5 Supplemental Materials

5.1 Implementation Details

The proposed deep networks are implemented in TensorFlow [3]. The filters of networks are randomly initialized from a zero mean Gaussian distribution with standard deviation 0.005. We use ADAM [15] as our autoencoder($\mathcal{G}$) and discriminator($\mathcal{D}$) optimizer. All deep-based models are trained on the same training images. In the training stage, we use the size of 32 mini-batch, and set the momentum to 0.9. The learning rates are set to $5e-4$ for autoencoder($\mathcal{G}$) and $5e-5$ for discriminator($\mathcal{D}$). We also decrease the learning rate by multiplying 0.99 after each epoch. In addition, the pixel values of the removal regions in the training and testing images are set to zero. The hyper-parameters in our models are empirically set as $\lambda_{\mathcal{G}} = 1$, $\lambda_{\mathcal{D}} = 0.03$. More, we also apply gradient clipping in the range of ± 10 to prevent exploding gradients in the networks.

5.2 Methods in Comparison

Previous methods can be roughly categorized into the vision-based[8, 12, 17, 29] and deep-based[14, 22, 26] approaches:

Vision-based: CSH[17] is a patch-based method that can find the best coherent patches and fill them into the missing regions. TNNR[12] and Field of Experts[29] are optimization methods that can obtain the best solutions based on image structures to recover corrupted regions. Finally, inpaints_nans[8] is a state-of-the-art inpainting tool which interpolates and extrapolates missing elements in a 2D array. This method has been rated five stars on the MathWorks file exchange. We use all the codes that provided by the authors on their project pages.

Deep-based: Deep learning methods are widely used in image restoration problems (eg. super-resolution, inpainting). RED-Net[22] is a deep autoencoder using l_2-norm optimization that tackles many image restoration tasks. In our comparison, we use a simple version (10 layers) which can fully utilize the receptive field of training images. Context Encoder (CE)[26] is the first conditional GANs for image inpainting. We use the torch code from the author's github, and the size input image is restricted to 128×128. Lately, image translation has been a popular topic that was raised by Isola et al.[14]. Their proposed method (pix2pix) can perform impressive transformations of different domain images, and they claimed their method can also be used for image inpainting. We also adapt the code from their project pages, and the size of input image is restricted to 256×256. For fair comparison, the deep model based methods are trained and validated on the same corrupted datasets in all the experiments.

5.3 Single vs Integrated Model

We want to prove that training with multiple types of corruptions can surely have better model performances and generalization than only training with a single type of corruptions. Hence, under the same network settings and hyper-parameters, we train four models with the training data containing only one type of corruptions (Single model: Text, Line, Scribble, Random), and compare them to our proposed model (Integrated model: include all types of corruptions). In Figure 5, it shows the comparison of PSNR of different models on MSCOCO [21] testing sets. There are five bars on a bin of corruption types. Each bar indicates a different model. The model that is trained on a single type of corruption images fails to well complete other types of corrupted images. This justified that merely using a model trained on a single corruption is hard to generalized on other type of corruptions, and by integrating all type of corruptions into training, the model can be not only more general, and even obtain a small enhancement than those single models.

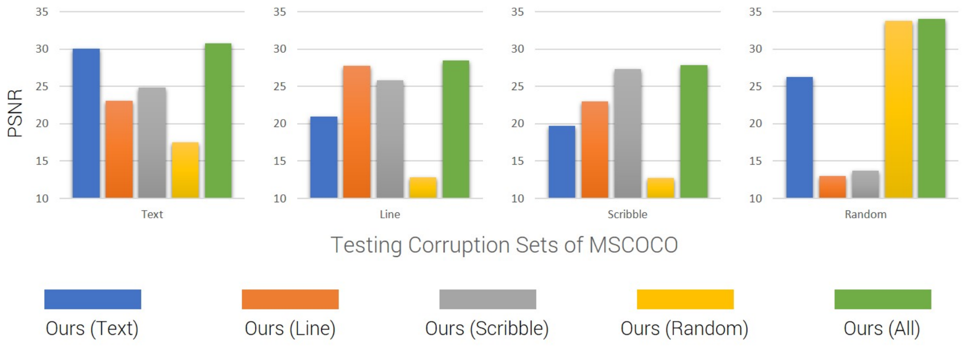

Figure 5: Quantitative results for models trained with different types of corruptions.

5.4 Corruption Masks

Here we show the training corruption examples of corruptions that generated by $\mathcal{M}$. As in Figure 6, it contains different types of missing regions with a certain level of variant, which helps our model in training, and pushes our model to capture the essential features in corrupted images in order to reconstruct back to well-completed images. In training stage, all the pixel values of corruption part are set to zero.

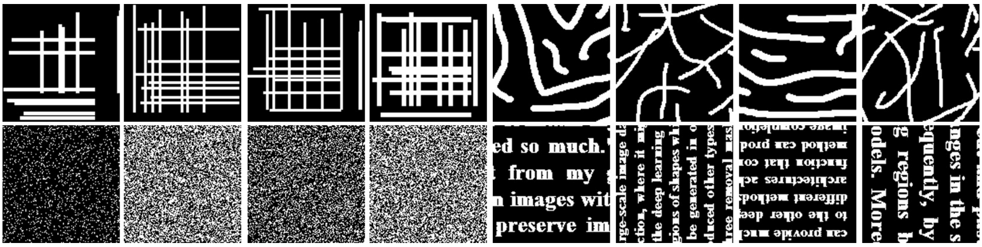

Figure 6: Examples of training corrupted masks generated from $\mathcal{M}$.

5.5 Performances beyond Model

There are some results on high resolution images and user-specified corruptions. In Figure 7, **first column** is the corrupted images created by users, and the corrupted parts are visualized in purple color. **second column** depicts the completed images by using our model. The resolution of each images are 600×900 and 630×1200, respectively.

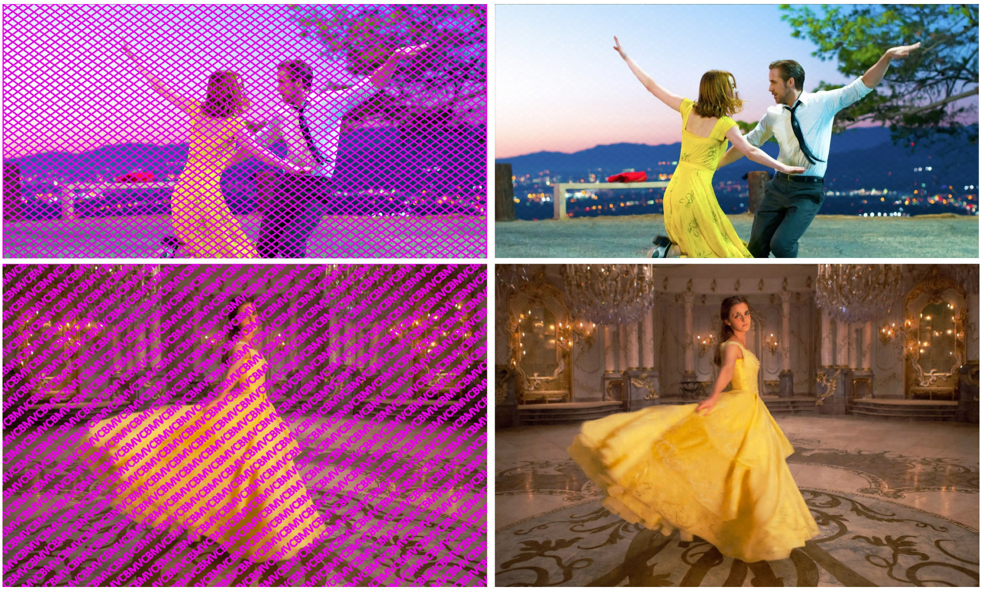

Figure 7: Qualitative results on high-resolution public images and user-defined corruption masks.

5.6 More Results on CUB, Flowers and MSCOCO

We also compare different methods on another dataset (**Particular**): 102-Category Flower[25]. It contains 8,189 flower images. We randomly pick 100 images for validation and 100 for testing, and the rest are the training images.

Datasets, types	CSH[17]	TNNR[12]	FoE[29]	nans[8]	RED-Net[22]	CE[26]	pix2pix[14]	Ours
	Algorithms							
Flowers[25], Text	22.54/0.807	25.28/0.834	31.33/0.955	**32.16/0.959**	25.15/0.807	28.58/0.872	25.95/0.839	32.05/0.948
Flowers[25], Line	21.34/0.794	15.38/0.631	26.64/0.897	29.35/**0.925**	23.36/0.767	27.75/0.860	24.27/0.802	**29.79**/0.921
Flowers[25], Scribble	20.11/0.766	20.59/0.771	25.99/0.894	27.70/**0.913**	23.29/0.794	26.97/0.862	23.95/0.801	**28.20**/0.909
Flowers[25], Random	22.34/0.652	32.36/0.908	36.73/0.979	**37.77/0.980**	21.08/0.572	28.69/0.832	24.44/0.780	35.35/0.961
Average	21.58/0.755	23.40/0.786	30.17/0.931	**31.74/0.944**	23.22/0.735	28.00/0.856	24.66/0.805	31.35/0.935

Table 5: Quantitative results on Flowers dataset[25] and different types of corruptions. The higher the (PSNR/SSIM) are, the closer the completed images are compared to the ground truth images. Bold and under-line indicate the best and the second best performance, respectively.

The following figures shows the additional results of four types of corruptions on different datasets. Figure 9 to 12 are the results on CUB[31]. Figure 13 to 16 are the results on Flowers[25]. Figure 17 to 20 are the results on MSCOCO[21]. Last, Figure 21 to Figure 24 are the results on BSDS500 test datasets.

5.7 Additional Results on Arbitrary Corruptions

In addition to the corruptions we introduced in the paper, we also apply our model toward image completion on arbitrary shape missing regions. We first generate arbitrary masks that are formed of different polygons , and then produce the training data and ground truth pairs. In the experiments, we tackle the task: face completion by using CelebA datasets. Figure 8 shows the quantitative results which evaluated by PSNR and SSIM, and the visual outcomes. For fair comparison, vision-based methods are not included in this evaluation since they lacks of semantics to filled the missing content by merely considering similar information around. Although we achieve the highest PSNR and SSIM, the quality of the recovered faces can be further justified (e.g. user study) in the future.

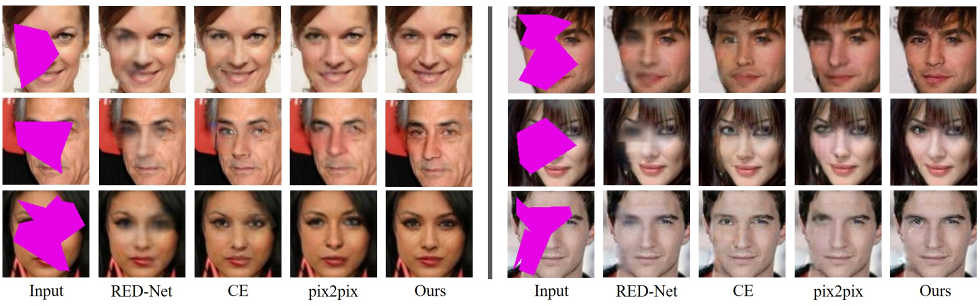

Figure 8: The testing results of face completion on CelebA datasets. The evaluation (PSNR/SSIM) metrics are shown within the bracket behind methods. **RED-Net**[22] (26.67/0.849), **CE**[26] (27.06/0.845), **pix2pix**[14] (26.83/0.853), and **Ours** (**27.90/0.869**).

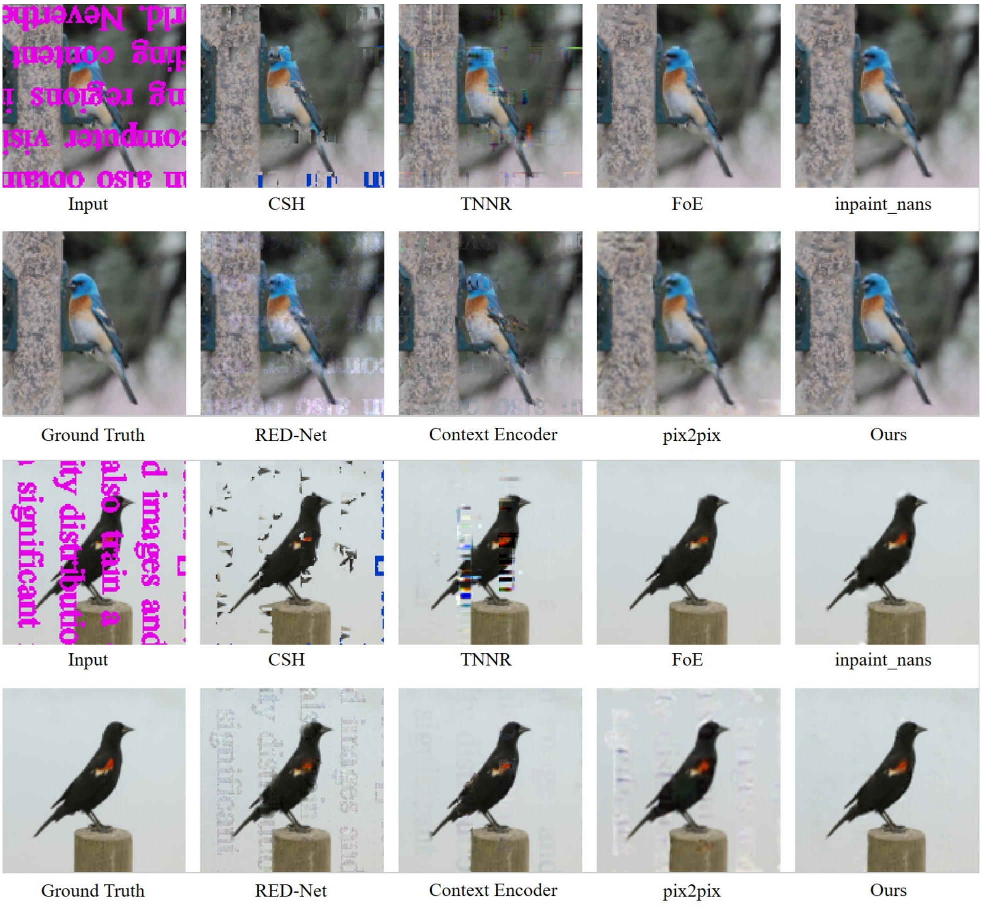

Figure 9: Results of **text** corruptions on CUB[31] datasets

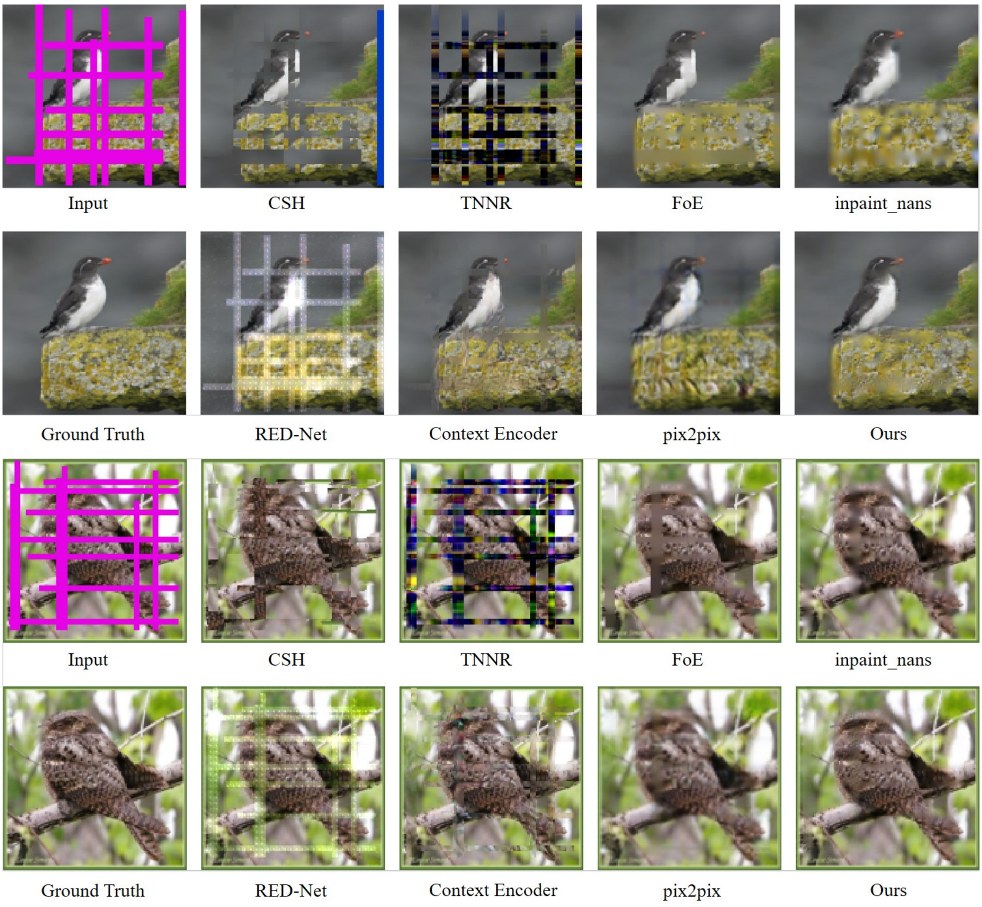

Figure 10: Results of **line** corruptions on CUB[31] datasets

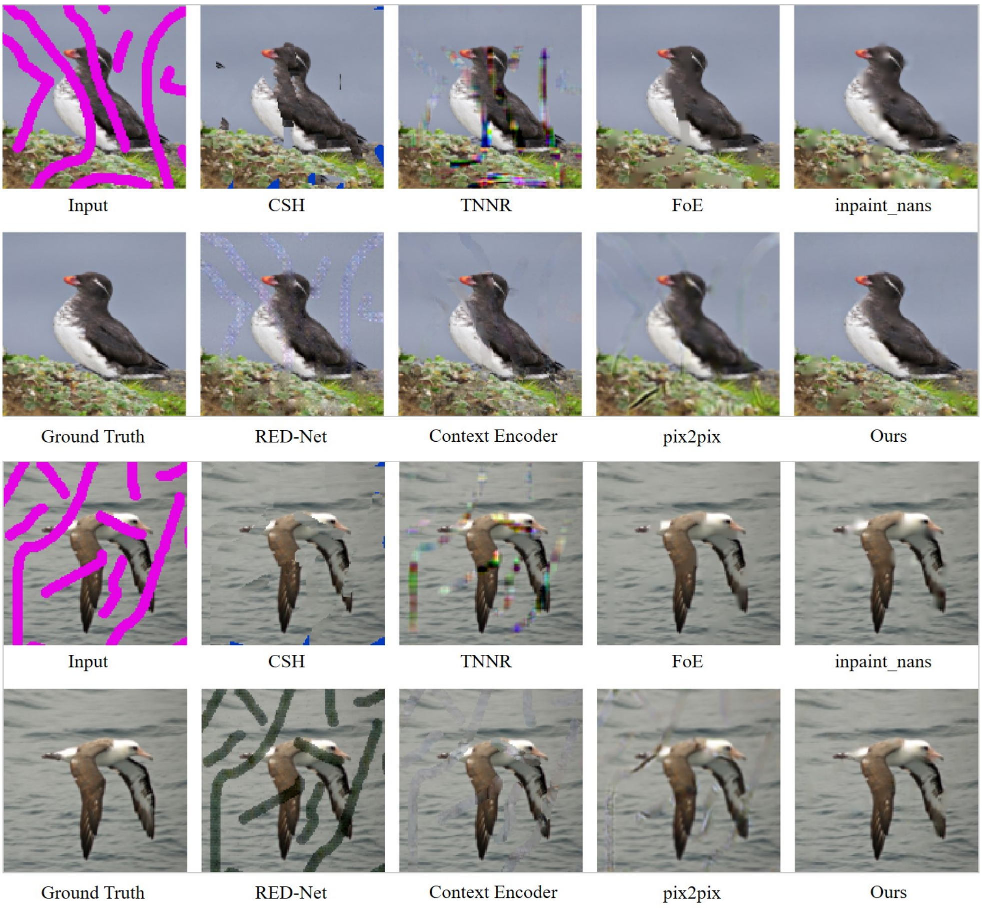

Figure 11: Results of **scribble** corruptions on CUB[31] datasets

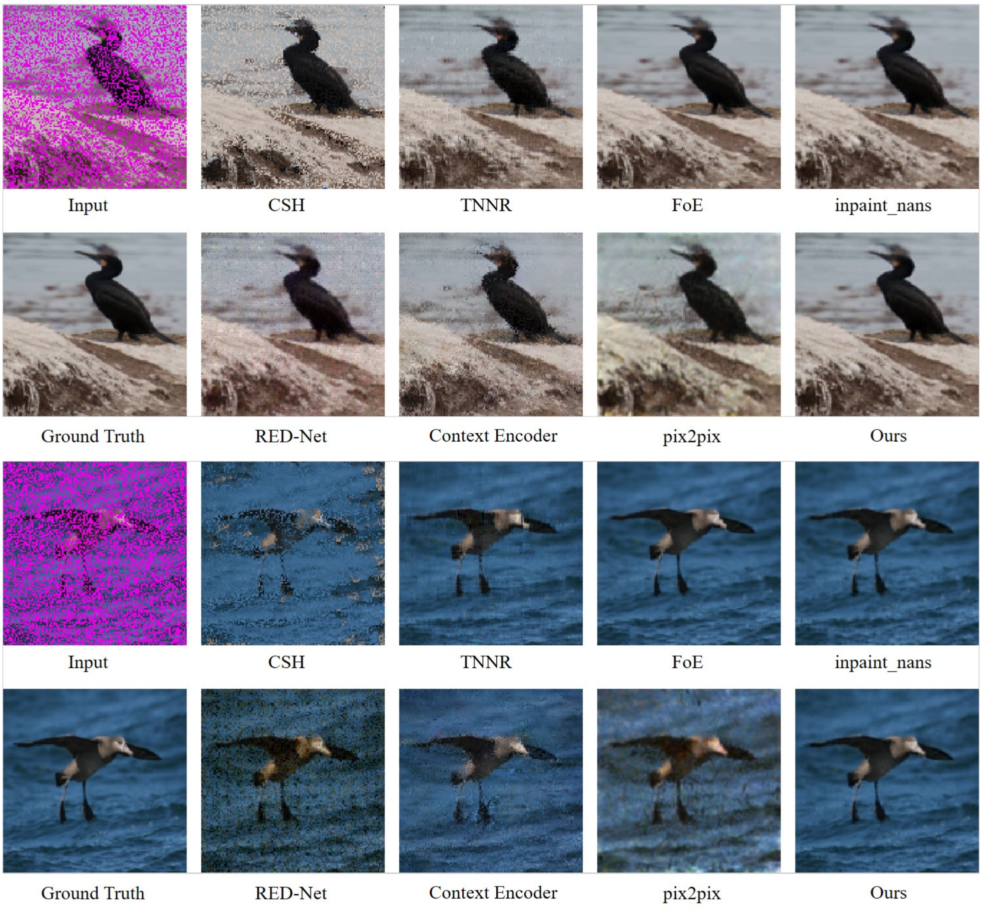

Figure 12: Results of **random** corruptions on CUB[31] datasets

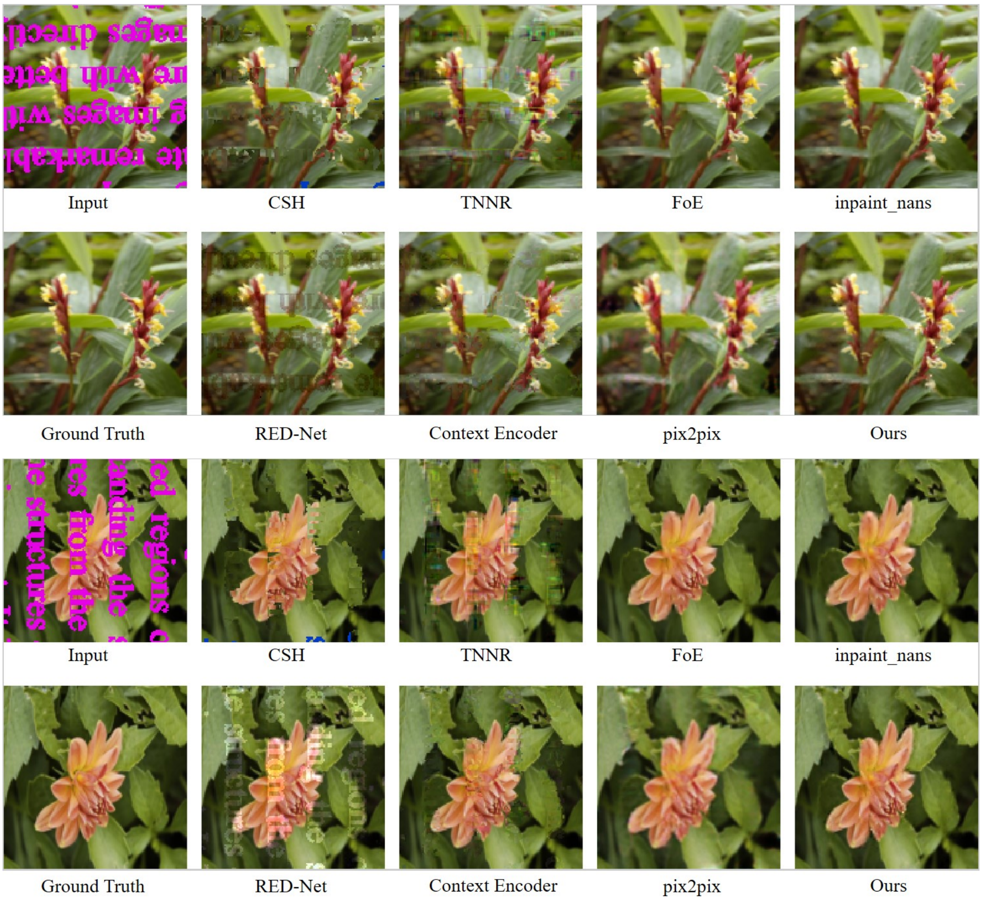

Figure 13: Results of **text** corruptions on Flowers[25] datasets

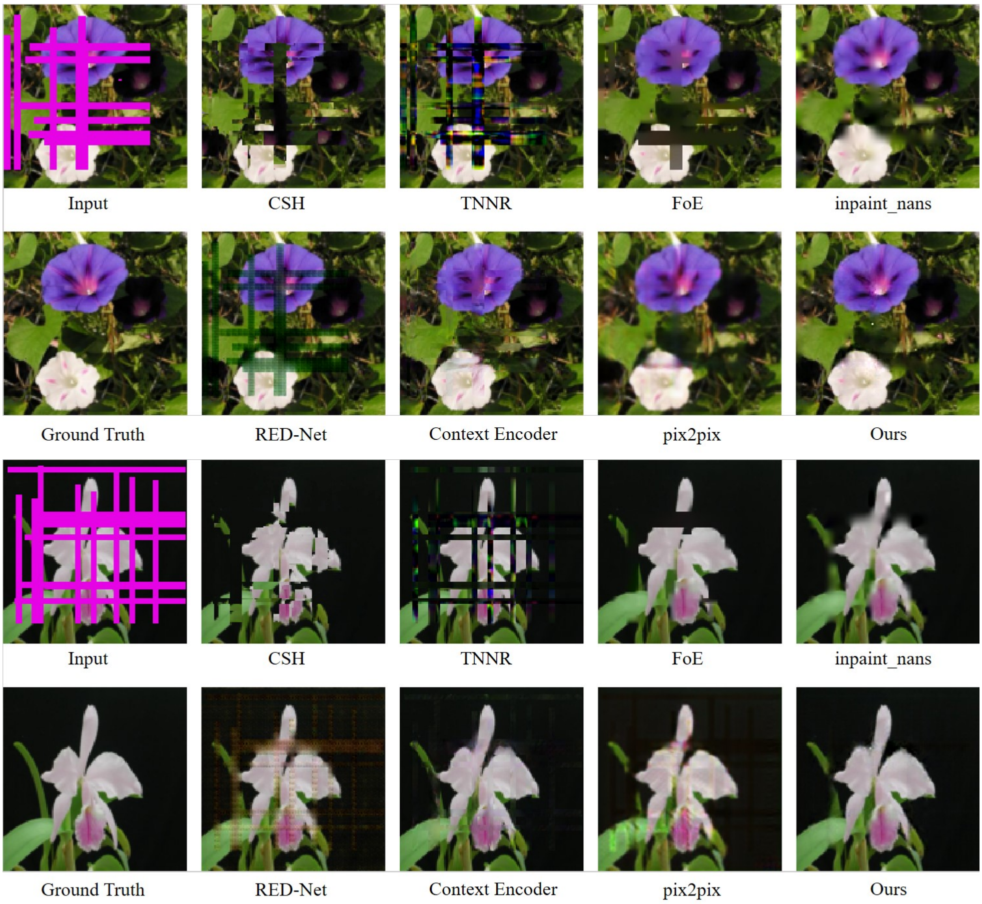

Figure 14: Results of **line** corruptions on Flowers[25] datasets

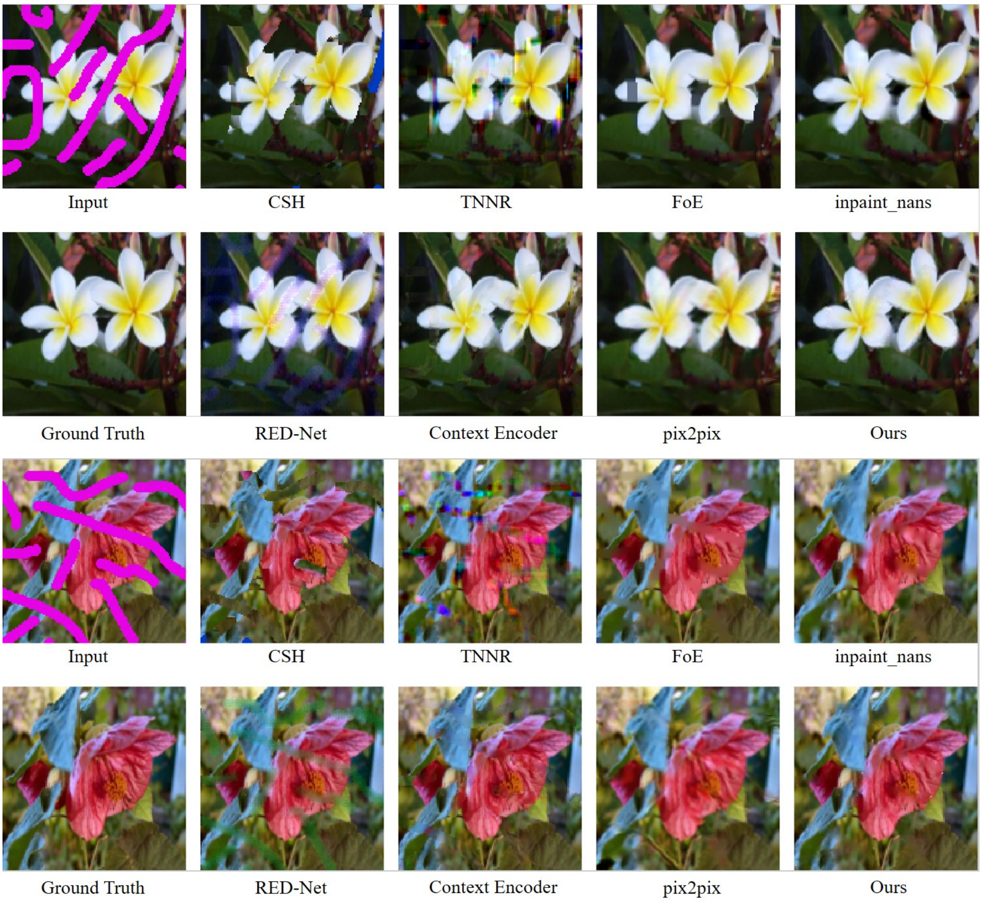

Figure 15: Results of **scribble** corruptions on Flowers[25] datasets

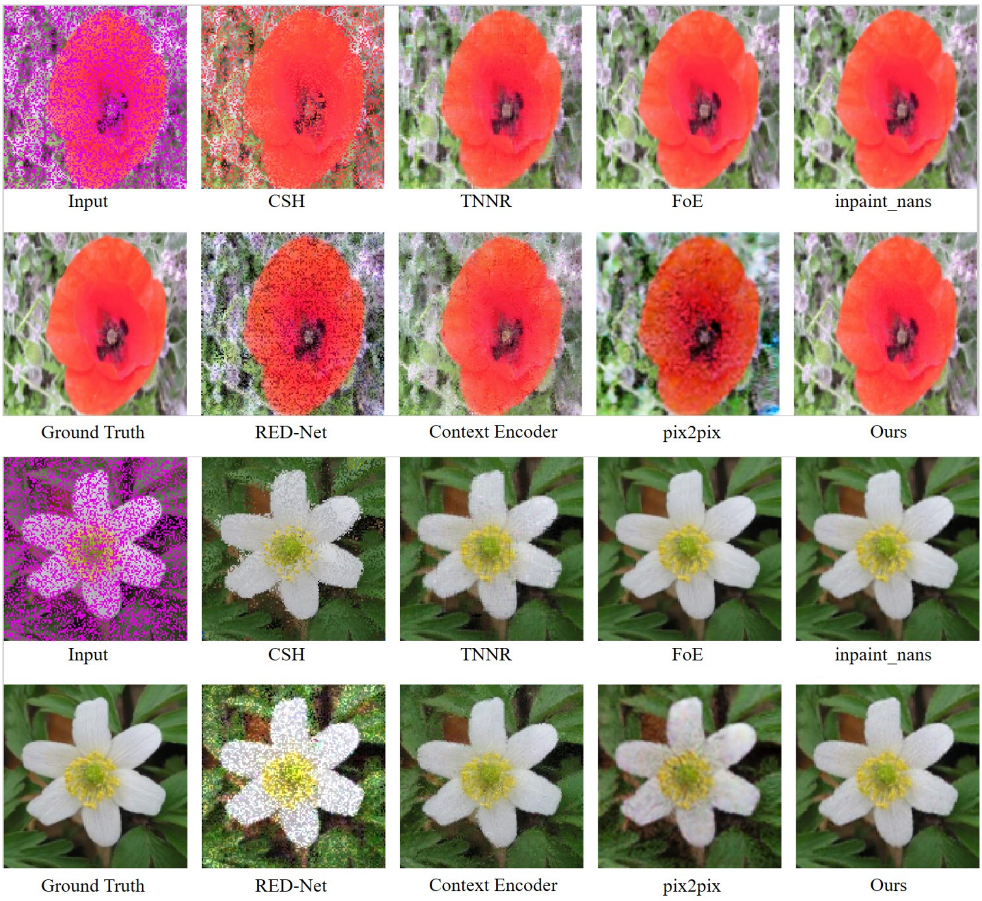

Figure 16: Results of **random** corruptions on Flowers[25] datasets

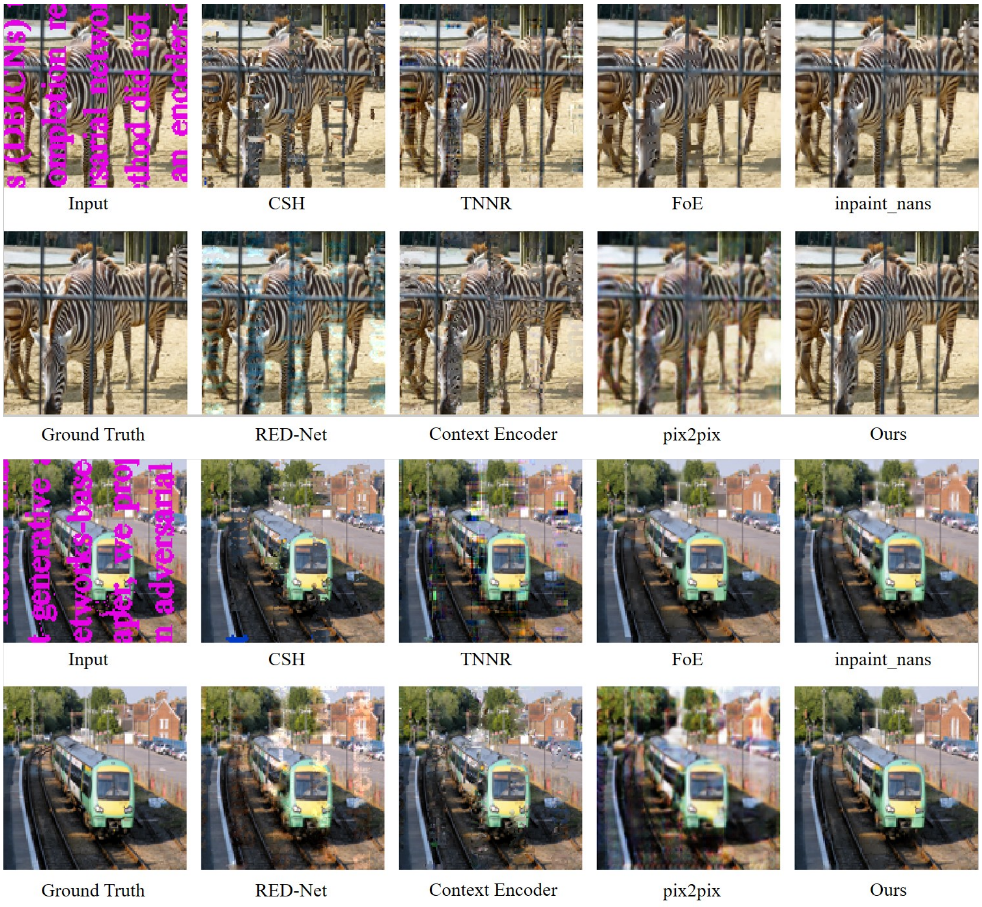

Figure 17: Results of **text** corruptions on MSCOCO[21] datasets

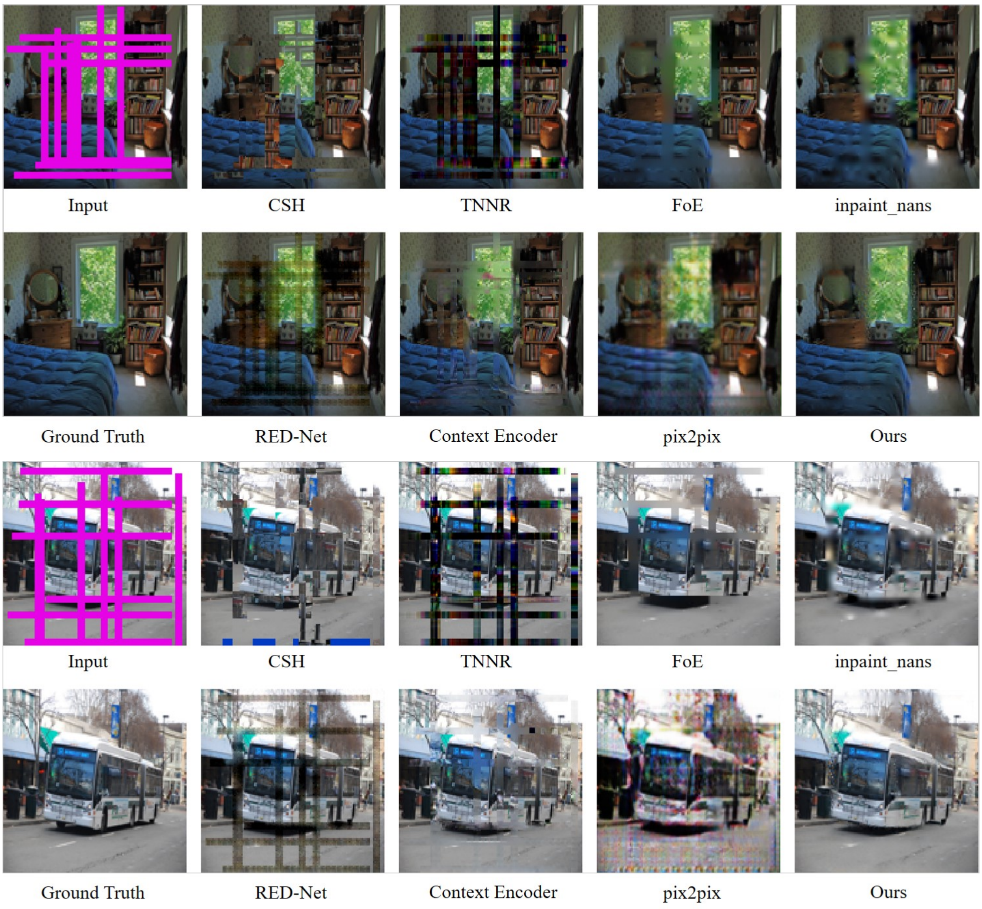

Figure 18: Results of **line** corruptions on MSCOCO[21] datasets

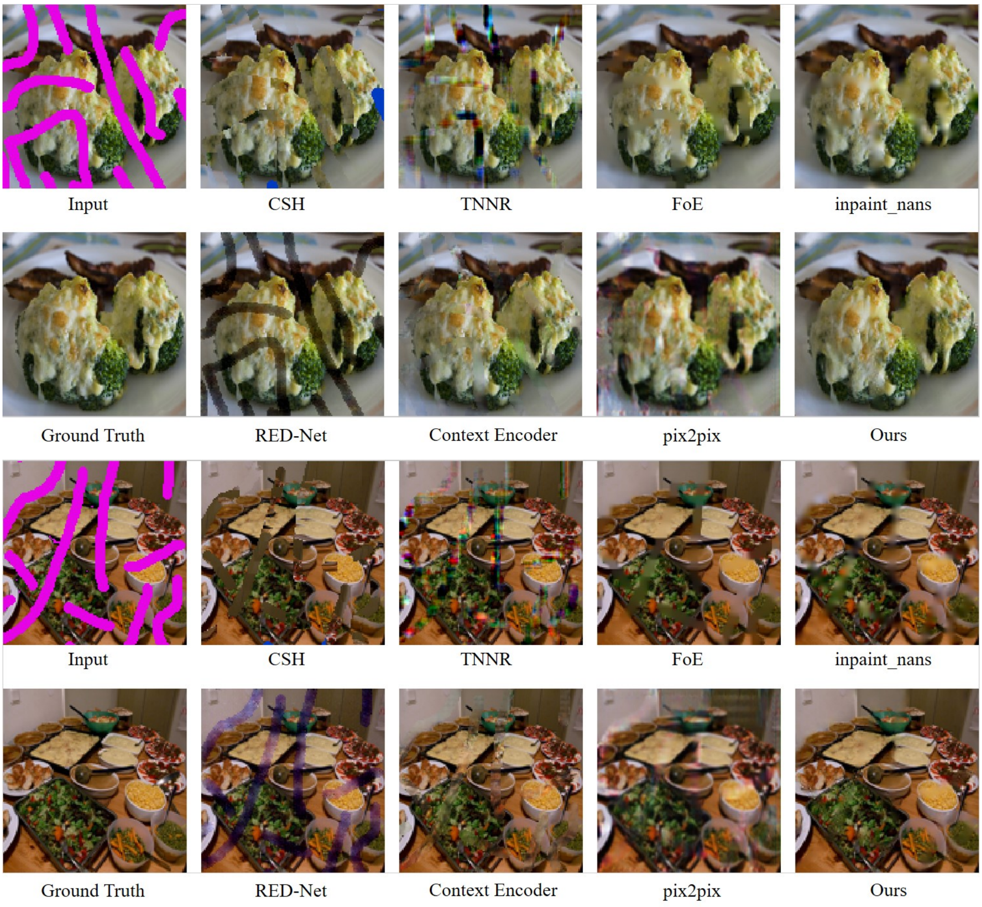

Figure 19: Results of **scribble** corruptions on MSCOCO[21] datasets

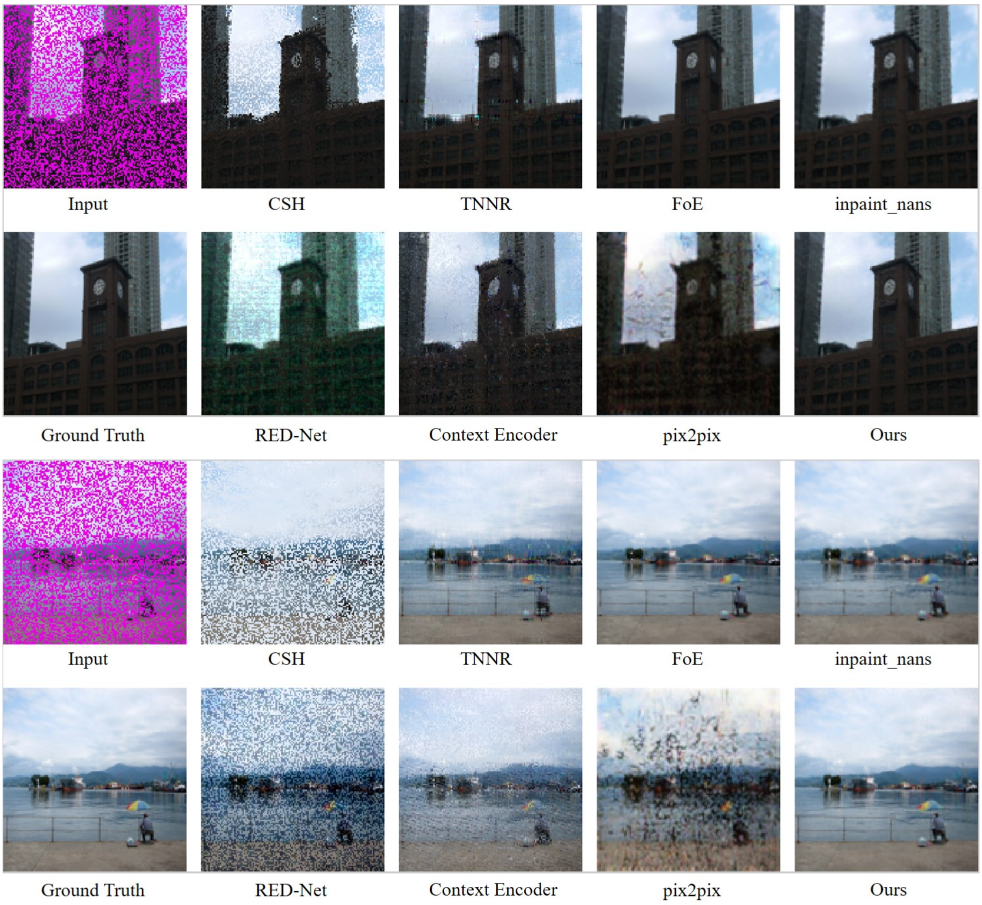

Figure 20: Results of **random** corruptions on MSCOCO[21] datasets

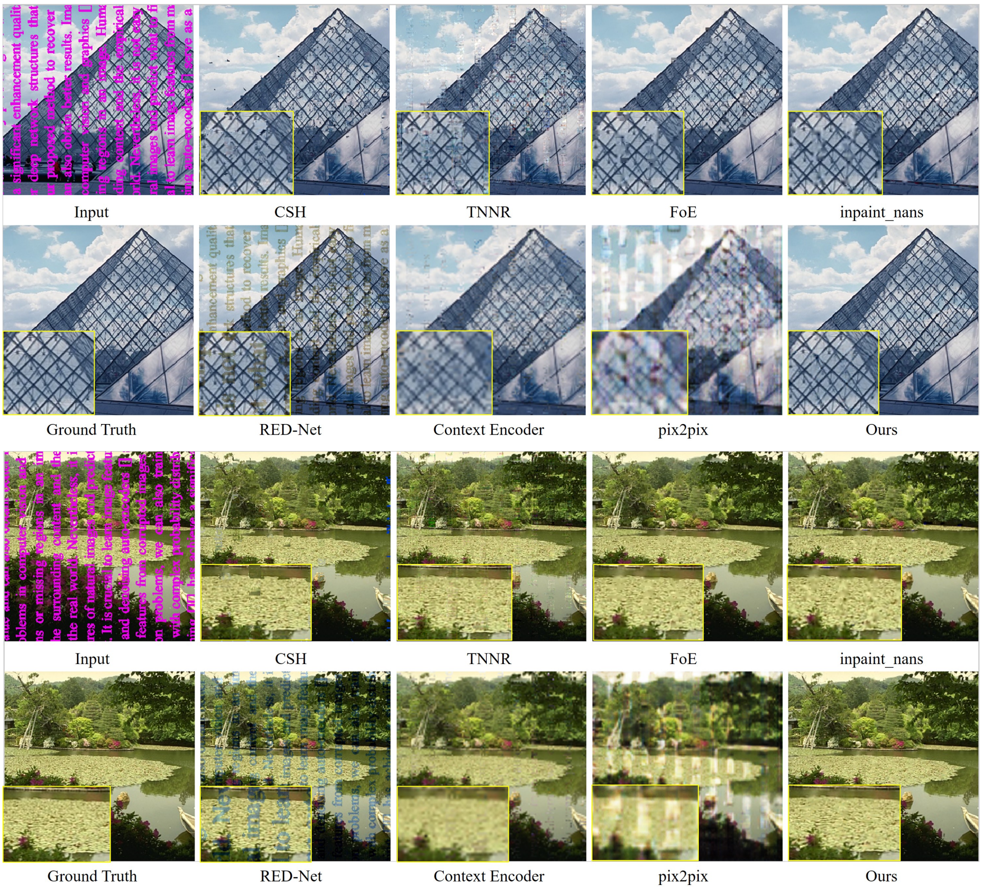

Figure 21: Results of **text** corruptions on BSDS500[24] datasets

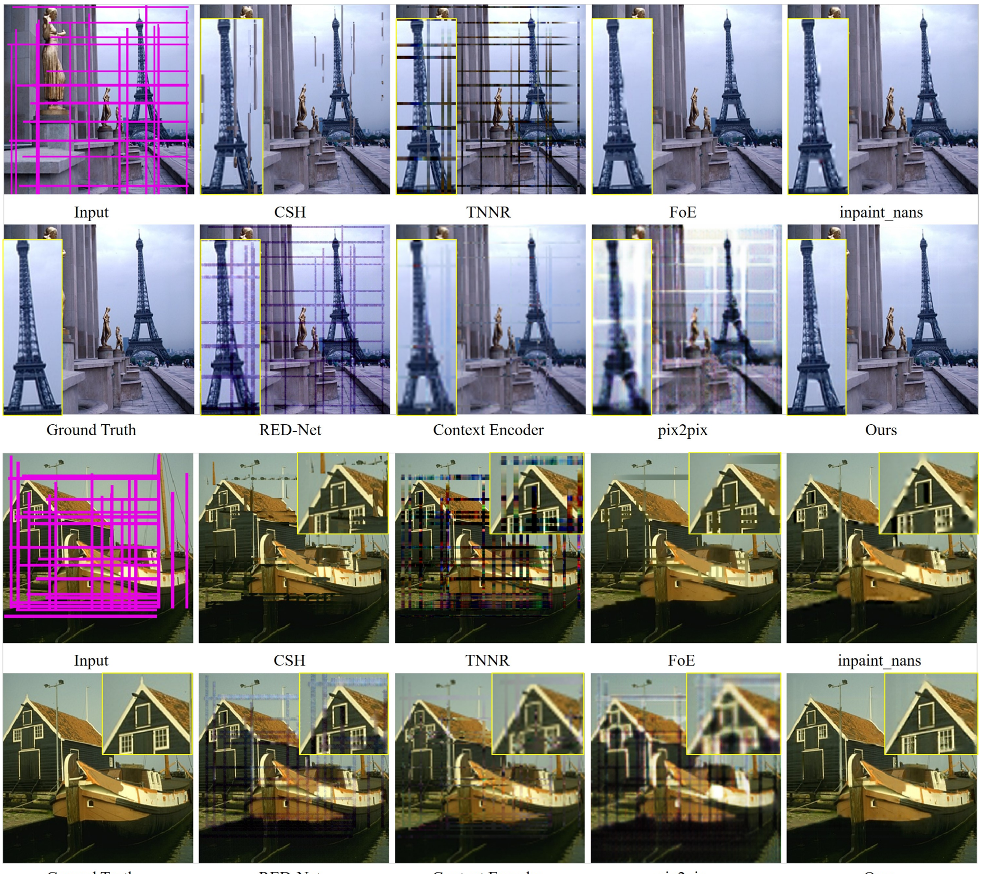

Figure 22: Results of **line** corruptions on BSDS500[24] datasets

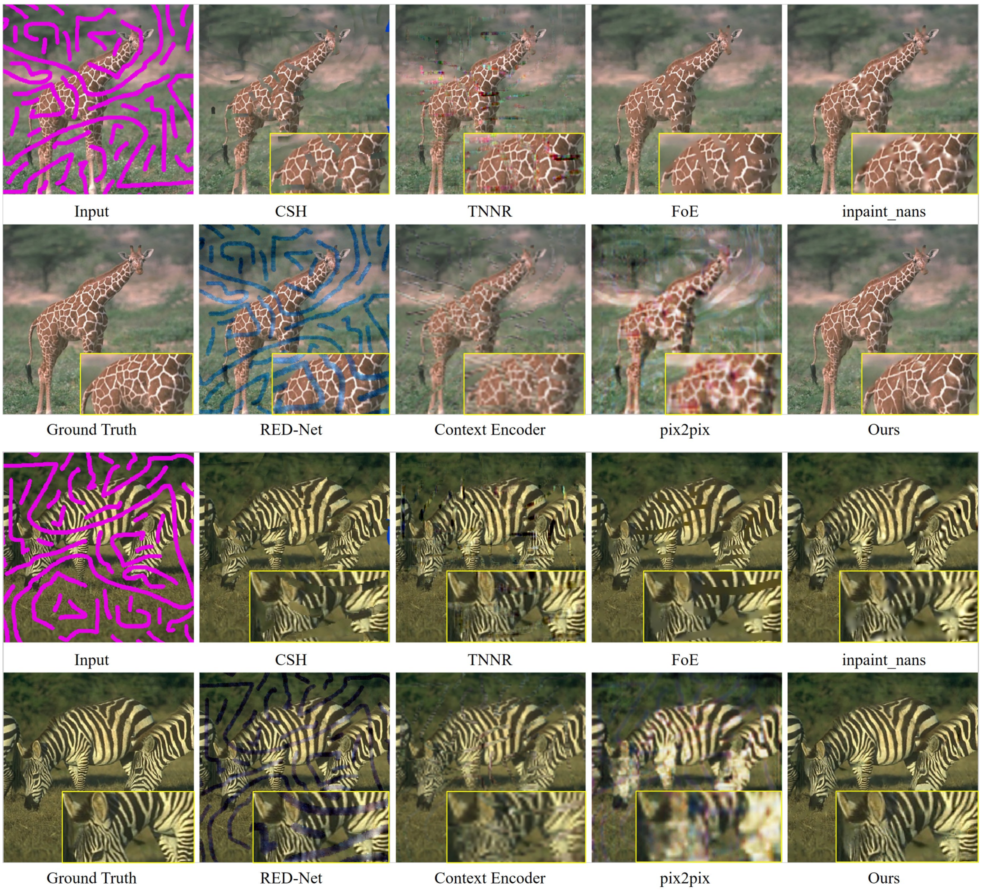

Figure 23: Results of **scribble** corruptions on BSDS500[24] datasets

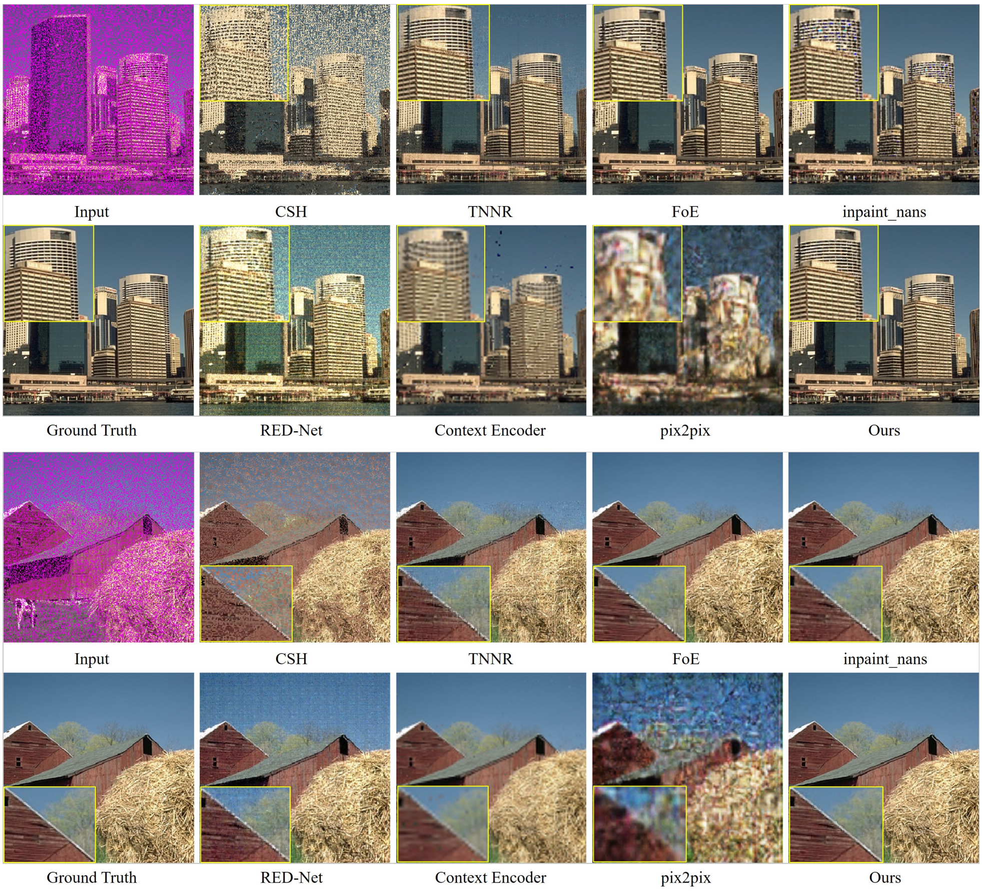

Figure 24: Results of **random** corruptions on BSDS500[24] datasets