

Program-Guided Image Manipulators

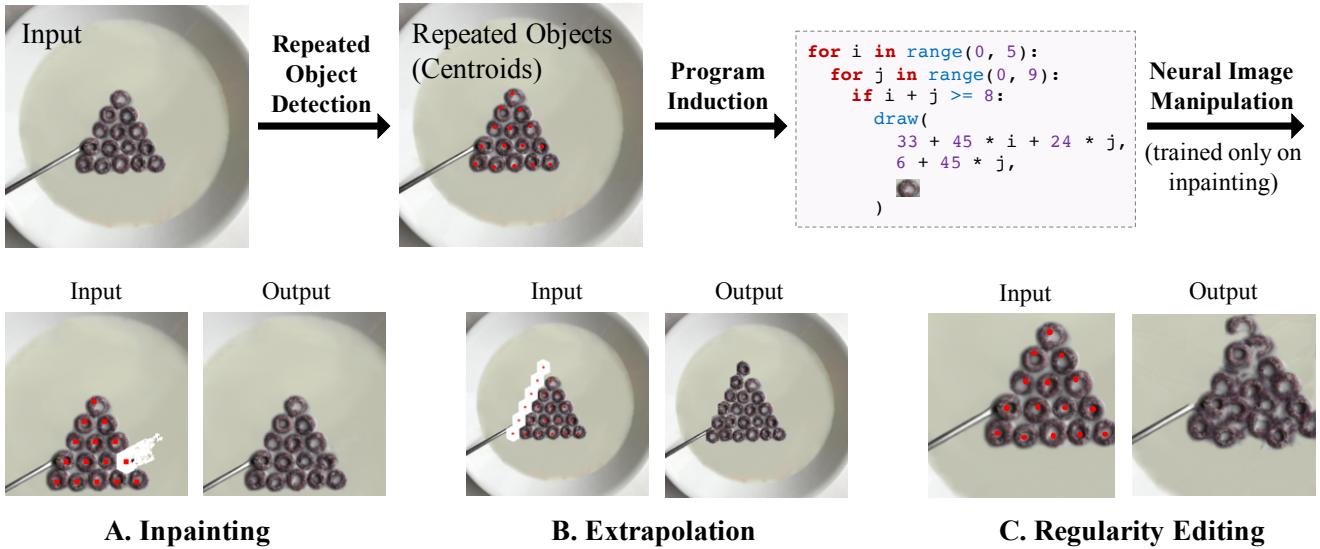
Jiayuan Mao^{1*}William T. Freeman^{1,3}Xiuming Zhang^{1*}Joshua B. Tenenbaum¹Yikai Li^{1,2}Jiajun Wu¹¹MIT CSAIL²Shanghai Jiao Tong University³Google Research<http://pgim.csail.mit.edu>

Figure 1: Given an input image, the Program-Guided Image Manipulator (PG-IM) detects repeated entities in the image (pieces of cereal) and then infers a program-like representation for describing the regularity of the image. The regularity representation empowers multiple downstream tasks, such as image inpainting, extrapolation, and regularity editing.

Abstract

Humans are capable of building holistic representations for images at various levels, from local objects, to pairwise relations, to global structures. The interpretation of structures involves reasoning over repetition and symmetry of the objects in the image. In this paper, we present the Program-Guided Image Manipulator (PG-IM), inducing neuro-symbolic program-like representations to represent and manipulate images. Given an image, PG-IM detects repeated patterns, induces symbolic programs, and manipulates the image using a neural network that is guided by the program. PG-IM learns from a single image, exploiting its internal statistics. Despite trained only on image inpainting, PG-IM is directly capable of extrapolation and regularity editing in a unified framework. Extensive experiments show that PG-IM achieves superior performance on all the tasks.

1. Introduction

Looking at the images in Figure 1, we effortlessly identify the objects (pieces of cereal) in the image, interpret their pairwise relations, and reason over the global *regularity*:

all pieces of cereal are organized on a 2D lattice with a triangular boundary. This holistic representation empowers our imagination of unseen objects: we can inpaint missing pixels in images, extrapolate images while preserving the regularity [33], and reduce or exaggerate the regularity.

While tremendous progress has been made in object recognition [17] and visual relation detection [27], a global representation for *structural regularity* is still missing in these studies. In this paper, we propose to augment deep networks, which are very powerful in pixel-level recognition, with symbolic programs, which are flexible to capture high-level regularity within the image. The intuition is that the disentanglement between perception and reasoning will enable complex image manipulation, preserving both high-level scene structure and low-level object appearance.

Our model, the Program-Guided Image Manipulator (PG-IM), induces symbolic programs for global regularities and manipulates images with deep generative models guided by the programs. PG-IM consists of three modules: a neural module that detects repeated patterns within the input image, a symbolic program synthesizer that infers programs for spatial regularity (lattice structure) and content regular-

* equal contribution; order determined by a coin toss.

ity (object attributes), and a neural generative model that manipulates images based on the inferred programs.

We demonstrate the effectiveness of PG-IM on two datasets: the Nearly-Regular Pattern dataset [22] and the Facade dataset [39]. Both datasets contain nearly-regular images with lattice patterns of homogeneous objects. We also extend our experiments to a collection of Internet images with non-lattice patterns and variations in object appearance. Our neuro-symbolic approach robustly outperforms neural and patch-matching-based baselines on multiple image manipulation tasks, such as inpainting, extrapolation, and regularity editing.

2. Related Work

Image manipulation. Image manipulation is a long-standing problem in computer vision, graphics, and computational photography, most often studied in the context of image inpainting. Throughout decades, researchers have developed numerous inpainting algorithms operating at various levels of image representations: pixels, patches, and most recently, holistic image features learned by deep networks. Pixel-based methods often rely on diffusion [2, 5] and work well when the holes are small; later, patch-based methods [11, 6] accelerate pixel-based methods and achieve better results. Both methods do not perform well in cases that require high-level information beyond background textures.

Deep networks are good at learning semantics from large datasets, and the learned semantic information has been applied to image manipulation [42, 31, 40]. Many follow-ups have been proposed to improve the results via multi-scale losses [19, 45], contextual attention [47], partial convolution [25], gated convolution [48], among others [50, 44]. Although these methods achieve impressive inpainting results with the learned semantic knowledge, they have two limitations: first, they rely on networks to learn object structure implicitly, and may fail to capture explicit, global object structures, such as the round shape of a clock [43]; second, the learned semantics is specific to the training set, while real-world test images are likely to be out-of-distribution. Very recently, Xiong et al. [43] and Nazeri et al. [29] tackled the first problem by explicitly modeling contours to help the inpainting system preserve global object structures. In this paper, we propose to tackle both problems using a combination of bottom-up deep recognition networks and the top-down neuro-symbolic program induction. We apply our approach to scenes with an arbitrary number of objects.

Program induction and procedural modeling. The idea of using procedural modeling for visual data has been a well-studied topic in computer graphics, mostly for indoor scenes [41, 24, 30] and 3D shapes [23]. More recently, with deep recognition networks, researchers have studied converting 2D images to line-drawing programs [12], primitive sets [36], markup code [10, 7], or symbolic programs with attributes [26]. These papers tackle synthetic images in a

constrained domain, while here we study natural images.

SPIRAL [15] used reinforcement learning to derive “drawing commands” for natural images. Their commands are, however, not interpretable, and it is unclear how they can be extended to handle complex relations among a set of objects. Most recently, Young et al. [46] integrated formal representations with deep generative networks and applied it to natural image inpainting. Still, our model differs from theirs in two aspects. First, we use neural modules for discovering repeated patterns in images, which does not require the patch of interest to repeat itself over the entire image (an assumption made in [46]). Second, their algorithm requires learning semantics on a pre-defined dataset for manipulation (e.g., image extrapolation); in contrast, our model exploits the idea of internal learning [38] and requires no training data during image manipulation other than the image itself.

Single-image learning. Because visual entropy inside a single image is lower than in a diverse collection of images [51], many works have exploited image-level (instead of dataset-level) statistics for various image editing tasks including deblurring [3, 28], super-resolution [16, 13, 18], and dehazing [4]. The same philosophy has also been proven successful in deep learning, where neural networks are trained on (and hence overfit to) a single image. Such image-specific networks effectively encode image priors unique to the input image [40]. They can be used for super-resolution [38], layer decomposition [14], texture modeling [8, 50], and even generation tasks [35, 37].

Powerful as these approaches are, they often lack a high-level understanding of the input image’s global structure (such as the triangular shape formed by the cereal in Figure 1). Consequently, there is usually no guarantee that the original structure gets preserved after the manipulation (e.g., Row 2 of Figure 5). This work augments single-image learning methods with symbolic reasoning about the input image’s global structure, not only providing a natural way of preserving such structure, but also enabling higher-level, semantic manipulation based on the structure (e.g., extrapolating an additional row of cereal following the triangular structure in the teaser figure).

3. Program-Guided Image Manipulator

The Program-Guided Image Manipulator (PG-IM) contains three modules, as shown in Figure 1. First, PG-IM detects repeated objects and make them a variable-length stack (Section 3.1). Then, it infers a program to describe the global regularity among the objects (Section 3.2), with program tokens such as for-loops for repetition and symmetry. Finally, the inferred program facilitates image manipulation, which is performed by a neural painting network (Section 3.3).

3.1. Repeated Object Detection

PG-IM detects repeated objects in the input image with a neural module based on Lettry et al. [22]. Given the input im-

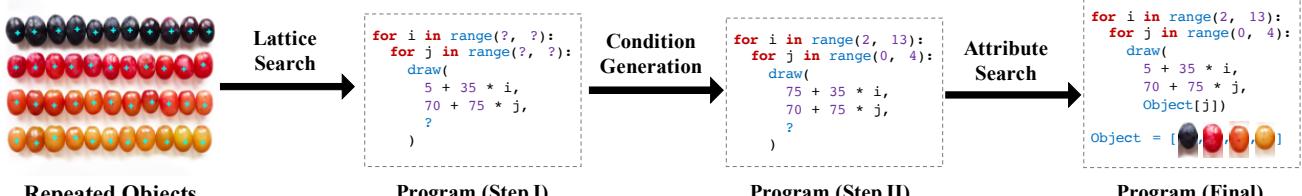


Figure 2: The three-step inference of a program describing the shown repeated pattern. Assuming the input keypoints follow a lattice pattern, we first search for parameters defining the lattice, such as the distance between nearby keypoints and the origin. Next, we fit boundary conditions for the program. Finally, we cluster objects into groups by their visual appearance, and fit an expression describing the variation.

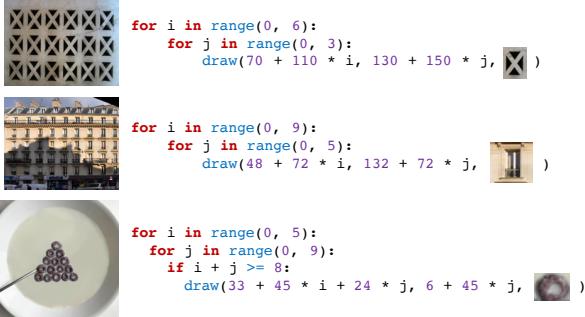


Figure 3: Illustrative programs inferred from (top row) the Nearly-Regular Pattern dataset [22], (middle row) the Facade dataset [39], and (bottom row) Internet images. The DSL of the inferred programs supports for-loops, conditions, and attributes.

age, it extracts convolutional feature maps from a pre-trained convolutional neural network (i.e., AlexNet [21]). A morphological filter is then applied to the feature maps for extracting activated neurons, resulting in a stack of *peakmaps*. Next, assuming the lattice pattern of repeated objects, a voting algorithm is applied to compute the displacements between nearby objects. Finally, an implicit pattern model (IPM) is employed to fit the centroids of objects. Please see [22] and the supplementary material for details of the algorithm.

3.2. Program Synthesizer

The program synthesizer takes the centroids of the repeated objects as input and infers a latent program describing the pattern. The input image is partitioned into object patches by constructing a Voronoi graph of all pixels. That is, each pixel is assigned to its nearest centroid, under the metric of Euclidean distance between pixel coordinates. Meanwhile, objects are clustered into multiple groups. When the program reconstructs an object with the `Draw` command, it is allowed to specify both the coordinate of the object’s centroid (x, y) and an integer (namely, the *attribute*), indicating which group this object belongs to. We implement our program synthesizer as a search-based algorithm that finds the *simplest* program that reconstructs the pattern.

Domain-specific language. We summarize the domain-specific language (DSL) used by PG-IM for describing object repetition in Table 1. In a nutshell, `ForStmt1` and `ForStmt2` jointly define a lattice structure; `CondDrawExpr` defines the boundary of the lat-

Program	\rightarrow	ForStmt
ForStmt	\rightarrow	For (i in range(Integer, Integer)) { For2Stmt }
For2Stmt	\rightarrow	For (i in range(Integer, Integer)) { CondDrawStmt }
CondDrawStmt	\rightarrow	If (Expr \geq 0) { CondDrawStmt }
CondDrawStmt	\rightarrow	DrawStmt
DrawStmt	\rightarrow	Draw (x=Expr, y=Expr, attribute=AttributeExpr)
AttributeExpr	\rightarrow	Expr // Integer
AttributeExpr	\rightarrow	1 If (Expr == 0) else 0
AttributeExpr	\rightarrow	1 If (Expr == 0 and Expr == 0) else 0
AttributeExpr	\rightarrow	1 If (Expr % Integer == 0) else 0
AttributeExpr	\rightarrow	1 If (Expr % Integer == 0 and Expr % Integer == 0) else 0
Expr	\rightarrow	Integer * i + Integer * j + Integer

Table 1: The domain-specific language (DSL) for describing image regularities. Language tokens including `For`, `If`, `Integer` and arithmetic/logical operators follow the convention of Python.

tice; `Draw` places an object at a given coordinate. `AttributeExpr` allows the attribute of the object to be conditioned on the loop variables (i and j). Figure 3 shows illustrative programs inferred from different datasets.

Program search. Finding the *simplest* program for describing a regularity pattern involves searching over a large compositional space of possible programs, which contains for-loops, if-conditions, coordinate expressions, and attribute expressions. To accelerate the search, we heuristically divides the search process into three steps, as illustrated in Figure 2. First, we search over all possible expressions for the coordinates, and find the one that fits the detected centroids the best. Second, we determine the conditions (the boundary). Finally, we find the expression for attributes.

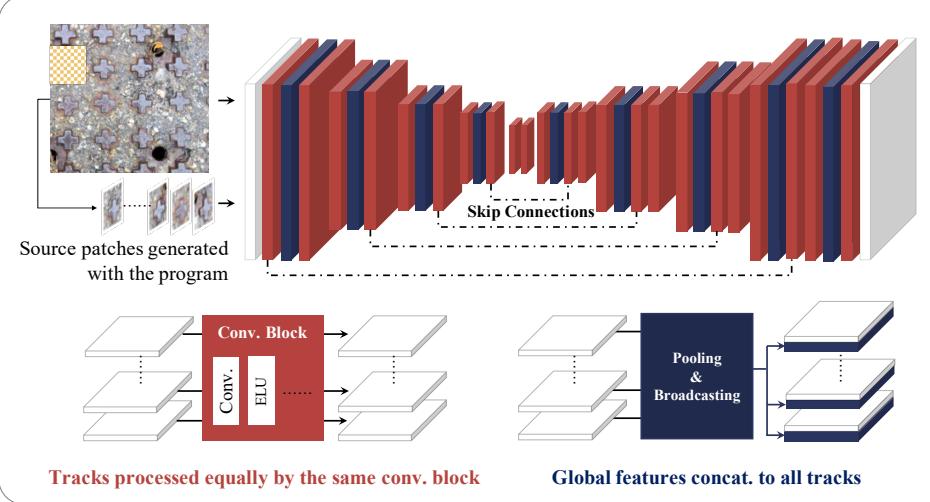
Lattice search. The lattice search finds the expressions for coordinates x and y , ignoring all potential conditions and attribute expressions. Thus, the search process can be simplified as finding a 5-tuple $(b_x, b_y, d_{x,i}, d_{x,j}, d_{y,j})$ that satisfies $x = b_x + i \cdot d_{x,i} + j \cdot d_{x,j}$ and $y = b_y + j \cdot d_{y,j}$.

Each tuple defines a set of centroids \mathbf{P} containing all (x, y) pairs whose coordinates are within the boundary of the whole image. We compare these sets with the centroids \mathbf{C} detected by the repeated pattern detector. We find the optimal tuple as the one that minimizes a cost function

$$\mathcal{L}_{\text{lat}} = \sum_{(x,y) \in \mathbf{C}} \min_{(u,v) \in \mathbf{P}} [(x-u)^2 + (y-v)^2] + \lambda |\mathbf{P}|, \quad (1)$$

where $\lambda = 5$ is a hyperparameter for regularization. It matches each detected centroid with the nearest one recon-

Trained on Inpainting



Tested on Three Tasks

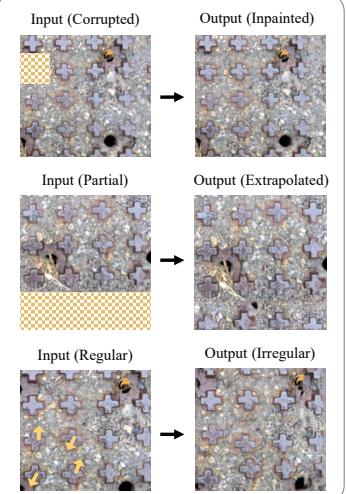


Figure 4: A neural painting network (NPN) takes as input an image and a set of source patches, derived from the image with its program description, and outputs a manipulated image. An NPN learns from a single image, exploiting the image’s internal statistics. Trained only on inpainting, it can directly extrapolate and edit the regularity of the input image in a unified inference framework, without any finetuning.

structed by the program. The goal is to minimize the distance between them and a regularization term over the size of \mathbf{P} . From a Bayesian inference perspective, \mathbf{P} defines a mixture of Gaussian distribution over the 2D plane. \mathcal{L}_{lat} approximates the log-likelihood of the observation \mathbf{C} and a prior distribution over possible \mathbf{P} ’s, which favors small ones.

Condition search. In the next step, we generate the conditions of the program, assuming all centroids fit in a convex hull. This assumption covers both rectangular lattices and triangular lattices (see Figure 3 for examples). Since all pairs (x, y) are computed by an affine transformation of all (i, j) ’s, the conditions can be determined by computing the convex hull of all (i, j) ’s that are matched with detected centroids.

Specifically, we first match each coordinate in \mathbf{P} with \mathbf{C} by computing a minimum cost assignment between two sets, where the distance metric is the Euclidean distance in the 2D coordinate space. We then find the convex hull of all assigned pairs (i, j) . We use the boundary of the convex hull as the conditions. The conditions include the boundary conditions of for-loops as well as optional if-conditions.

Attribute search. The last step is to find the expression that best describes the variance in object appearance (i.e., their attributes). Attributes are represented as a set of integers. Instead of clustering, we assign discrete labels to individual patches. The label of the patch in row p_i , column p_j is a function of (p_i, p_j) . Shown in Table 1, each possible AttributeExpr defines an attribute assignment function $A(p) \triangleq A(p_i, p_j)$ for all centroids $p = (p_i, p_j) \in \mathbf{P}$. We say an expression fits the image if patches of the same label share similar visual appearance. Formally, we find the optimal parameters for the attribute expression that minimizes

$$\mathcal{L}_{\text{attr}} = \sum_{p \in \mathbf{P}} \sum_{q \in \mathbf{P}} (\text{sgn}(A(p), A(q)) \cdot d(p, q)) + \mu |A(\mathbf{P})|, \quad (2)$$

where $\text{sgn}(A(p), A(q)) = 1$ if $A(p) = A(q)$, and -1 otherwise. d computes the pixel-level difference between two patches centered at (p_i, p_j) and (q_x, q_y) , respectively. $\mu = 10$ is a scalar hyperparameter of the regularization strength. $|A(\mathbf{P})|$ computes the number of distinct values of $A(p)$ for all $p \in \mathbf{P}$. The inference is done by searching over possible integer templates (e.g., $ai + bj + c$) and binary templates (e.g., $(ai + bj + c \text{ } \& \text{ } d \% e) == 0$), and the coefficients (a, b, c, \dots) .

3.3. Neural Painting Networks

We propose the neural painting network (NPN), a neural architecture for manipulating images with the guidance of programs. It unifies three tasks: inpainting, extrapolation, and regularity editing in a single framework. The key observation is that all three tasks can be cast as filling pixels in images. For illustrative simplicity, we first consider the task of inpainting missing pixels in the image, and then discuss how to perform extrapolation and regularity editing using the same inpainting-trained network.

Patch aggregation. We first aggregate all pixels from other “objects” (loosely defined by the induced program) to inpaint the missing pixels. Denote all object centroids reconstructed by the program as \mathbf{P} , the centroid of the object patch containing missing pixels (x_0, y_0) , and all other centroids $\mathbf{P}^- = \mathbf{P} \setminus \{(x_0, y_0)\}$. The aggregation is performed by generating $|\mathbf{P}^-|$ images, the i -th of which is obtained by translating the original image such that the centroid of the i -th object in \mathbf{P}^- is centered at (x_0, y_0) . Pixels without a value after the shift are treated as 0. We stack the input image with missing pixels plus all the $|\mathbf{P}^-|$ images (the “patch source”) as the input to the network.

Architecture. Our neural painting network (NPN) has a U-Net [34] encoder-decoder architecture, designed to handle

a variable number of input images and be invariant to their ordering. Demonstrated in Figure 4, the network contains a stack of shared-weight convolution blocks and max-pooling layers that aggregate information across all inputs. Paired downsampling and upsampling layers (convolution layers with strides) are skip-connected. The input of the network is the stack of the corrupted input image plus source patches, and the output of the network is the inpainted image. A detailed printout of the generator’s architecture can be found in the supplemental document.

The key insight of our design of the NPN is that it handles a variable number of input images in any arbitrary order. To this end, inspired by Aittala et al. [1] and Qi et al. [32], we have a single encoder-decoder that processes the $|P^-| + 1$ images equally (“tracks”), and the intermediate feature maps from these tracks get constantly max-pooled into a “global” feature map, which is then broadcast back to the $|P^-| + 1$ tracks and concatenated to each track’s local feature map to be processed by the next block. Intuitively, the network is guided to produce salient feature maps that will “survive” the max-pooling, and the tracks exchange information by constantly absorbing the global feature map.

Extrapolation and regularity editing as recurrent inpainting. A key feature of program-guided NPNs is that although they are trained only on the inpainting task, they are able to be used directly for image extrapolation and regularity editing. With the program description of the image, NPNs are aware of where the entities are in the image, and hence able to cast extrapolation as recurrent inpainting of multiple corrupted objects. For instance, to extrapolate a 64-pixel wide stripe to the right, an NPN first queries the program description for where the new peaks are, and then recurrently inpaints each object given all the previously inpainted ones. Similarly for image regularity editing, when the (regularly spaced) centroids provided by the program get randomly perturbed, the pixels falling into their Voronoi cells move together with them accordingly, leaving many “cracks” on the image, which the NPN then inpaints recurrently.

Training. We train our NPNs with the same training paradigm as Isola et al. [20]. We compute an L1 loss and a patch-based discriminator loss, between the generated (inpainted) image and the ground-truth image. We train image-specific NPNs for each individual image in the dataset. While only training the network to inpaint missing pixels, we show that the network can perform other tasks such as image extrapolation and regularity editing, by only changing the input to the network during inference. Other implementation details such as the hidden dimensions, convolutional kernel sizes, and training hyperparameters can be found in the supplementary material.

4. Experiments and Applications

We provide both quantitative and qualitative comparisons with the baselines on two standard image manipulation tasks:

inpainting and extrapolation. We also show the direct application of our approach to image regularity editing, a task where the regularity of an image’s global structure gets exaggerated or reduced. It is worth mentioning that these three problems can be solved with a single model trained for inpainting (see Section 3.3 for details). Finally, we demonstrate how our program induction easily incorporates object attributes (e.g., colors) in Internet images, in turn enabling our NPNs to manipulate images with high-level reasoning in an attribute-aware fashion. Please see the supplemental material for ablation studies that evaluate each major component of PG-IM. We start with an introduction to the datasets and baseline methods we consider.

4.1. Dataset

We compare the performance of PG-IM with other baselines on two datasets: the Nearly-Regular Pattern (NRP) dataset [22] and the Facade dataset [39]. The Nearly-Regular Pattern dataset contains a collection of 48 rectified images with a grid or nearly grid repetition structure. The Facade dataset, specifically the CVPR 2010 subset, contains 109 rectified images of facades.

4.2. Baselines

We consider two groups of baseline methods: non-learning-based and learning-based. Among the non-learning-based methods are Image Quilting [11] and PatchMatch [6], both of which are based on the stationary assumption of the image structure. Intuitively, to inpaint a missing pixel, they fill it with the content of another existing pixel with the most similar context. Being unaware of the objects in the image, they rely on human-specified hyperparameters, such as the context window size, to produce reliable results. More importantly, in the case of extrapolation, the user needs to specify which pixels to paint, implicitly conveying the concept of objects to the algorithms. For PatchMatch and Image Quilting, we search for one set of optimal hyperparameters and apply that to the entire test set.

We also compare PG-IM with a learning-based, off-the-shelf algorithm for image inpainting: GatedConv [48]. They use neural networks for inpainting missing pixels by learning from a large-scale dataset (Place365 [49]) of natural images. GatedConv is able to generate novel objects that do not appear in the input image, which is useful for semantic photo editing. However, this may not be desired when the image of interest contains repeated but *unique* patterns: although a pattern appears repeatedly in the image of interest, it may not appear anywhere else in the dataset. Therefore, we also consider another learning-based baseline, originally designed for image extrapolation: Non-Stationary Texture Synthesis (Non-Stationary) [50]. In their framework, an image-specific neural network is trained for each input image. Its objective is to extrapolate a small (usually unique) patch ($k \times k$) into a large one ($2k \times 2k$). Although both of their method

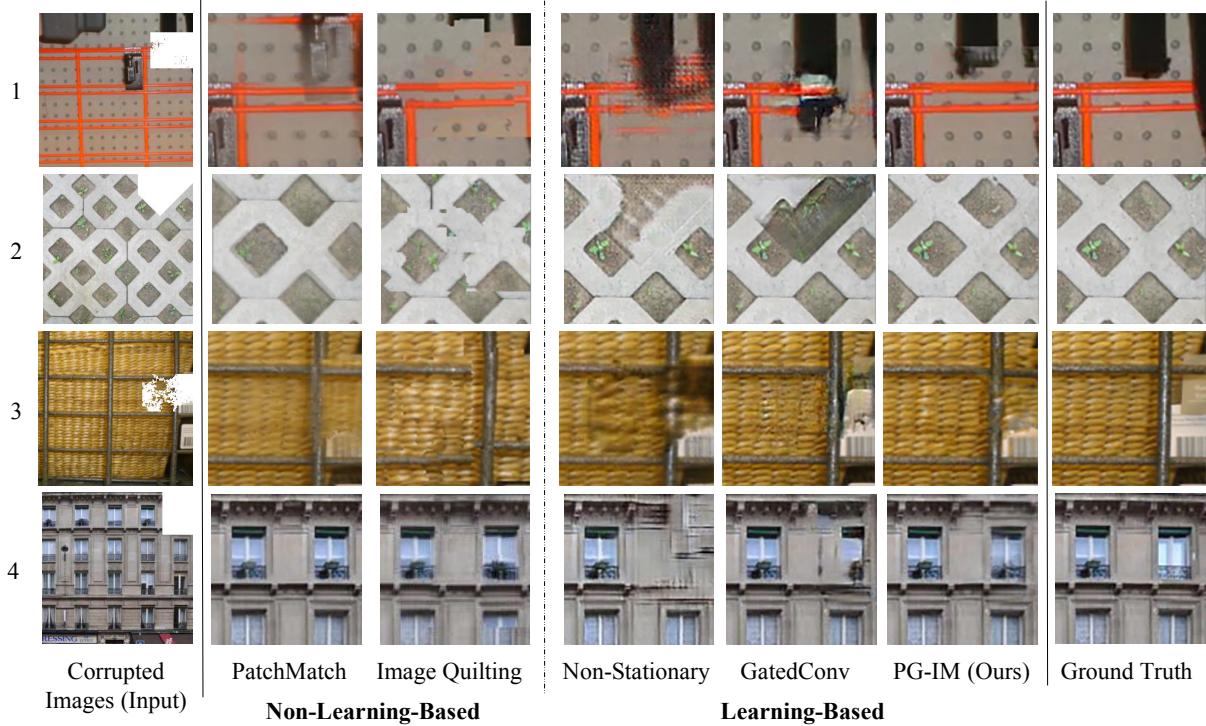


Figure 5: Corrupted input images and inpainting results (zoomed-in) by PG-IM and the baselines. The white pixels in the leftmost column are missing pixels to inpaint. The rightmost column shows the ground-truth patches. PG-IM inverts realistic image patches that are consistent with the intricate global regularity and meanwhile different from the original, ground-truth patches.

Method	L1 Mean (Std.)	Inception Score
Nearly-Regular Patterns [22]		
Image Quilting [11]	12.30 (2.903)	1.253
PatchMatch [6]	83.91 (17.62)	1.210
GatedConv [48]	50.45 (16.46)	1.196
Non-Stationary [50]	103.7 (23.87)	1.186
PG-IM (ours)	21.48 (5.375)	1.229
Facade [39]		
Image Quilting [11]	13.50 (6.379)	1.217
PatchMatch [6]	81.35 (25.28)	1.219
GatedConv [48]	26.26 (133.9)	1.186
Non-Stationary [50]	133.9 (39.75)	1.199
PG-IM (ours)	14.40 (7.781)	1.218

Table 2: We compare PG-IM against off-the-shelf neural baselines for image inpainting on both datasets. Our method outperforms neural baselines with a remarkable margin across all metrics.

and PG-IM use single-image training for generating missing pixels, PG-IM uses symbolic programs as the guidance of the networks, enjoying both interpretability and better performance for complex structures. We also implement a variant of Non-Stationary, which keeps the neural architecture and training paradigm as the original version for texture synthesis, but use the same inpainting data as our method for inpainting. For a fair comparison, we train Non-Stationary and PG-IM with single sets of optimal hyperparameters on

all test images. For more results and analysis, please refer to the supplementary material.

4.3. Inpainting

We compare PG-IM with GatedConv, Image Quilting, and PatchMatch on the task of image inpainting. For quantitative evaluations, we use the NRP and Facade datasets, each of whose images gets randomly corrupted 100 times, giving us a total of around 15,000 test images.

Table 2 summarizes the quantitative scores of different methods. Following [25], we compare the L1 distance between the inpainted image and the original image, as well as Inception score (IS) of the inpainted image. For all the approaches, we hold out a test patch whose pixels are never seen by the networks during training, and use that patch for testing. Quantitatively, PG-IM outperforms the other learning-based methods by large margins across both datasets in both metrics. PG-IM recovers missing pixels a magnitude more faithful to the ground-truth images than Non-Stationary in the L1 sense. It also has a small variance across different images and input masks. For comparisons with non-learning-based methods, although Image Quilting achieves the best L1 score, it tends to break structures in the images, such as lines and grids (see Figure 5 for such examples). Note that the reason why PatchMatch has worse L1 scores is that it also modifies pixels around the holes to achieve better image-level consistency. In contrast, the other methods including PG-IM only inpaint holes and modify

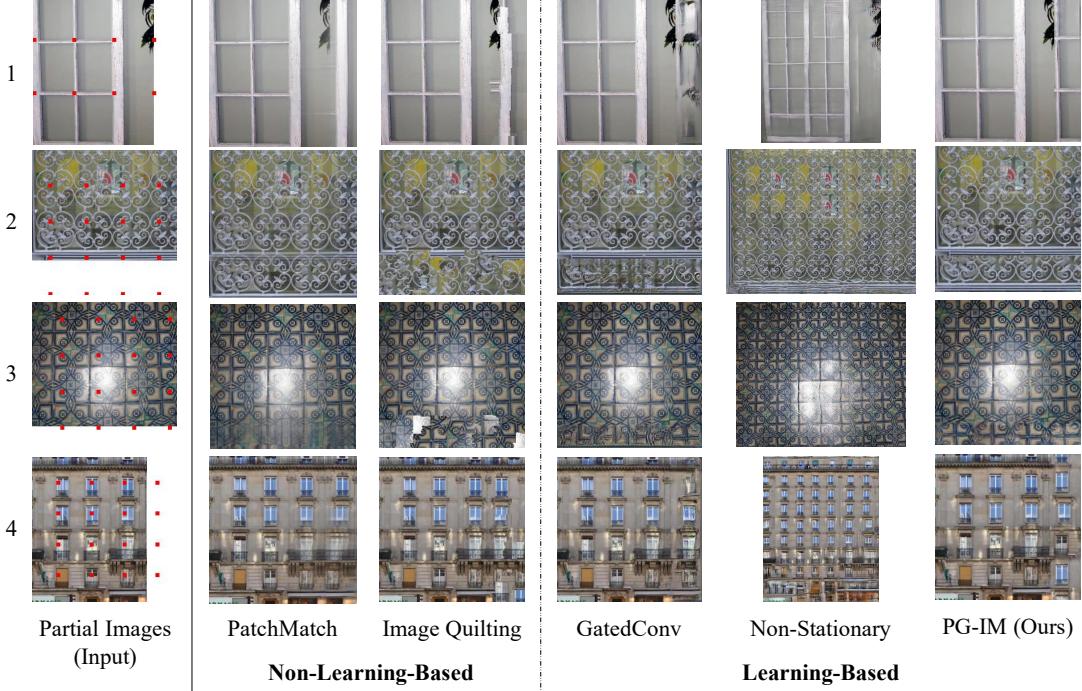


Figure 6: Extrapolation results by PG-IM and the baselines. The white pixels in the leftmost column indicate the pixels to be extrapolated. PG-IM generates realistic images while preserving global regularity. In contrast, GatedConv fails to capture the regularity; Non-Stationary does not preserve the original image contents; PatchMatch tends to generate blurry images in smoothing the transition; Image Quilting does not guarantee the global structure gets preserved.

nothing else in the images.

Qualitative results for inpainting are presented in Figure 5. Overall, our approach is able to preserve the “objects” in the test images even if the objects are completely missing, while other learning-based approaches either miss the intricate structures (Non-Stationary on Images 1 and 2), or produce irrelevant patches (learned from largely diverse image datasets) that break the global structure of this particular image (e.g., GatedConv on Image 2). Note how the image patches inpainted by our approach is realistic and meanwhile quite different from the ground-truth patches (compare our inpainting with the ground-truth Image 4). For the non-learning-based approaches, the baselines suffer from blurry outputs and sometimes produce inconsistent connections to the original image on boundaries. Moreover, as we demonstrate in Figure 7, unlike our approach that combines high-level symbolic reasoning and lower-level pixel manipulations, PatchMatch fails to manipulate the pixels in an attribute-aware fashion.

Runtime-wise, learning-methods including PG-IM, once trained, inpaint an image in a forward pass (around 100ms on GPUs), whereas non-learning-based approaches take around 15 minutes to inpaint one image.

4.4. Extrapolation

Figure 6 shows the extrapolation results by PG-IM and the baselines. With the program description of the images,

PG-IM naturally knows *where* to extrapolate to, e.g., by incrementing the for-loop range. This contrasts with the baselines that either require the user to specify which pixels to extrapolate (PatchMatch, Image Quilting, and GatedConv), or simply extrapolate to every possible direction (Non-Stationary). Knowing where to extrapolate is particularly crucial for images where the objects do not scatter all over. Take the pieces of cereal in Figure 1B as an example. PG-IM reasons about the global structure that the pieces of cereal form, decides where to extrapolate to by relaxing its program conditions, and finally extrapolates a new row.

As PatchMatch greedily “copies from” patches with the most similar context, certain pixels may come from different patches, therefore producing blurry extrapolation results (Images 1, 3, and 4). Learning from large-scale image datasets, GatedConv fails to capture the repeated patterns specific to each individual image, thus generating patterns that do not connect to the image boundary consistently (Images 2 and 3). Non-Stationary treats the entire image as consisting of only patterns of interest and expands the texture along all four directions; artifacts show up when the image contains more than the texture (bottom of Image 4). Also interesting is that Non-Stationary can be viewed as a super-resolution algorithm, in the sense that it is interpolating among the replicated objects. As the rightmost column shows, during extrapolation, PG-IM produces realistic and sharp patches (Image 1), preserves the images’ global regularity, and con-

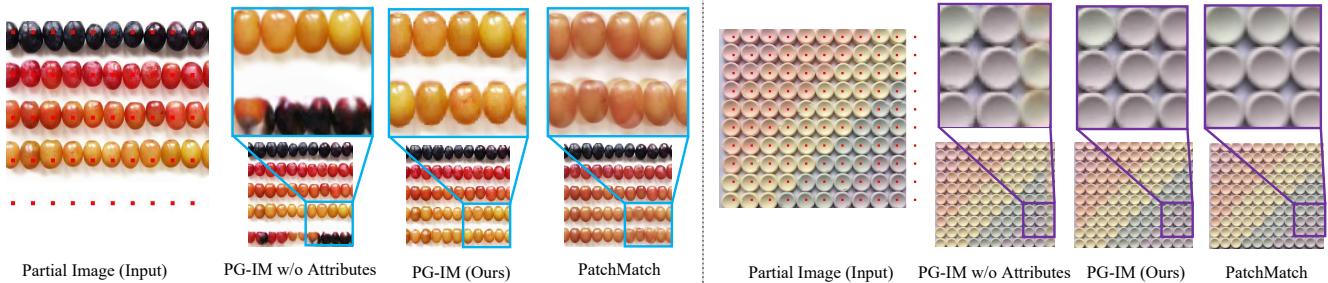


Figure 7: PG-IM can reason about the attribute regularity of images, which supports object appearance-aware image extrapolation. PG-IM w/o Attributes denotes a variant of PG-IM that does not include attributes. See the main text for detailed analysis and comparison.

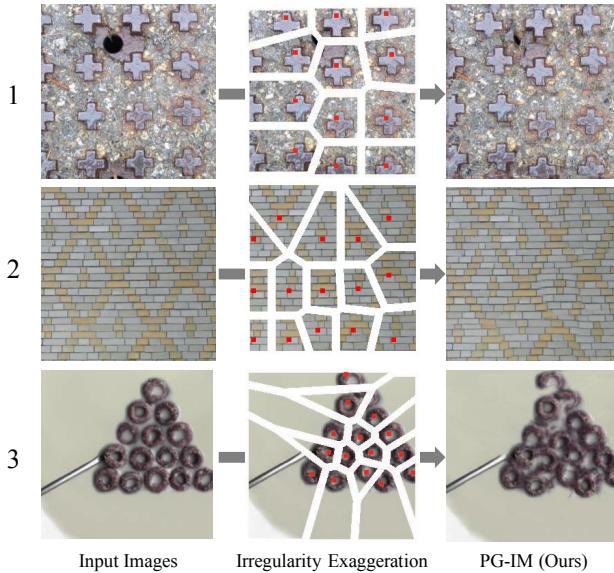


Figure 8: PG-IM enables automated and semantic-aware irregularity exaggeration. By comparing the centroids of the detected objects and the ones reconstructed by the program, we can measure and exaggerate the structural irregularity of input images.

ncts consistently to the image boundary (Images 2-4).

4.5. Image Regularity Editing

With a program describing the image’s *ideal* global regularity, PG-IM is able to exaggerate imperfections in the global regularity by magnifying the discrepancy between what the program depicts and the detected object centroids. A similar task has been discussed by [9]. In Figure 8, we magnify the displacement vectors between the program-provided and detected centroids by two, and shift the Voronoi cells together with their respective centroids, leaving missing values among the cells. An NPN then fills in the gaps by recurrent inpainting.

4.6. Attribute Regularity

Beyond using for-loops and if-conditions to capture the global regularity of objects, PG-IM can also reason about the regularity of object appearance variations (i.e., the *attribute* regularity). Our model automatically clusters objects into groups. Beyond knowing where to extrapolate to, with the

attribute regularity described by the program, our NPNs generate new pixels from only patches of the correct attributes.

Figure 7 illustrates this idea. We show the image extrapolation results on images with attribute regularities, and compare PG-IM with a variant that does not consider object attributes, as well as a strong baseline: PatchMatch. Without explicit modeling of object attributes, the color of the new objects generated by PG-IM w/o Attributes fails to preserve the global attribute regularity. Meanwhile, due to the existence of objects with similar colors, PatchMatch mixes up two different colors, resulting in blurry output patches (Figure 7L) or extrapolation results that break the global attribute regularity (the middle object in the top row of Figure 7R’s zoom-in windows should be purple, not green).

5. Discussion

This paper presents a neuro-symbolic approach to describing and manipulating natural images with repeated patterns. It combines the power of program induction—as symbolic tools for describing repetition, symmetry, and attributes—and deep neural networks—as powerful image generative models. PG-IM support various tasks: image inpainting, extrapolation, and regularity editing.

Our results also suggest multiple future directions. First, the variations in object appearance are currently handled as discrete properties. We leave the interpretation of attributes that have continuous values, such as the color spectrum in Figure 7, as future work. Second, given only a facade image containing a number of windows, humans can extrapolate the image by adding doors at the bottom and roof at the top. Combining regularity inference and data-driven approaches is a meaningful direction. Finally, the representational power of PG-IM is limited by the DSL. PG-IM currently does not generalize to unseen patterns, such as rotational patterns. Future works may consider a more flexible DSL, or even discovering new patterns from data.

Acknowledgements. We thank Michal Irani for helpful discussions and suggestions. This work is supported by the Center for Brains, Minds and Machines (NSF #1231216), NSF #1447476, ONR MURI N00014-16-1-2007, IBM Research, and Facebook.

References

- [1] Miika Aittala and Frédo Durand. Burst image deblurring using permutation invariant convolutional neural networks. In *ECCV*, 2018. 5
- [2] Michael Ashikhmin. Synthesizing natural textures. In *I3D*, 2001. 2
- [3] Yuval Bahat, Netalee Efrat, and Michal Irani. Non-uniform blind deblurring by reblurring. In *ICCV*, 2017. 2
- [4] Yuval Bahat and Michal Irani. Blind dehazing using internal patch recurrence. In *ICCP*, 2016. 2
- [5] Coloma Ballester, Marcelo Bertalmio, Vicent Caselles, Guillermo Sapiro, and Joan Verdera. Filling-in by joint interpolation of vector fields and gray levels. *IEEE TIP*, 10(8):1200–1211, 2001. 2
- [6] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan Goldman. Patchmatch: a randomized correspondence algorithm for structural image editing. *ACM TOG*, 28(3):24, 2009. 2, 5, 6
- [7] Tony Beltramelli. Pix2code: Generating code from a graphical user interface screenshot. In *ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS, 2018. 2
- [8] Urs Bergmann, Nikolay Jetchev, and Roland Vollgraf. Learning texture manifolds with the periodic spatial gan. In *ICML*, 2017. 2
- [9] Tali Dekel, Tomer Michaeli, Michal Irani, and William T Freeman. Revealing and modifying non-local variations in a single image. *ACM TOG*, 34(6):227, 2015. 8
- [10] Yuntian Deng, Anssi Kanervisto, Jeffrey Ling, and Alexander M Rush. Image-to-markup generation with coarse-to-fine attention. In *ICML*, 2017. 2
- [11] Alexei A Efros and William T Freeman. Image quilting for texture synthesis and transfer. In *CGIT*, 2001. 2, 5, 6
- [12] Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. Learning to infer graphics programs from hand-drawn images. In *NeurIPS*, 2018. 2
- [13] Gilad Freedman and Raanan Fattal. Image and video upscaling from local self-examples. *ACM TOG*, 30(2):12, 2011. 2
- [14] Yossi Gandelsman, Assaf Shocher, and Michal Irani. “Double-DIP”: Unsupervised image decomposition via coupled deep-image-priors. In *CVPR*, 2019. 2
- [15] Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, S. M. Ali Eslami, and Oriol Vinyals. Synthesizing programs for images using reinforced adversarial learning. In *ICML*, 2018. 2
- [16] Daniel Glasner, Shai Bagon, and Michal Irani. Super-resolution from a single image. In *ICCV*, 2009. 2
- [17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1
- [18] Qixing Huang, Hai Wang, and Vladlen Koltun. Single-view reconstruction via joint analysis of image and shape collections. *ACM TOG*, 34(4):87, 2015. 2
- [19] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Globally and locally consistent image completion. *ACM TOG*, 36(4):107, 2017. 2
- [20] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017. 5
- [21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012. 3
- [22] Louis Lettry, Michal Perdoch, Kenneth Vanhoey, and Luc Van Gool. Repeated pattern detection using cnn activations. In *WACV*, 2017. 2, 3, 5, 6
- [23] Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao Zhang, and Leonidas Guibas. Grass: Generative recursive autoencoders for shape structures. In *SIGGRAPH*, 2017. 2
- [24] Manyi Li, Akshay Gadi Patil, Kai Xu, Siddhartha Chaudhuri, Owais Khan, Ariel Shamir, Changhe Tu, Baoquan Chen, Daniel Cohen-Or, and Hao Zhang. Grains: Generative recursive autoencoders for indoor scenes. *ACM TOG*, 38(2):12:1–12:16, 2019. 2
- [25] Guilin Liu, Fitzsum A. Reda, Kevin J. Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. Image inpainting for irregular holes using partial convolutions. In *ECCV*, 2018. 2, 6
- [26] Yunchao Liu, Zheng Wu, Daniel Ritchie, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. Learning to describe scenes with programs. In *ICLR*, 2019. 2
- [27] Cewu Lu, Ranjay Krishna, Michael Bernstein, and Li Fei-Fei. Visual relationship detection with language priors. In *ECCV*, 2016. 1
- [28] Tomer Michaeli and Michal Irani. Blind deblurring using internal patch recurrence. In *ECCV*, 2014. 2
- [29] Kamyar Nazeri, Eric Ng, Tony Joseph, Faisal Qureshi, and Mehran Ebrahimi. Edgeconnect: Generative image inpainting with adversarial edge learning. *arXiv:1901.00212*, 2019. 2
- [30] Chengjie Niu, Jun Li, and Kai Xu. Im2Struct: Recovering 3D Shape Structure from a Single RGB Image. In *CVPR*, 2018. 2
- [31] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *CVPR*, 2016. 2
- [32] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*, 2017. 5
- [33] Irvin Rock and Stephen Palmer. The legacy of gestalt psychology. *Sci. Amer.*, 263(6):84–91, 1990. 1
- [34] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015. 4
- [35] Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. Singan: Learning a generative model from a single natural image. In *ICCV*, 2019. 2
- [36] Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. Csgnet: Neural shape parser for constructive solid geometry. In *CVPR*, 2018. 2
- [37] Assaf Shocher, Shai Bagon, Phillip Isola, and Michal Irani. Ingan: Capturing and remapping the “dna” of a natural image. In *ICCV*, 2019. 2
- [38] Assaf Shocher, Nadav Cohen, and Michal Irani. zero-shot super-resolution using deep internal learning. In *CVPR*, 2018. 2

- [39] Olivier Teboul, Loic Simon, Panagiotis Koutsourakis, and Nikos Paragios. Segmentation of building facades using procedural shape priors. In *CVPR*, 2010. [2](#), [3](#), [5](#), [6](#)
- [40] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. In *CVPR*, 2018. [2](#)
- [41] Yanzhen Wang, Kai Xu, Jun Li, Hao Zhang, Ariel Shamir, Ligang Liu, Zhiqian Cheng, and Yueshan Xiong. Symmetry hierarchy of man-made objects. *CGF*, 30(2):287–296, 2011. [2](#)
- [42] Junyuan Xie, Linli Xu, and Enhong Chen. Image denoising and inpainting with deep neural networks. In *NeurIPS*, 2012. [2](#)
- [43] Wei Xiong, Zhe Lin, Jimei Yang, Xin Lu, Connelly Barnes, and Jiebo Luo. Foreground-aware image inpainting. In *CVPR*, 2019. [2](#)
- [44] Zhaoyi Yan, Xiaoming Li, Mu Li, Wangmeng Zuo, and Shiguang Shan. Shift-net: Image inpainting via deep feature rearrangement. In *ECCV*, 2018. [2](#)
- [45] Chao Yang, Xin Lu, Zhe Lin, Eli Shechtman, Oliver Wang, and Hao Li. High-resolution image inpainting using multi-scale neural patch synthesis. In *CVPR*, 2017. [2](#)
- [46] Halley Young, Osbert Bastani, and Mayur Naik. Learning neurosymbolic generative models via program synthesis. In *ICML*, 2019. [2](#)
- [47] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Generative image inpainting with contextual attention. In *CVPR*, 2018. [2](#)
- [48] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Free-form image inpainting with gated convolution. In *ICCV*, 2019. [2](#), [5](#), [6](#)
- [49] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *IEEE TPAMI*, 40(6):1452–1464, 2017. [5](#)
- [50] Yang Zhou, Zhen Zhu, Xiang Bai, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Non-stationary texture synthesis by adversarial expansion. *SIGGRAPH*, 37(4), 2018. [2](#), [5](#), [6](#)
- [51] Maria Zontak and Michal Irani. Internal statistics of a single natural image. In *CVPR*, 2011. [2](#)

Supplementary Material: Program-Guided Image Manipulators

The supplementary material is organized as follows. In Section A, we first discuss the detailed implementation of PG-IM. We then, in Section B, provide ablation studies evaluating the effects of PG-IM’s different modules or designs. Finally, we show more results produced by PG-IM in Section C, and also quantitatively compare them with results by the baselines.

A. Implementation Details

We discuss the implementation details necessary to reproduce our results. Our code and data are available on the project page: <http://pgim.csail.mit.edu/>.

A.1. Repeated Object Detection

We replicated the algorithm developed by Lettry et al. [4]. The algorithm works on nearly-regular images and detects the repeated objects within the image. The model builds on the assumption that features learned from ImageNet [1] encode visual concepts at various levels: from edges to patches to high-level semantic objects. The algorithm takes as input the feature map from a pretrained AlexNet [3] and outputs a set of 2D coordinates representing the centroids of repeated objects. In this section, we explain the algorithm as six-step procedure.

- 1) For each input feature map \mathbf{f} , it extracts all activated neurons at different locations. Denote the neuron at the 2D position (i, j) being activated by $act(\mathbf{f}_{i,j}) = 1$, which is given by

$$act(\mathbf{f}_{i,j}) = \mathbf{1} \left[\mathbf{f}_{i,j} == \max_{(u,v), |u-i| \leq K, |v-j| \leq K} \mathbf{f}_{u,v} \right],$$

where $\mathbf{1}[\cdot]$ is the indicator function, and K is a hyperparameter for “kernel size.” Feature maps at different layers of the AlexNet use different kernel sizes.

- 2) All activated 2D neurons are collected. For each pair of activated neurons from the same feature map, a displacement vector is computed. For simplicity, we use $d^{i,j} = (d_x^{i,j}, d_y^{i,j})$ to index the j -th displacement vector from the i -th feature map. A majority voting is performed over all displacement vectors. The output of this step is an image-level displacement vector $d^* = (d_x^*, d_y^*)$ between centroids of different objects

$$d_x^* = \arg \max_x \sum_{i,j} \exp(-\|d_x^{i,j} - x\|_2^2) \quad \text{and} \quad d_y^* = \arg \max_y \sum_{i,j} \exp(-\|d_y^{i,j} - y\|_2^2).$$

- 3) Based on the image-level displacement vector d^* , all displacement vectors are filtered. A vector is selected if

$$\|d^{i,j} - d^*\|_2 \leq 3 \times \alpha,$$

where α is a scalar hyperparameter of radius.

- 4) Each feature map is weighted by the number of selected displacement vectors from this feature map. Only highly weighted feature maps are selected.

- 5) For each 2D coordinate (x, y) , we define a helper function

$$\mathcal{M}(x, y) \triangleq (x \bmod d_x^*, y \bmod d_y^*).$$

An offset o^* is then determined by optimizing the following objective:

$$o^* = \arg \min_o \sum_{i,j} \left(w_{i,j} \|M(d^{i,j} - o) - \frac{d^*}{2}\| \right),$$

where

$$w_{i,j} = \frac{1}{K_i + \phi} \cdot \exp \left(-\frac{\|d^{i,j} - d^*\|_2^2}{2\alpha^2} \right).$$

K_i is the number of displacement vectors of feature map i . ϕ and α are hyperparameters. In our implementation, we use gradient descent to find the optimal o^* .

- 6) For each object region detected, the coordinates for all activated neurons within the region vote for the centroid. This step allows distortions of objects that may cause irregular lattices.

A.2. Adaptive Network Depth for PatchGAN

In order to encourage creative and sharp patch generation, we employ a discriminator loss provided by AdaPatchGAN, our adaptive variant of PatchGAN [2], in addition to the L1 loss. In the original PatchGAN, the classifier architecture is fixed with a receptive field of a certain size. However, in images of our interest, the scale of texture varies violently across different images: the repeated pattern is sometimes a small entity but appearing many times, while other times the pattern is a larger object appearing only a few times. Given a single image, since its inferred program description provides a way of dividing it into patches, AdaPatchGAN computes a rough patch size and automatically adds convolutional layers until its receptive field is large enough to cover this patch size. In practice, we observe that our AdaPatchGAN enables the generator to produce more realistic patches than the original PatchGAN.

A.3. Extrapolation as Recurrent Inpainting

As mentioned in the main text, a single neural painting network (NPN) trained on the task of image inpainting is able to perform additional manipulation tasks including extrapolation without any finetuning. The straightforward way of doing this is treating the pixels to extrapolate as missing, and just inpainting them in one go. This approach of treating extrapolation as “feedforward inpainting” of multiple objects, however, results in distorted objects that fail to look realistic, as shown in an ablation study below. This is because NPNs are better at inpainting a localized patch (what they are trained to do as in inpainting) than inpainting a region spanning multiple objects.

Bearing this key observation in mind, our NPNs cast extrapolation into a task of *recurrent* inpainting, where it inpaints one object at a time, conditioned on the context provided by the previous inpainting. Note that this is only possible because the NPNs are guided by programs of images and therefore know how many objects should be extrapolated as well as where each of them should appear. Implementation-wise, this is achieved by repeatedly filling the current empty patch with the network output, and running network inference on the next empty patch. By this design, an empty patch gets inpainted so that it connects seamlessly to both the previously inpainted patches and the original image boundary.

A.4. Detailed Network Specifications and Training Parameters

We append to the end of this document a commented printout of the generator architecture, where channel numbers of the feature maps, kernel sizes, strides, padding, and more can be found. The training of PG-IM generally follows the procedure described in pix2pix [2], except that we do not decay our learning rate after 100 epochs like pix2pix does. Each of our epochs contains 1,000 training samples (i.e., random crops at different scales of the same image of interest), and the networks usually converge within 150 epochs.

B. Ablation Studies

We report three ablations studies to provide intuitions as to why PG-IM works. Key to our approach is the use of program-like descriptions for images in guiding neural painting networks (NPNs) to perform pixel manipulations. The first study demonstrates why such descriptions facilitate the image manipulation tasks. Next, we justify a crucial design for image extrapolation: extrapolation should be cast into the task of *recurrent* inpainting for NPNs, rather than “feedforward inpainting.” Supplemental to Figure 7 of the main paper, the third study emphasizes the advantages of considering high-level attributes in the lower-level image manipulation tasks.

B.1. With vs. Without Source Patches

Our program-like description for an image informs NPNs of where the pattern of interest (repeatedly) appears in the image. NPNs can then align the available patterns to the corrupted or missing one in order to inpaint it (for simplicity, we discuss only image inpainting in this study, but the conclusion stands for the other tasks). These aligned image patches serve as “source patches,” from which NPNs smartly copy to fill in the missing pixels.

Without program-like descriptions, NPNs would not have access to these source patches, which in turn leads to inferior performance such as results with the correct background but missing objects. As Figure 1 shows, when programs (and hence source patches) are not available to the networks to smartly copy from, they fail to ensure the presence of the missing object in their inpainting (e.g., the cross in the left pane and the diamond in the right pane). In contrast, our NPNs exploit the source patches provided by programs and can inpaint the missing objects even if they are corrupted completely in the input. This result supports that PG-IM outperforms repetition-aware but “programless” CNNs.

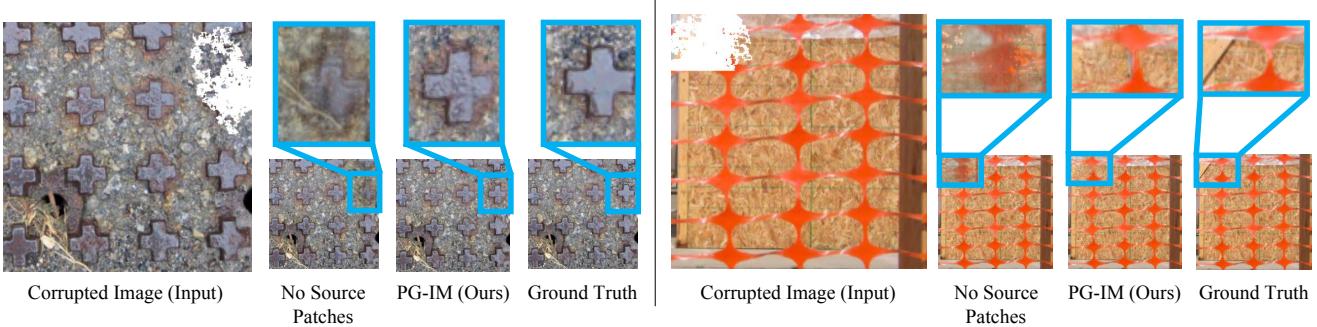


Figure 1: Inpainting with vs. without the source patches provided by program. For each pane, the leftmost figure is the corrupted input to inpaint; to the right are results inpainted with and without source patches as well as the ground truth. With programs describing where the object centers are, our NPNs align the intact objects to the corrupted object and exploit such alignment to inpaint the missing pixels. Without such source patches, the networks have difficulty ensuring the “objectness” of the inpainted patch.

B.2. With vs. Without Recurrent Inpainting

In this study, we show empirical evidence that NPNs produce better results when extrapolation is cast into the task of recurrent inpainting than feedforward inpainting. As Figure 2 shows, when the entire stripe gets extrapolated in one go (“extrapolation as feedforward inpainting”), the extrapolated objects tend to be distorted, since our NPNs are trained only on more localized inpainting. By treating extrapolation as *recurrent* inpainting, our NPNs are essentially doing, repeatedly, what they are trained on and hence generate more realistic-looking results.

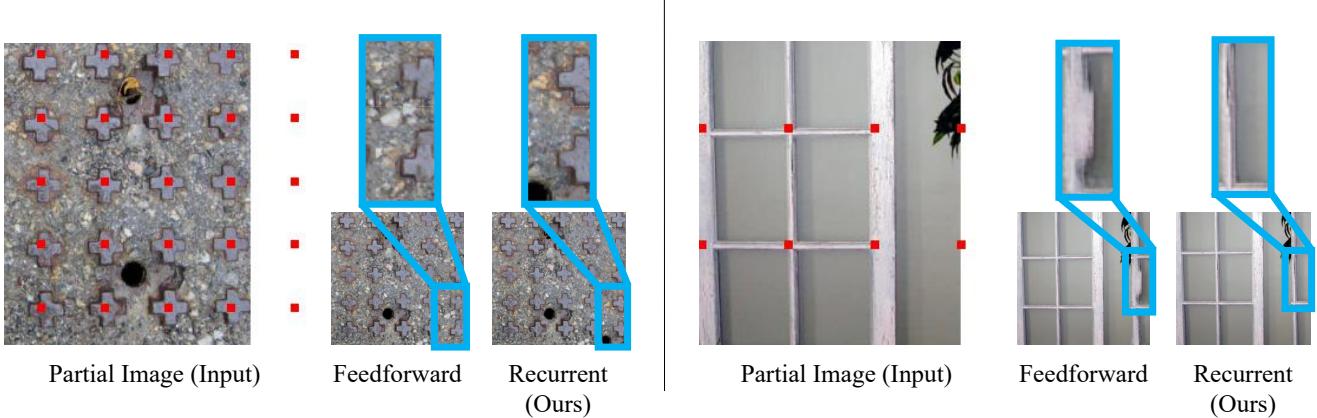


Figure 2: Extrapolation as feedforward vs. recurrent inpainting. For each pane, the leftmost figure is the partial input to be extrapolated; to the right are results extrapolated in one go and recurrently. Recall that the networks are trained only on inpainting one entity, so the networks tend to produce distorted objects if they are asked to inpaint the whole stripe spanning multiple entities. Instead, our NPNs treat extrapolation as recurrent inpainting, where they repeatedly solve the task they are good at and therefore produce better results.

B.3. With vs. Without Considering Attributes

In Figure 7 of the main paper, we show our program-like descriptions of images can naturally incorporate attributes, enabling the NPNs to perform attribute-aware image manipulation, e.g., by referencing only patches with the same attributes as the patch to inpaint. We supplement two more examples demonstrating why attribute-aware image manipulations are important, and how PG-IM achieves superior performance on such tasks.

As shown in Figure 3 left, the tomato to be inpainted is next to four orange tomatoes and one green tomato. Since convolutions are more of a local operation, one can expect a convolutional generator to produce a tomato with mixed colors of both orange and green, as is the case for PG-IM w/o Attributes. However, we, as humans, reason at a higher level over the entire image, coming to the conclusions that the tomato to inpaint on the left should be orange, and that the candy to inpaint on the right should be red. With similar capabilities, PG-IM correctly preserves the global regularity of attributes in its inpainting.

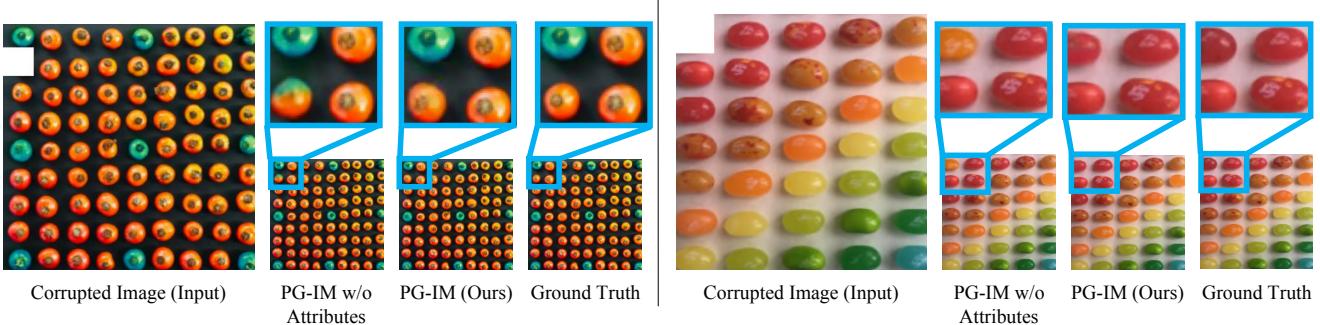


Figure 3: Inpainting with *vs.* without considering object attributes. For each pane, the leftmost figure is the corrupted input to be inpainted; to the right are results inpainted with and without considering attributes as well as the ground truth. The attribute-agnostic networks tend to produce patches with mixed colors that break the global regularity of attributes. PG-IM is able to respect such regularity by referencing only patches with the same attribute as the patch to inpaint.

C. More Experiments and Results

We supplement more results by PG-IM on the tasks of image inpainting, extrapolation, and regularity editing. For PatchMatch and Image Quilting, we search for one set of optimal hyperparameters and use that for the entire test set. Similarly, we train Non-Stationary [6] and PG-IM with single sets of optimal hyperparameters on each test image. For GatedConv [5], we use the trained model released by the authors.

C.1. Inpainting

Figure 4 shows how PG-IM compares in image inpainting with the baselines on another six test images. PG-IM is able to ensure the presence of the repeated object in its inpainting (e.g., the cross structure in Image 1 and the yellow tag in Image 5), while the baselines tend to have the object missing (e.g., PatchMatch completely misses the yellow tag in Image 5) or incomplete (e.g., the pig in Image 3) due to lack of the concept of objects.

C.2. Extrapolation

We supplement four more examples of image extrapolation by PG-IM and how they compare with results by the baselines. As Figure 5 shows, extrapolation by PG-IM is sharp, respects the global regularity (e.g., the “pig array” in Image 3), and seamlessly connects to the original images. In contrast, PatchMatch tends to blur over the image boundaries for smooth transition to the extrapolated contents (e.g., the blurriness over the boundary in Image 2). Image Quilting and GatedConv tend to break the global regularity (e.g. Images 2 and 4). Finally, Non-Stationary should be considered as a super-resolution method instead of an extrapolation one, because it essentially replicates the patterns on a larger canvas and interpolates in between; notice how it does *not* extrapolate beyond the texture boundaries in Images 2 and 4.

C.3. Regularity Editing

We supplement four more examples of image regularity editing by PG-IM in Figure 6. Although PG-IM has been trained only on inpainting, with the program descriptions of images, it is naturally capable of exaggerating the irregularity in the images’ global structures.

C.4. Failure Cases

Figure 7 demonstrates a failure case for inpainting. In this challenging case, the transmission and reflection effects make the repetitive patterns less homogeneous, leading to non-photorealistic inpainting.

C.5. Baseline Finetuning

PG-IM learns from a single input image by exploiting the repetition structure, whereas other learning-based methods (GatedConv and PartialConv) learn from a large dataset (Places365). We supplement extra experiments, where we finetune the GatedConv model (pre-trained on Places365) on 50 Facade images and evaluate it on the held-out images (Figure 8). Learning from similar facade images does help the model produce more photorealistic results (Inception score: 1.187 → 1.191). However, the model still fails on structured objects such as windows, which can be well handled by our patch-based, program-guided NPN (PG-IM’s Inception score is 1.210).

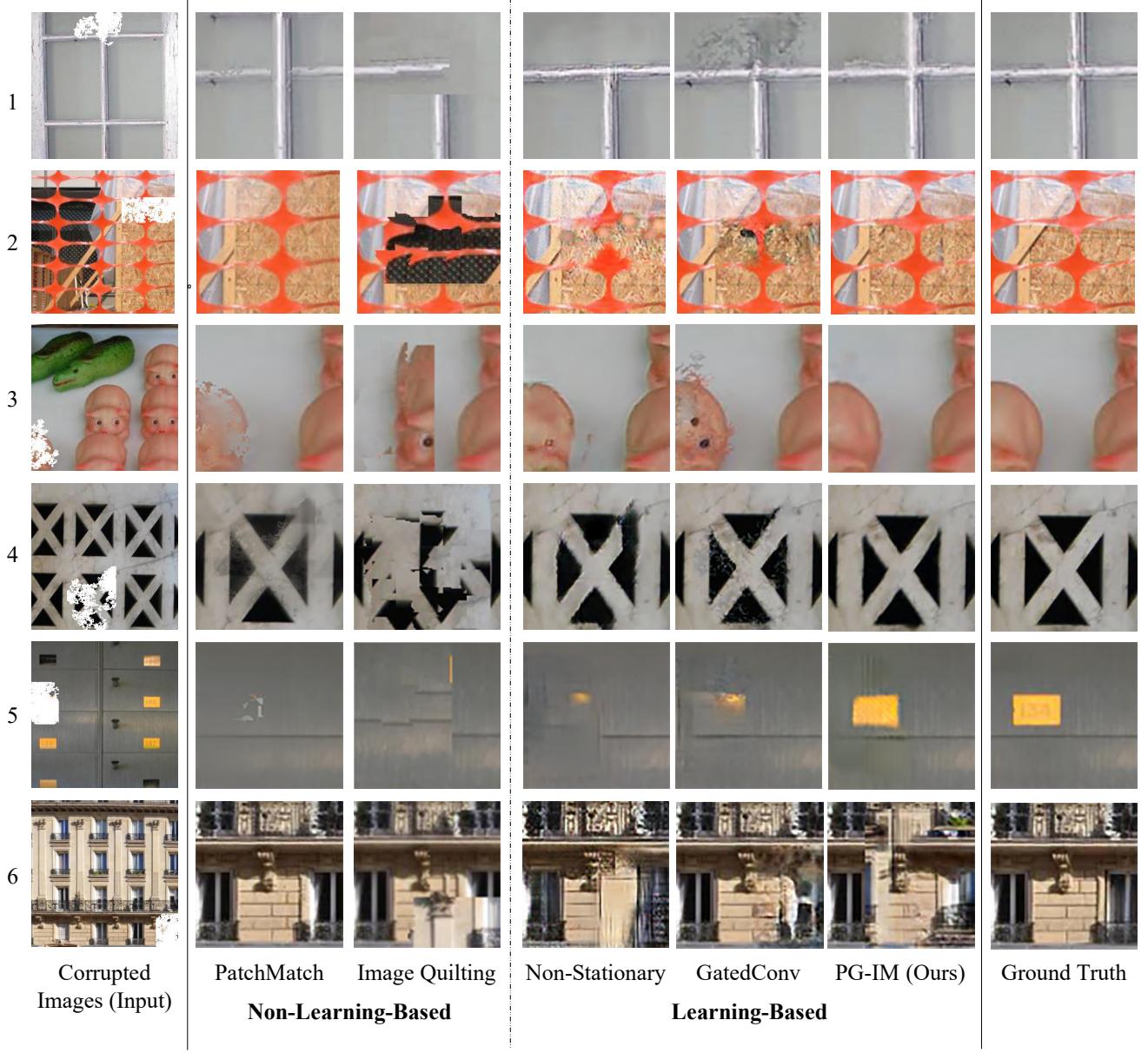


Figure 4: More inpainting results by PG-IM and the baselines. The white pixels in the leftmost column are missing pixels to inpaint; the rightmost column shows the ground-truth patches. For easier comparison, we show only the close-up views of the inpainted region (and its proximity for some context). Aware of the images’ global regularity, PG-IM inpaints the missing objects that maintain this regularity. Note this is not necessarily the case for other methods: baselines miss out the pig entity of Image 3 and the yellow tag of Image 5. Although PatchMatch gets the cross structure in Image 1, its result is less realistic than ours. Another advantage of PG-IM over the baselines is the ability to generate pixels that seamlessly connect to the original image contents (compare the results for Images 1, 2, and 4).

References

- [1] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. [1](#)
- [2] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017. [2](#)
- [3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012. [1](#)

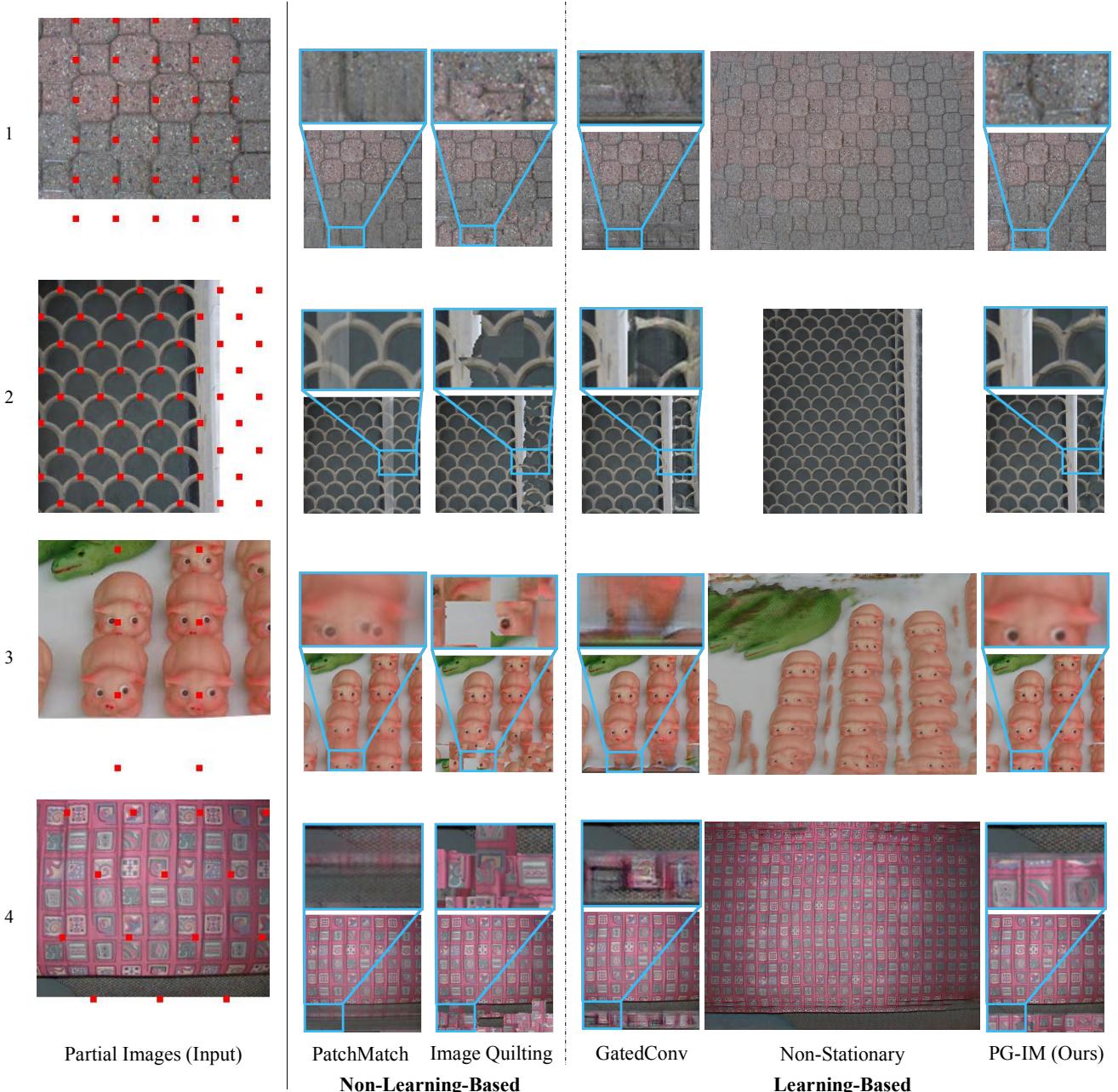


Figure 5: Extrapolation results by PG-IM and the baselines. The white pixels in the leftmost column indicate the pixels to be extrapolated, and the red dots are object centers given by the program descriptions. With such program descriptions, PG-IM knows where to extrapolate to automatically, whereas other methods require the user to specify where to extrapolate to. PG-IM generates realistic images while preserving the global regularity. In contrast, GatedConv fails to capture the regularity; Non-Stationary does not preserve the original content of images; non-learning-based baselines sometimes generate blurry images because of the repetition of similar objects in the partial image.

- [4] Louis Lettry, Michal Perdoch, Kenneth Vanhoey, and Luc Van Gool. Repeated pattern detection using cnn activations. In *WACV*, 2017. [1](#)
- [5] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Free-form image inpainting with gated convolution. In *ICCV*, 2019. [4](#)
- [6] Yang Zhou, Zhen Zhu, Xiang Bai, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Non-stationary texture synthesis by adversarial expansion. *SIGGRAPH*, 37(4), 2018. [4](#)

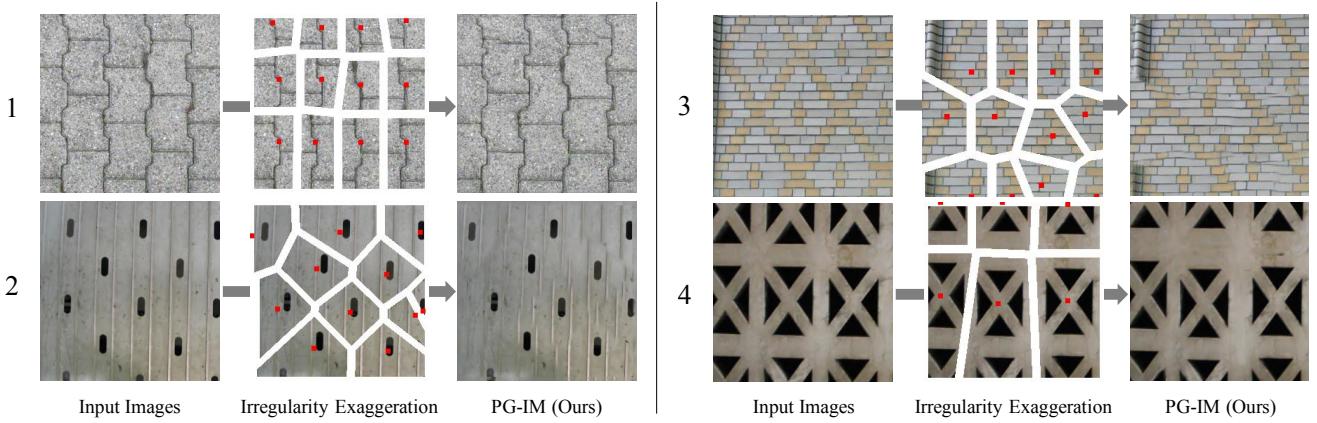


Figure 6: PG-IM enables automated and semantic-aware irregularity exaggeration. By comparing the centroids of the detected objects and the ones reconstructed by the program, we can measure and exaggerate the structural irregularity of input images. In these examples, we first multiply by 2 the displacement vectors between the object centroids provided by the programs and the detected object centroids, and then randomly flip the sign for each displacement vector. According to these new displacement vectors, we shift the patches and then let the NPN fill in the missing pixels.

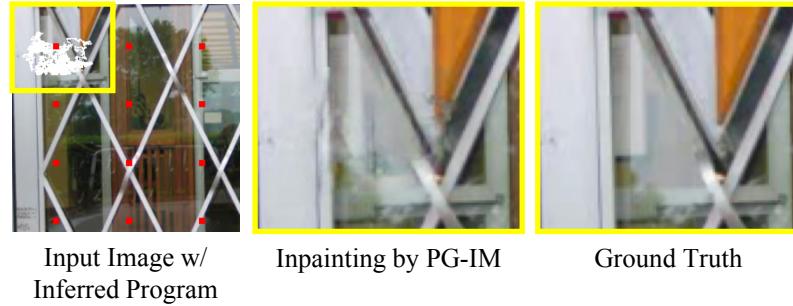


Figure 7: A failure case.

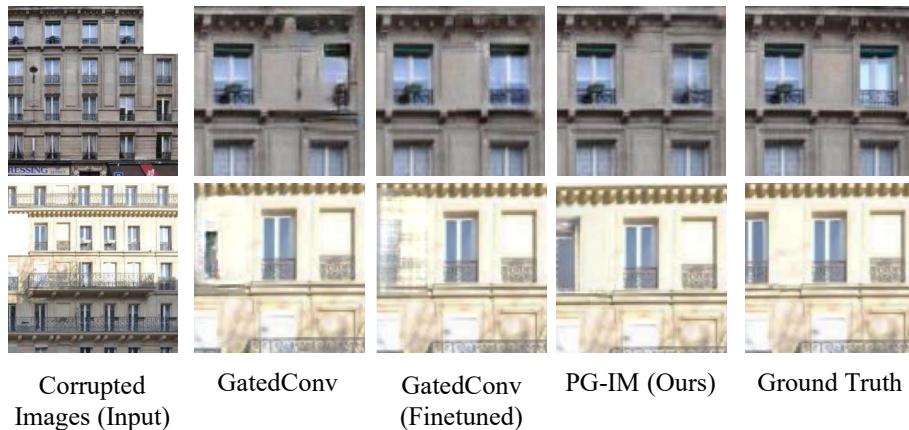


Figure 8: GatedConv model finetuned on the Facade dataset. Note that our PG-IM is only trained on a single input image.

Appendix: Detailed Generator Architecture

```
('1', ReppadConv2d(3, 64, 3, stride=1, padding=1)),
('1a', nn.ELU()),
# 2: maxpool across tracks
# 3: concat 2
('4:+pool', ReppadConv2d(128, 96, 1, stride=1)),
('4a', nn.ELU()),
('5', ReppadConv2d(96, 96, 4, stride=2, padding=1)),
('5a', nn.ELU()),
# 6: maxpool across tracks
# 7: concat 6
('8:+pool', ReppadConv2d(192, 128, 1, stride=1)),
('8a', nn.ELU()),
('9', ReppadConv2d(128, 128, 4, stride=2, padding=1)),
('9a', nn.ELU()),
# 10: maxpool across tracks
# 11: concat 10
('12:+pool', ReppadConv2d(256, 256, 1, stride=1)),
('12a', nn.ELU()),
('13', RepadConv2d(256, 256, 4, stride=2, padding=1)),
('13a', nn.ELU()),
# 14: maxpool across tracks
# 15: concat 14
('16:+pool', RepadConv2d(512, 384, 1, stride=1)),
('16a', nn.ELU()),
('17', RepadConv2d(384, 384, 4, stride=2, padding=1)),
('17a', nn.ELU()),
('18', RepadConv2d(384, 384, 3, stride=1, padding=1)),
('18a', nn.ELU()),
# -----
('19', nn.ConvTranspose2d(384, 384, 4, stride=2, padding=1)),
('19a', nn.ELU()),
# 20: maxpool across tracks
# 21: concat 16a, 20
('22:+pool+16a', RepadConv2d(1152, 384, 1, stride=1)),
('22a', nn.ELU()),
('23', RepadConv2d(384, 384, 3, stride=1, padding=1)),
('23a', nn.ELU()),
('24', nn.ConvTranspose2d(384, 256, 4, stride=2, padding=1)),
('24a', nn.ELU()),
# 25: maxpool across tracks
# 26: concat 12a, 25
('27:+pool+12a', RepadConv2d(768, 256, 1, stride=1)),
('27a', nn.ELU()),
('28', RepadConv2d(256, 256, 3, stride=1, padding=1)),
('28a', nn.ELU()),
('29', nn.ConvTranspose2d(256, 192, 4, stride=2, padding=1)),
('29a', nn.ELU()),
# 30: maxpool across tracks
# 31: concat 8a, 30
('32:+pool+8a', RepadConv2d(512, 192, 1, stride=1)),
('32a', nn.ELU()),
('33', RepadConv2d(192, 192, 3, stride=1, padding=1)),
('33a', nn.ELU()),
('34', nn.ConvTranspose2d(192, 96, 4, stride=2, padding=1)),
('34a', nn.ELU()),
# 35: maxpool across tracks
```

```
# 36: concat 1a, 35
('37:+pool+1a', ReppadConv2d(256, 96, 1, stride=1)),
('37a', nn.ELU()),
('38', ReppadConv2d(96, 96, 3, stride=1, padding=1)),
('38a', nn.ELU()),
# 39: maxpool across tracks
('40:pool', ReppadConv2d(96, 64, 3, stride=1, padding=1)),
('40a', nn.ELU()),
('41', ReppadConv2d(64, nch_out, 3, stride=1, padding=1)),
```