Appendix for: Inverting Adversarially Robust Networks for Image Synthesis

The appendix is organized as follows:

- In Sec. A1, we present **a third application**, GAN-based *One-vs-All* anomaly detection using AR features, and show its benefits over standard techniques.
- In Sec. A2, we provide additional experimental results on feature inversion.
- In Sec. A3, we provide additional experimental results on downstream tasks.
- In Sec. A4, we provide implementation and experimental setup details.

A1 Anomaly Detection using AR Representations

A1.1 Approach

One-vs-All anomaly detection is the task of identifying samples that do not fit an expected pattern [13,12,66,67]. Given an unlabeled image dataset with normal (*positives*) and anomalous instances (*negatives*), the goal is to distinguish between them. Following GAN-based techniques [12], we train our proposed AR AlexNet autoencoder exclusively on positives to learn a how to accurately reconstruct them. Once trained on such a target distribution, we use its reconstruction accuracy to detect negatives.

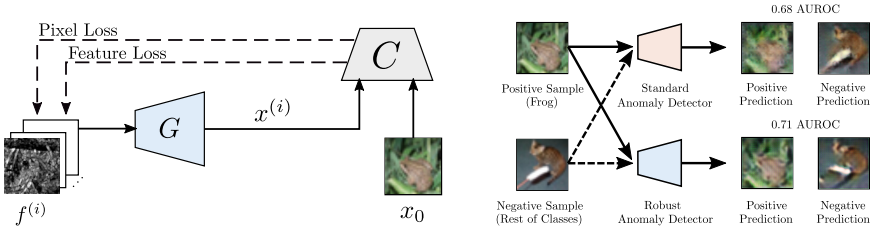
Given an unlabeled sample x and its AR features f, we search for $\hat{f}$ that yields the best reconstruction $\hat{x} = G_{\tilde{\phi}}(\hat{f})$ based on the following criterion (Fig. A1):

$$\hat{f} = \arg\min_{f} \alpha_{\text{pix}} \|G_{\tilde{\phi}}(f) - x\|_1 + \alpha_{\text{feat}} \|F_{\tilde{\theta}} \circ G_{\tilde{\phi}}(f) - F_{\tilde{\theta}}(x)\|_2^2, \tag{8}$$

where $\alpha_{\text{pix}}, \alpha_{\text{feat}} \in \mathbb{R}_{++}$ are hyperparameters. Essentially, x is associated to $\hat{f}$ that minimizes pixel and feature losses between estimated and target representations. Since $G_{\tilde{\phi}}$ has been trained on the distribution of positive samples, latent codes of negative samples generate abnormal reconstructions, revealing anomalous instances.

A1.2 Experiments

We hypothesize that our AR generator widens the reconstruction gap between in and out-of-distribution samples, improving its performance on anomaly detection. Given a labeled dataset, our generator is trained to invert AR features from samples of a single class (*positives*). Then, we evaluate how accurately samples from the rest of classes (*negatives*) are distinguished from positives on an unlabeled test set.



(a) Reconstruction-based Anomaly Score (b) *One-vs-All* Anomaly Detection

Fig. A1: Anomaly Detection using adversarially robust features.

Experimental Setup. We compare our technique using AR and standard features against ADGAN [12,13].We evaluate the performance on CIFAR10 and Cats vs. Dogs [68] datasets, where AUROC is computed on their full test sets.

Standard and AR encoders are fully-trained on ImageNet using the parameters described in Sec. A2. By freezing the encoder, generators are trained using pixel and feature losses on positives from the dataset of interest, CIFAR10 or Cats vs. Dogs. Input images are rescaled to 224×224 px. before being passed to the model, no additional data augmentation is applied during the generator training. The regularization parameters for both standard and AR autoencoders are heuristically selected as:

- Standard autoencoder: $\lambda_{\text{pix}} = 2 \times 10^{-3},\ \lambda_{\text{feat}} = 1 \times 10^{-2}$.
- AR autoencoder: $\lambda_{\text{pix}} = 2 \times 10^{-6},\ \lambda_{\text{feat}} = 1 \times 10^{-2}$.

Iterative Optimization Details. After training the generator on a particular class of interest, the optimal latent code $\hat{f}$ associated to an arbitrary target image x is obtained via stochastic gradient descent. For both standard and AR autoencoders, the optimization criteria are identical to that used during the generator training. Specifically, we minimize pixel and feature loss components using the following hyperparameters:

- Standard autoencoder: $\alpha_{\text{pix}} = 2 \times 10^{-3},\ \alpha_{\text{feat}} = 1 \times 10^{-2}$.
- AR autoencoder: $\alpha_{\text{pix}} = 2 \times 10^{-6},\ \alpha_{\text{feat}} = 1 \times 10^{-2}$.

Detection is performed by solving Eq. (8), where $f \in \mathbb{R}^{6\times6\times256}$ is initialized as white Gaussian noise and optimized for $i_{\text{max}} = 100$ iterations. The initial learn rate is chosen as 0.1 and linearly decreases along iterations down to 0.001.

Results. Full *one-vs-all* anomaly detection results for CIFAR-10 and Cats vs. Dogs datasets are shown in Tab. A1. On average, our AR model improves on outlier detection over its standard version and ADGAN. Our AR model gets 6.51% and 8.84% relative AUROC improvement over ADGAN on CIFAR-10 and Cats vs. Dogs, respectively. This shows our generator better distinguishes positives and negatives due to its improved reconstruction accuracy.

Dataset	Positive Class	ADGAN [12]	Proposed (Standard)	Proposed (AR)
CIFAR-10	0	0.649	0.6874	0.6533
	1	0.39	0.3498	0.3755
	2	0.652	0.6756	0.662
	3	0.481	0.5708	0.6123
	4	0.735	0.751	0.7538
	5	0.476	0.5101	0.5278
	6	0.623	0.6895	0.7113
	7	0.487	0.4773	0.4526
	8	0.66	0.7232	0.7008
	9	0.378	0.362	0.4408
	Average	0.553	0.5797	**0.589**
Cats vs. Dogs	0	0.507	0.663	0.649
	1	0.481	0.392	0.427
	Average	0.494	0.527	**0.538**

Table A1: AUROC of our proposed *one-versus-all* anomaly detection method for each class. Detection evaluated on CIFAR-10 and Cats vs. Dogs datasets. Best results highlighted in black.

A2 Additional Experiments on Feature Inversion

A2.1 Ablation Study

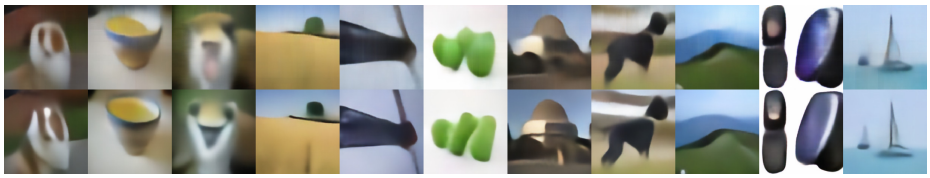
Feature inversion results obtained using different optimization criteria are illustrated in Fig. A2. Results clearly show the effect of each term, ℓ_1 pixel, feature and GAN components, in the final reconstruction. Samples correspond to the ImageNet validation set. Particularly, when inverting features using pixel and feature losses, adversarially robust features show a significant improvement with respect to their standard counterparts. This agrees with the idea of adversarially robust features being perceptually aligned.

A2.2 Robustness to Scale Changes

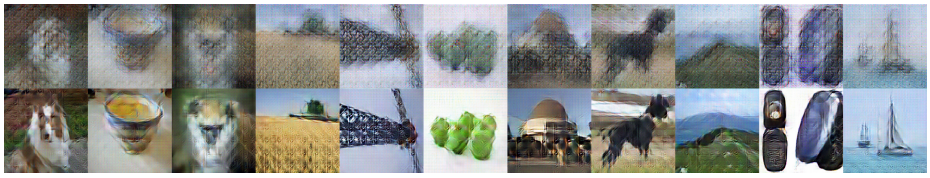
Inversion accuracy on upscaled low-resolution images is illustrated in Fig. A3 for scale factors $L \in \{1, \dots, 10\}$. While standard inversions show significant distortions for large upscaling factors L, reconstructions from adversarially robust representations show almost perfect reconstruction for high upscaling factors. Quantitative results are included in Tab. A2. Results improve almost monotonically when inverting AR representations, even without exposing the Autoencoder to high-resolution images during training and without any fine-tuning.

On the other hand, extended results on feature inversion from high-resolution images are illustrated in Fig. A4. Notice that, in contrast to the previous case, input samples correspond to natural high-resolution images and are encoded without any scaling. Results show a good color and edge preservation from our AR autoencoder, while inverting standard features show bogus components and noticeable color distortions.

(a) Ground-truth images.



(b) Inverting standard (top) and AR (bottom) features using pixel losses.



(c) Inverting standard (top) and AR (bottom) features using pixel and feature losses.

(d) Inverting standard (top) and AR (bottom) features using pixel, feature and GAN losses.

Fig. A2: CNN-based feature inversion of standard and AR representations. AlexNet Conv5 **standard (top)** and **AR (bottom)** features are inverted using an image generator trained on (a) ℓ_1 Pixel loss, (b) Pixel and feature losses, and (c) Pixel, feature and GAN losses.

A2.3 ResNet-18: Robustness Level vs. Reconstruction Accuracy

We take the ResNet-18 model trained on CIFAR-10 from the *Robustness* library [69], invert its third residual block ($4 \times 4 \times 512$) based on our approach using pixel and feature losses, and evaluate its reconstruction accuracy for standard and AR cases.

We measure the reconstruction accuracy for different robustness levels by training six AR classifiers via ℓ_2 PGD attacks (Madry et al.) with attack radii ε covering from 0 to 3.5 (see Tab. A3). Accuracy for each model is measured in terms of PSNR, SSIM and LPIPS. We also report the robustness obtained by each model against ℓ_2 PGD attacks.

Results show the best accuracy is reached for $\varepsilon = 1.5$ in terms of PSNR and for $\varepsilon = 1$ in terms of SSIM and LPIPS. Quality increases almost monotonically for models with low robustness and reaches a peak of approximately 19.62 dB

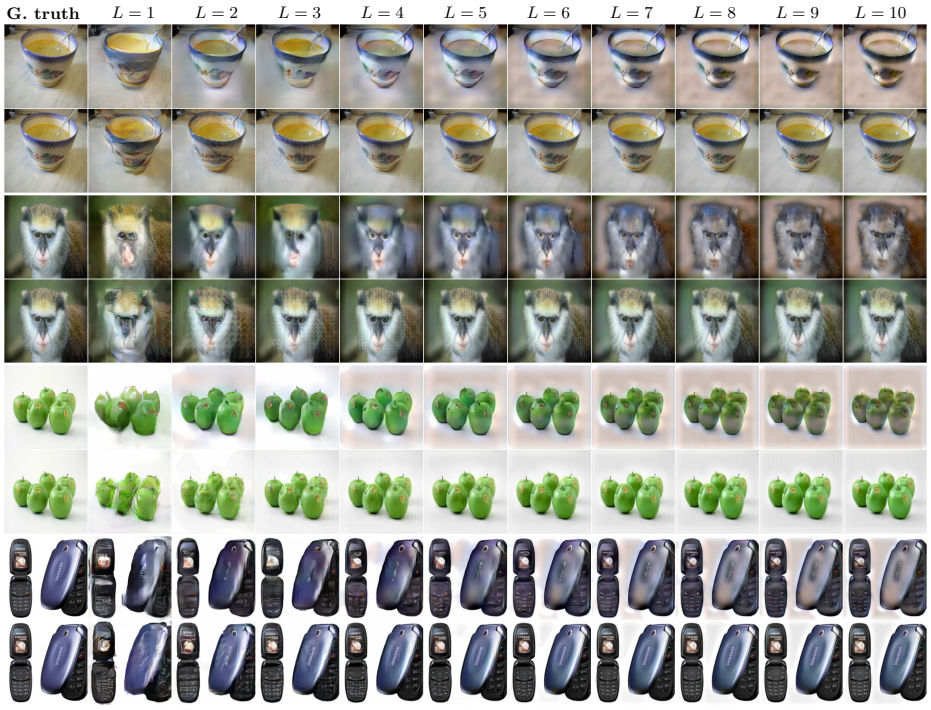


Fig. A3: Reconstructing upscaled images. Upscaled ImageNet samples are inverted from their standard and AR representations. While standard representations (top row) are severely degraded, AR representations (bottom row) show an outstanding accuracy that improves with the scaling factor.

PSNR. Models with higher robustness slowly decrease in accuracy, yet obtaining a significant boost over the standard model ($\varepsilon = 0$).

A2.4 Comparison Against Alternative Methods

Feature inversion accuracy obtained by our proposed model is compared against DeePSiM [19] and RI [23] methods. Fig. A5 illustrates the reconstruction accuracy obtained by each method. As previously explained, our generator yields photorealistic results with 37% the trainable parameters required by the DeePSiM generator. Qualitatively, the color distribution obtained by our AR autoencoder is closer to that obtained by DeepSiM. Specifically, without any postprocessing, DeePSiM's results show severe edge distortions, while out method shows minor edge distortions. On the other hand, the optimization based approach from RI introduces several artifacts, despite its use of robust representations. In contrast, our method takes advantage of AR features and minimizes the distortions in a much more efficient manner by replacing the iterative process by a feature inverter (image generator).

L	Standard AlexNet			Robust AlexNet		
	PSNR (dB)↑	SSIM↑	LPIPS↓	PSNR (dB)↑	SSIM↑	LPIPS↓
1 (224×224)	15.057	0.3067	0.5473	17.2273	0.3580	0.5665
2 (448×448)	16.2777	0.4068	0.4234	20.3554	0.4859	0.469
3 (672×672)	16.0668	0.4317	0.4143	21.3696	0.5265	0.4376
4 (896×896)	15.4258	0.4655	0.4136	22.575	0.5892	0.4012
5 (1120×1120)	14.9726	0.4753	0.4235	22.9861	0.6074	0.4018
6 (1344×1344)	14.3093	0.4887	0.4358	23.4824	0.6527	0.383
7 (1568×1568)	13.8922	0.4852	0.4587	23.5778	0.6588	0.3898
8 (1792×1792)	13.4781	0.4967	0.4656	23.7604	0.70178	0.3638
9 (2016×2016)	13.2869	0.4882	0.4834	23.7907	0.6924	0.3906
10 (2240×2240)	13.1013	0.4969	0.486	23.9566	0.7244	0.3892

Table A2: Reconstructing upscaled images ($L \in \{1, \ldots, 10\}$). Upscaled 224×224 ImageNet samples are reconstructed from standard and AR AlexNet features, the latter predominantly obtaining higher accuracy.

Architecture details and training parameters used to train out proposed model are included in Sec. A4.1. DeePSiM results were obtained using its official Caffe implementation. RI results were obtained using its official PyTorch implementation, modified to invert AlexNet `conv5` layer.

A3 Additional Results on Downstream Tasks

A3.1 Style Transfer

Fig. A6 shows additional stylization results obtained via the Universal Style Transfer algorithm using standard and AR AlexNet autoencoders. Qualitatively, the multi-level stylization approach used in our experiments show that AR representations allow a good texture transferring while better preserving the content image structure. Regardless the type of scene being stylized (*e.g.* landscapes, portraits or single objects), aligning AR robust features allows to preserve sharp edges and alleviates the distortions generated by aligning standard features. Ar-

	ℓ_2 PGD Attack (ε)							
	0	0.5	1	1.5	2	2.5	3	3.5
Standard Accuracy	94.93	88.28	81.07	72.47	64.48	64.17	56.77	53.8
ℓ_2 PGD Attack	28.29 ($\varepsilon = 0.25$)	68.75 ($\varepsilon = 0.5$)	52.24 ($\varepsilon = 1.0$)	41.29 ($\varepsilon = 1.5$)	34.45 ($\varepsilon = 2.0$)	29.63 ($\varepsilon = 2.5$)	25.58 ($\varepsilon = 3.0$)	23.48 ($\varepsilon = 3.5$)
PSNR (dB) ↑	14.7259	18.5161	19.2427	**19.6278**	19.5234	18.7568	19.3713	19.4376
SSIM ↑	0.2958	0.5179	**0.5399**	0.5332	0.5265	0.4878	0.501	0.4951
LPIPS ↓	0.6305	0.5024	**0.4832**	0.4905	0.5019	0.5312	0.5172	0.5321

Table A3: Reconstruction vs. Robustness. ResNet-18 experiments on CIFAR-10 show that learning to invert contracted features with different AR levels significantly affects the reconstruction accuracy.

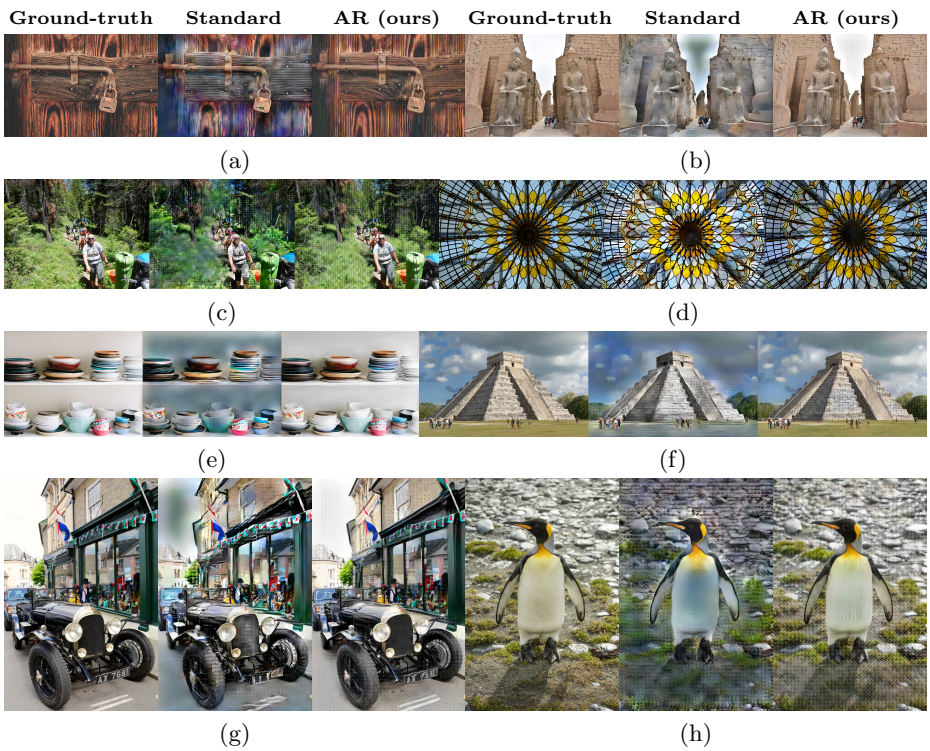


Fig. A4: At a resolution of 2040×1536, 10 times higher than the training resolution, standard reconstructions show color and structure degradation. In contrast, reconstructions from our AR autoencoder do not suffer from such distortions and are closer to target DIV2K images.

chitecture details and training parameters for the style transfer experiments are covered in Sec. A4.2.

A3.2 Image Denoising

Fig. A7 shows additional denoising results using our standard and AR autoencoders for the CBSDS68, Kodak24 and McMaster datasets. As previously discussed, we leverage the low-level feature representations by adding skip connections to our proposed autoencoder. Low-level features complement the contracted feature map obtained from AlexNet `conv5`, improving the detail preservation. This is observed in the results, both with standard and AR autoencoders.

On the other hand, despite the effect of using skip connections, reconstructions from AR representations show a notorious improvement with respect to standard reconstructions. Specifically, by combining skip connections with the rich information already encapsulated in robust representations, results on all three datasets show a substantial denoising improvement.

(a) Ground-truth images.

(b) DeePSiM inversion results [19].

(c) RI results [23].

(d) Standard autoencoder.

(e) Robust autoencoder (ours).

Fig. A5: Feature inversion accuracy contrast between our proposed model and alternative inversion methods.

A4 Implementation Details

A4.1 Architecture and Training Details

Encoder. For all downstream tasks, our adversarially robust AlexNet classifier was trained using PGD attacks [22]. The process was performed on ImageNet using stochastic gradient descent. The AR training parameters are as follows:

- Perturbation constraint: ℓ_2 ball with $\varepsilon = 3$
- PGD attack steps: 7
- Step size: 0.5
- Training epochs: 90

On the other hand, the standard AlexNet classifier was trained using cross-entropy loss as optimization criteria. For both cases, the training parameters were the following:

- Initial learning rate: 0.1
- Optimizer: Learn rate divided by a factor of 10 every 30 epochs.
- Batch size: 256

Tested under AutoAttack ($\ell_2, \varepsilon = 3$), our AR AlexNet obtains a 18.7% top-1 robust accuracy, while our standard AlexNet classifier obtains a 0% top-1 robust accuracy.

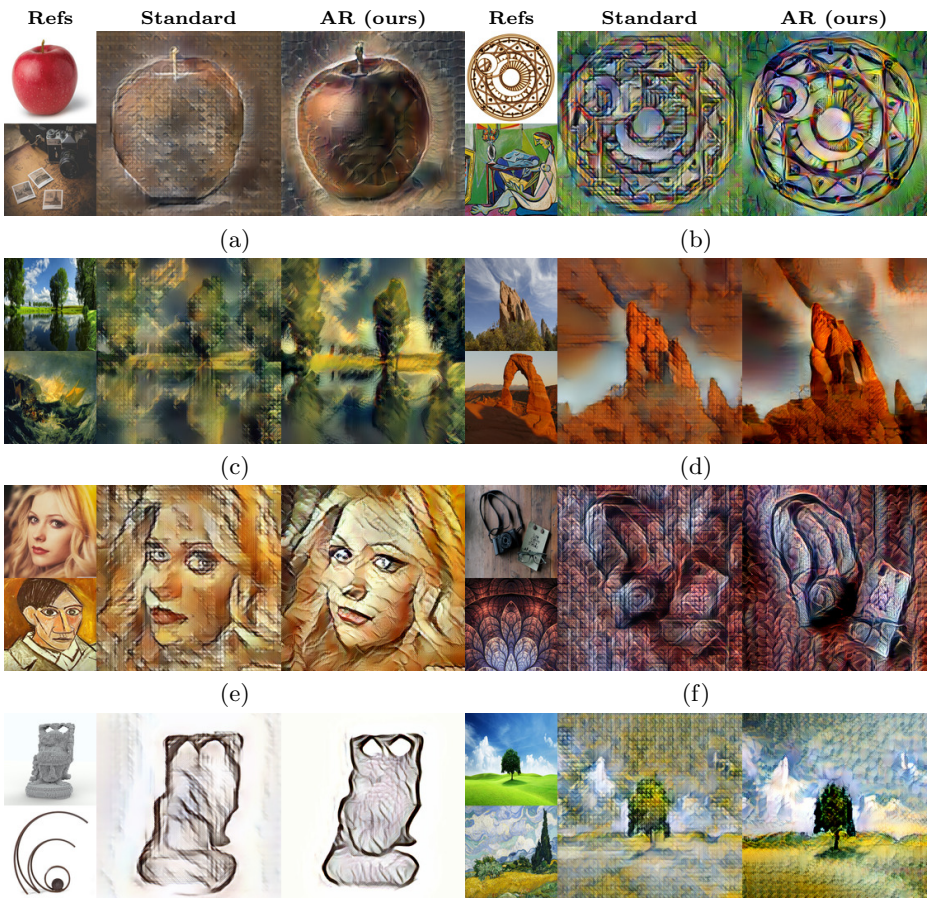


Fig. A6: Style transfer results using standard and robust AlexNet representations. Stylization obtained using the universal style transfer algorithm [2].

AR training was performed using the *Robustness* library [69] on four Tesla V100 GPUs. Additional details about the model architecture and training parameters used for each experiment and downstream task are as follows.

Feature Inversion Experiments. A fully convolutional architecture is used for the decoder or image generator. Tab. A4 describes the decoder architecture used to invert both standard and AR representations, where `conv2d` denotes a 2D convolutional layer, `tconv2d` a 2D transposed convolutional layer, `BN` batch normalization, `ReLU` the rectified linear unit operator and `tanh` the hyperbolic tangent operator.

Tab. A5 shows the discriminator architecture, where `leakyReLU` corresponds to the leaky rectified linear unit, `linear` to a fully-connected layer, `apooling` to average pooling and `sigmoid` to the Sigmoid operator. Motivated by the architecture proposed by Dosovitskiy & Brox [19], the discriminator takes as input both

Layer	Layer Type	Kernel Size	Bias	Stride	Pad	Input Size	Output Size	Input Channels	Output Channels
1a	conv2d + BN + ReLU	3×3	✗	1	1	6×6	6×6	256	256
2a	tconv2d + BN + ReLU	4×4	✗	1	1	6×6	7×7	256	256
2b	conv2d + BN + ReLU	3×3	✗	1	1	7×7	7×7	256	256
3a	tconv2d + BN + ReLU	4×4	✗	2	1	7×7	14×14	256	256
3b	conv2d + BN + ReLU	3×3	✗	1	1	14×14	14×14	256	256
4a	tconv2d + BN + ReLU	4×4	✗	2	1	14×14	28×28	256	256
4b	conv2d + BN + ReLU	3×3	✗	1	1	28×28	28×28	256	128
5a	tconv2d + BN + ReLU	4×4	✗	2	1	28×28	56×56	128	128
5b	conv2d + BN + ReLU	3×3	✗	1	1	56×56	56×56	128	64
6a	tconv2d + BN + ReLU	4×4	✗	2	1	56×56	112×112	64	64
6b	conv2d + BN + ReLU	3×3	✗	1	1	112×112	112×112	64	32
7a	tconv2d + BN + ReLU	4×4	✗	2	1	112×112	224×224	32	32
7b	conv2d + BN + ReLU	3×3	✗	1	1	224×224	224×224	32	3
7c	conv2d + tanh	3×3	✓	1	1	224×224	224×224	3	3

Table A4: Generator architecture used for feature inversion.

a real or fake image and its target conv5 feature map to compute the probability of the sample being real. Fig. A8 shows the discriminator architecture.

Standard and AR autoencoders were trained on ImageNet using ℓ_1 pixel, feature and GAN losses using ADAM. In both cases, all convolutional and transposed convolutional layers are regularized using spectral normalization [70]. Training was performed using Pytorch-only code on two Tesla V100 GPUs.

The loss weights and training setup for both standard and AR cases correspond to:

- Generator weights: $\lambda_{\text{pix}} = 2 \times 10^{-6}, \lambda_{\text{feat}} = 1 \times 10^{-2}, \lambda_{\text{GAN}} = 100$
- Discriminator weight: $\lambda_{\text{disc}} = 2 \times 10^{-6}$
- Training epochs: 90
- Generator initial learning rate: 3×10^{-4} (divided by a factor of 10 every 30 epochs).
- Discriminator initial learning rate: 12×10^{-4} (divided by a factor of 10 every 30 epochs).
- LeakyReLU factor: 0.2
- ADAM $\beta \in [0, 0.9]$
- Batch size: 128

A4.2 Style Transfer

While, for standard and AR scenarios, the autoencoder associated to conv5 corresponds to the model described in Sec. A4.1, those associated to conv1 and conv2 use Nearest neighbor interpolation instead of transposed convolution layers to improve the reconstruction accuracy and to avoid the checkerboard effect generated by transposed convolutional layers. Tab. A6, and Tab. A7 describe their architecture details.

Layer	Layer Type	Kernel Size	Bias	Stride	Pad	Input Size	Output Size	Input Channels	Output Channels
Feature Extractor 1 (D_1)									
1a	conv2d + ReLU	3×3	✓	4	1	256×256	56×56	3	32
2a	conv2d + ReLU	5×5	✓	1	1	56×56	52×52	32	64
2b	conv2d + ReLU	3×3	✓	2	1	52×52	23×23	64	128
3a	conv2d + ReLU	3×3	✓	1	1	23×23	21×21	128	256
3b	conv2d + ReLU	3×3	✓	2	1	21×21	11×11	256	256
4	ave. pooling	11×11	–	–	–	11×11	1×1	256	256
Classifier 1 (D_2)									
4a	Linear + ReLU	–	✓	–	1	9216	1024	–	–
4b	Linear + ReLU	–	✓	–	1	1024	512	–	–
Classifier 2 (D_3)									
5a	Linear + ReLU	–	✓	–	1	768	512	–	–
5b	Linear + Sigmoid	–	✓	–	1	512	1	–	–

Table A5: Discriminator architecture used for feature inversion.

Layer	Layer Type	Kernel Size	Bias	Stride	Pad	Input Size	Output Size	Input Channels	Output Channels
1a	conv2d + BN + ReLU	3×3	✗	1	1	27×27	27×27	64	64
2a	tconv2d + BN + ReLU	4×4	✗	1	1	27×27	28×28	64	64
2b	conv2d + BN + ReLU	3×3	✗	1	1	28×28	28×28	64	64
3a	NN interpolation	–	–	2	–	28×28	56×56	64	64
3b	conv2d + BN + ReLU	3×3	✗	1	1	56×56	56×56	64	64
3c	conv2d + BN + ReLU	3×3	✗	1	1	56×56	56×56	64	32
4a	NN interpolation	–	–	2	–	56×56	112×112	32	32
4b	conv2d + BN + ReLU	3×3	✗	1	1	112×112	112×112	32	32
5a	NN interpolation	–	–	2	–	112×112	224×224	32	32
5b	conv2d + BN + ReLU	3×3	✗	1	1	224×224	224×224	32	16
5c	conv2d + BN + ReLU	3×3	✗	1	1	224×224	224×224	16	3
5d	conv2d + tanh	3×3	✓	1	1	224×224	224×224	3	3

Table A6: Conv1 generator architecture used for style transfer.

All generators were fully-trained on ImageNet using Pytorch-only code on two Tesla V100 GPUs. The regularization parameters and training setup for both cases are as follows:

- Standard generator weights: $\lambda_{\text{pix}} = 2 \times 10^{-4}, \lambda_{\text{feat}} = 1 \times 10^{-2}$.
- AR generator weights: $\lambda_{\text{pix}} = 2 \times 10^{-6}, \lambda_{\text{feat}} = 1 \times 10^{-2}$.
- Training epochs: 90.
- Generator initial learning rate: 3×10^{-4} (divided by a factor of 10 every 30 epochs).
- ADAM $\beta \in [0, 0.9]$.
- Batch size: 128.

Layer	Layer Type	Kernel Size	Bias	Stride	Pad	Input Size	Output Size	Input Channels	Output Channels
1a	conv2d + BN + ReLU	3×3	✗	1	1	13×13	13×13	192	192
2a	tconv2d + BN + ReLU	4×4	✗	1	1	13×13	14×14	192	192
2b	conv2d + BN + ReLU	3×3	✗	1	1	14×14	14×14	192	96
3a	NN interpolation	–	–	2	–	14×14	28×28	96	96
3b	conv2d + BN + ReLU	3×3	✗	1	1	28×28	28×28	96	96
3c	conv2d + BN + ReLU	3×3	✗	1	1	28×28	28×28	96	64
4a	NN interpolation	–	–	2	–	28×28	56×56	64	64
4b	conv2d + BN + ReLU	3×3	✗	1	1	56×56	56×56	64	64
5a	NN interpolation	–	–	2	–	56×56	112×112	64	64
5b	conv2d + BN + ReLU	3×3	✗	1	1	112×112	112×112	64	64
6a	NN interpolation	–	–	2	–	112×112	224×224	64	64
6b	conv2d + BN + ReLU	3×3	✗	1	1	224×224	224×224	64	32
6c	conv2d + BN + ReLU	3×3	✗	1	1	224×224	224×224	32	3
6d	conv2d + tanh	3×3	✓	1	1	224×224	224×224	3	3

Table A7: Conv2 generator architecture used for style transfer.

A4.3 Image Denoising

Our image denoising model consists of standard and AR autoencoders equipped with skip connections to better preserve image details. Fig. A9 illustrates the proposed denoising model, where skip connections follow the Wavelet Pooling approach [3]. Tab. A8 and Tab. A9 include additional encoder and decoder architecture details, respectively.

Encoder pooling layers are replaced by Haar wavelet analysis operators, generating an approximation component, denoted as $\{w_{k,\mathrm{LL}}\}$, and three detail components, denoted as $\{w_{k,\mathrm{LH}}, w_{k,\mathrm{HL}}, w_{k,\mathrm{HH}}\}$, where k corresponds to the pooling level. While the approximation (low-frequency) component is passed to the next encoding layer, details are skip-connected to their corresponding stages in the decoder. Following this, transposed convolutional layers in the decoder are replaced by unpooling layers (Haar wavelet synthesis operators), reconstructing a signal with well-preserved details at each level and improving reconstruction.

In contrast to the AlexNet architecture, all convolutional layers on the decoder use kernels of size 3×3. Also, given the striding factor of the first two AlexNet convolutional layers, two additional interpolation layers of striding factor 2 are used to recover the original input size (224×224).

Standard and AR robust generators were trained using exclusively ℓ_1 pixel and feature losses. Training was performed on ImageNet using Pytorch-only code on four Tesla V100 GPUs. Generator loss weights and training parameters for both cases correspond to:

- Generator weights: $\lambda_{\mathrm{pix}} = 2 \times 10^{-6}, \lambda_{\mathrm{feat}} = 1 \times 10^{-2}$.
- Training epochs: 90.
- Generator initial learning rate: 3×10^{-4} (divided by a factor of 10 every 30 epochs).
- ADAM $\beta \in [0, 0.9]$.
- Batch size: 128.

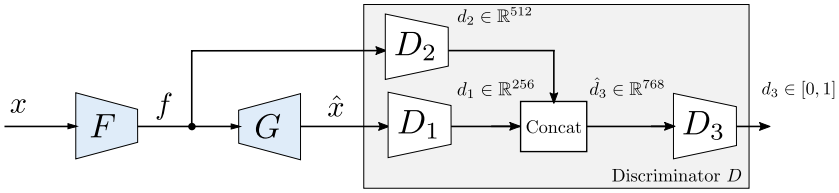
Layer	Layer Type	Kernel Size	Bias	Stride	Pad	Input Size	Output Size	Input Channels	Output Channels
1a	conv2d + ReLU	11×11	✓	4	2	224×224	55×55	3	64
2a	Wavelet pooling	–	–	2	–	55×55	27×27	64	64
2b	conv2d + ReLU	5×5	✓	1	2	27×27	27×27	64	192
3a	Wavelet pooling	–	–	2	–	27×27	13×13	192	192
3b	conv2d + ReLU	3×3	✓	1	1	13×13	13×13	192	384
3c	conv2d + ReLU	3×3	✓	1	1	13×13	13×13	384	256
3c	conv2d + ReLU	3×3	✓	1	1	13×13	13×13	256	256
4a	Wavelet pooling	–	–	2	–	13×13	6×6	256	256

Table A8: Encoder architecture used for image denoising.

	Layer Type	Kernel Size	Bias	Stride	Pad	Input Size	Output Size	Input Channels	Output Channels
1a	conv2d + BN + ReLU	3×3	✗	1	1	6×6	6×6	256	256
2a	Wavelet unpooling	–	–	2	–	6×6	12×12	256	256
2b	conv2d + BN + ReLU	3×3	✗	1	1	12×12	12×12	256	256
2c	Reflection padding	–	–	–	–	12×12	13×13	256	256
2d	conv2d + BN + ReLU	3×3	✗	1	1	13×13	13×13	256	256
2e	conv2d + BN + ReLU	3×3	✗	1	1	13×13	13×13	256	192
3a	Wavelet unpooling	–	–	2	–	13×13	26×26	192	192
3b	Reflection padding	–	–	–	–	26×26	27×27	192	192
3c	conv2d + BN + ReLU	3×3	✗	1	1	27×27	27×27	192	128
3d	conv2d + BN + ReLU	3×3	✗	1	1	27×27	27×27	128	64
4a	Wavelet unpooling	–	–	2	–	27×27	55×55	64	64
4b	Reflection padding	–	–	–	–	55×55	56×56	64	64
4c	conv2d + BN + ReLU	3×3	✗	1	1	56×56	56×56	64	64
5a	NN interpolation	–	–	2	–	56×56	112×112	64	64
5b	conv2d + BN + ReLU	3×3	✗	1	1	112×112	112×112	64	32
5c	conv2d + BN + ReLU	3×3	✗	1	1	112×112	112×112	32	32
6a	NN interpolation	–	–	2	–	112×112	224×224	32	32
6b	conv2d + BN + ReLU	3×3	✗	1	1	224×224	224×224	32	3
6c	conv2d + BN + ReLU	3×3	✗	1	1	224×224	224×224	3	3
6d	conv2d + tanh	3×3	✓	1	1	224×224	224×224	3	3

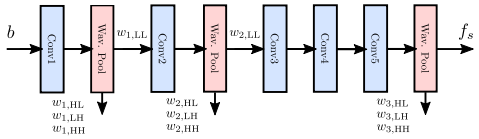
Table A9: Decoder architecture used for image denoising.

Fig. A7: Image denoising results using standard and AR encoders (AlexNet) from the CBSD68 and Kodak24 sets. Samples corrupted by clipped white Gaussian noise ($\sigma = \frac{50}{255}$).

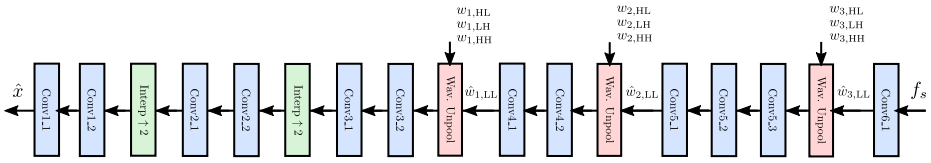


(a)

Fig. A8: Discriminator model.



(a) Skip connected AlexNet encoder



(b) Skip connected AlexNet decoder

Fig. A9: Proposed denoising autoencoder including skip connections.