perform the projection operation on the 384×512 images, we use the same projection network learned from ImageNet dataset and apply it to 64×64 patches. As shown in Figure 4 and Figure 5, when σ is larger than 40, the proposed method consistently outperform BM3D.

6. More examples

More results on MS-Celeb-1M and ImageNet dataset are shown in Figure 6 and Figure 7, respectively.

References

[1] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015. 1

[2] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. Bm3d image denoising with shape-adaptive principal component analysis. In *Signal Processing with Adaptive Sparse Structured Representations*, 2009. 4

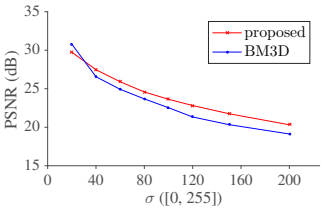
[3] G. Freedman and R. Fattal. Image and video upscaling from local self-examples. *ACM TOG*, 28(3):1–10, 2010. 3, 4

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1

[5] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *ECCV*, 2016. 1

[6] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *NIPS*, 2016. 1

ground truth

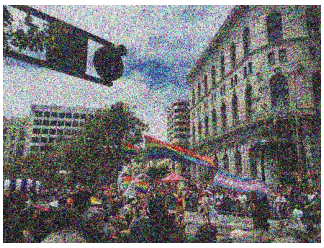


PSNR *vs.* σ

input ($\sigma = 40$)

BM3D (PSNR=26.56)

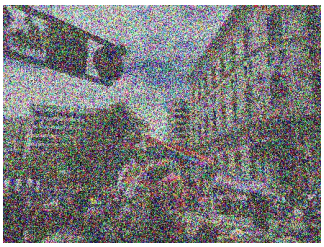
proposed (PSNR=27.47)



input ($\sigma = 100$)

BM3D (PSNR=22.54)

proposed (PSNR=23.64)

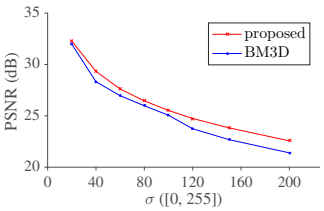


input ($\sigma = 200$)

BM3D (PSNR=19.13)

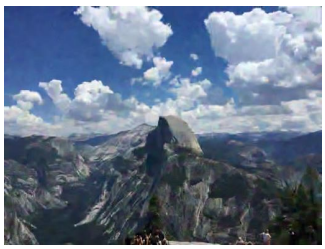
proposed (PSNR=20.32)

ground truth

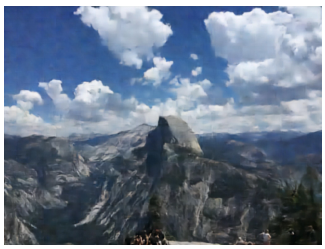


PSNR *vs.* σ

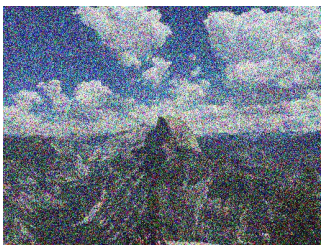
input ($\sigma = 40$)



BM3D (PSNR=28.32)



proposed (PSNR=29.33)



input ($\sigma = 100$)

BM3D (PSNR=25.06)

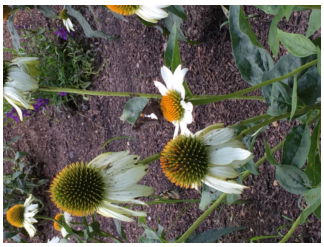
proposed (PSNR=25.53)

input ($\sigma = 200$)

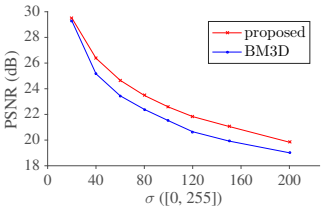
BM3D (PSNR=21.40)

proposed (PSNR=22.57)

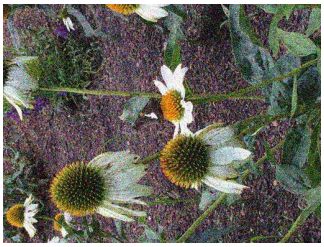
Figure 4: Comparison to BM3D on image denoising



ground truth

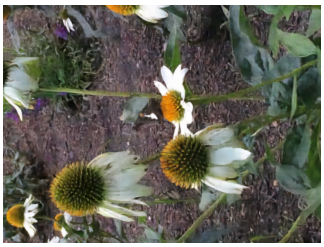


PSNR *vs.* σ



input ($\sigma = 40$)

BM3D (PSNR=25.16)

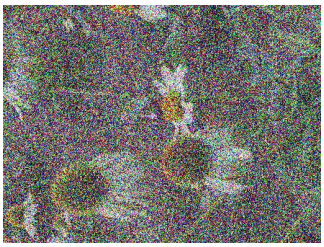


proposed (PSNR=26.38)

input ($\sigma = 100$)

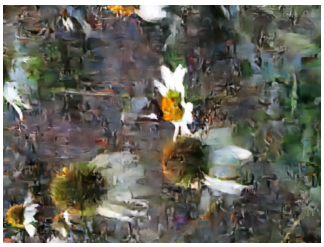
BM3D (PSNR=21.51)

proposed (PSNR=22.58)

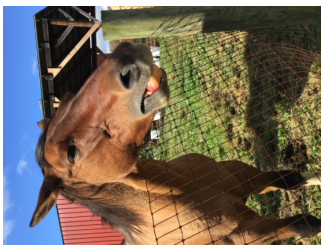


input ($\sigma = 200$)

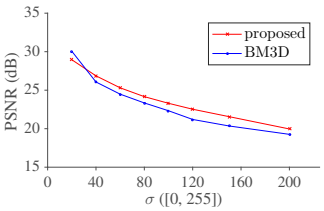
BM3D (PSNR=19.01)



proposed (PSNR=19.84)

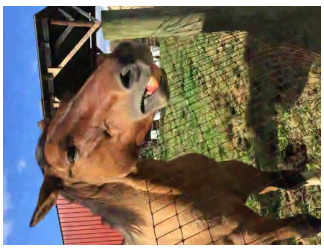


ground truth

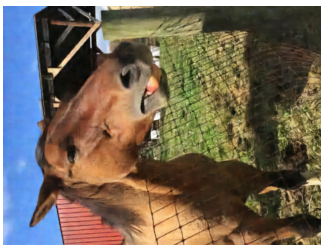


PSNR *vs.* σ

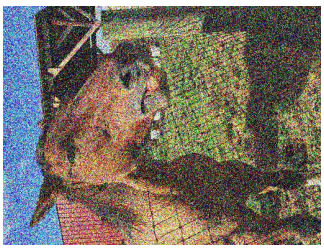
input ($\sigma = 40$)



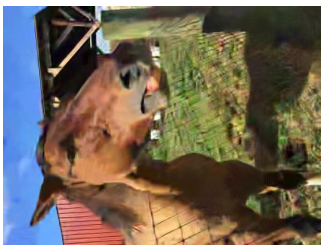
BM3D (PSNR=26.06)



proposed (PSNR=26.86)



input ($\sigma = 100$)



BM3D (PSNR=22.33)

proposed (PSNR=23.30)

input ($\sigma = 200$)

BM3D (PSNR=19.27)

proposed (PSNR=19.97)

Figure 5: Comparison to BM3D on image denoising

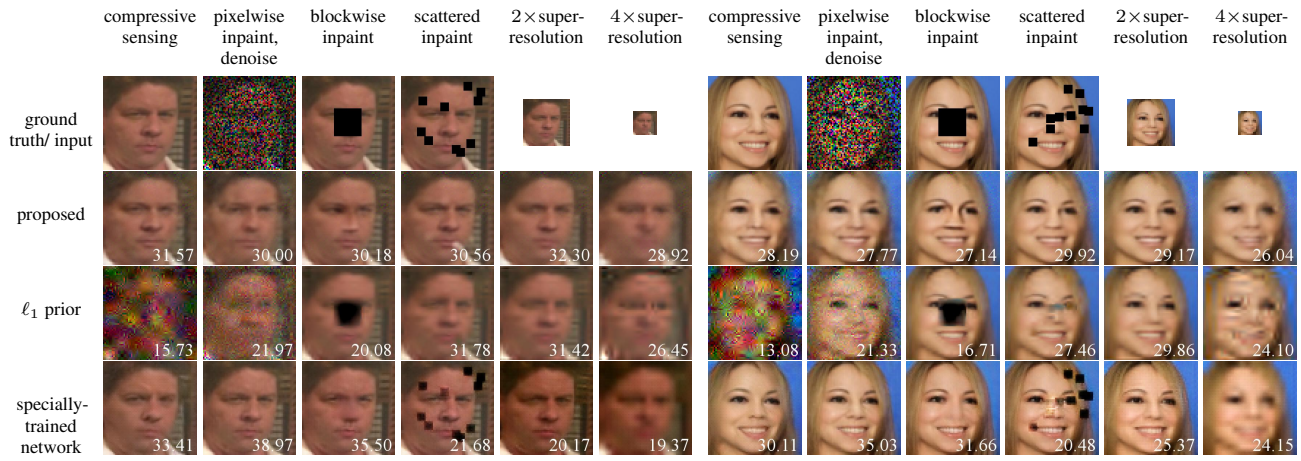


Figure 6: More results on MS-Celeb-1M dataset. The PSNR values are shown in the lower-right corner of each image. For compressive sensing, we test on $\frac{m}{d} = 0.1$. For pixelwise inpainting, we drop 50% of the pixels and add Gaussian noise with $\sigma = 0.1$. We use $\rho = 1.0$ on both super resolution tasks.

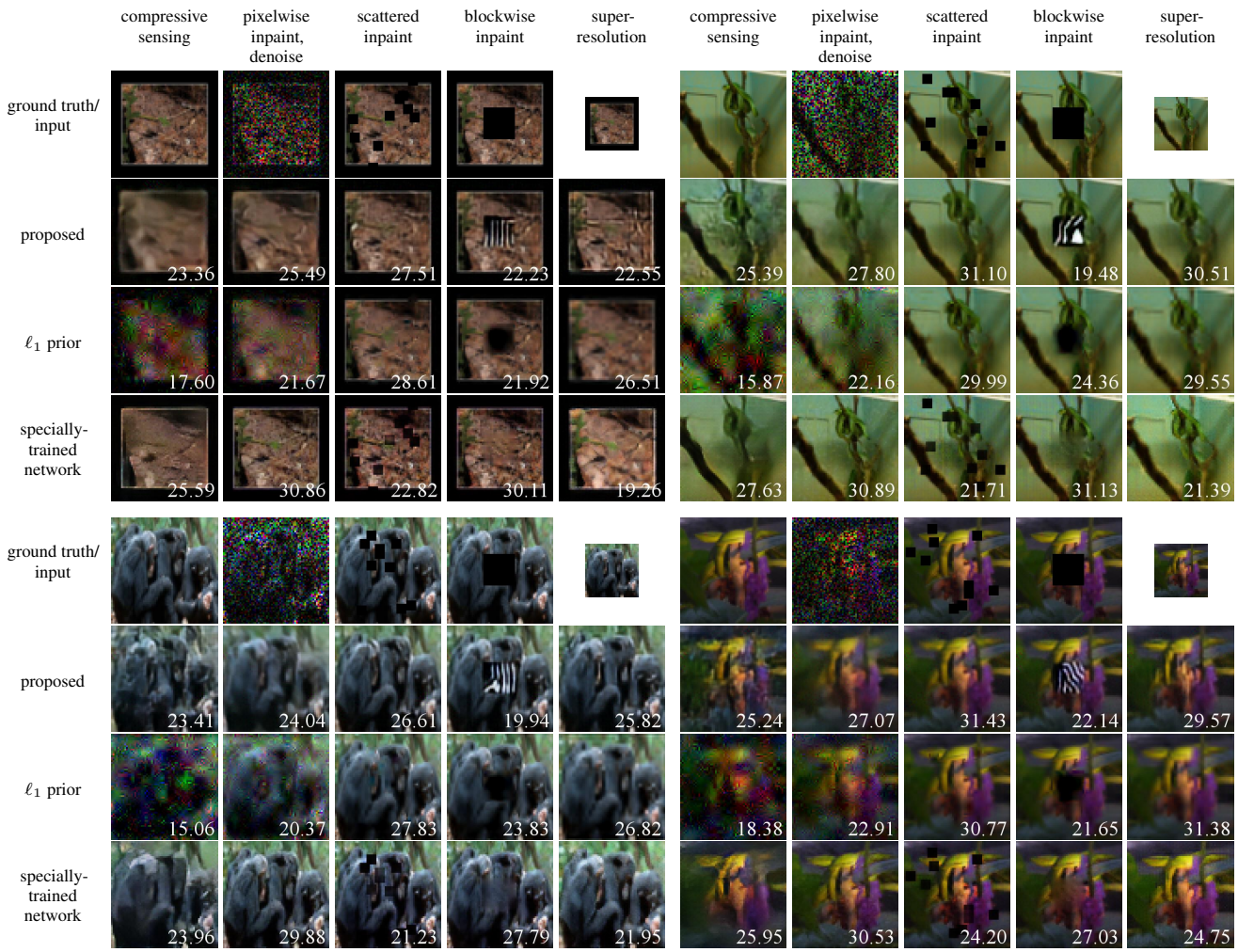


Figure 7: More results on ImageNet dataset. Compressive sensing uses $\frac{m}{d} = 0.1$. For pixelwise inpainting, we drop 50% of the pixels and add Gaussian noise with $\sigma = 0.1$. We use $\rho = 0.05$ on scattered inpainting and $\rho = 0.5$ on super resolution.