Learning Neurosymbolic Generative Models via Program Synthesis

Halley Young [1] **Osbert Bastani** [1] **Mayur Naik** [1]

Abstract

Generative models have become significantly more powerful in recent years. However, these models continue to have difficulty capturing global structure in data. For example, images of buildings typically contain spatial patterns such as windows repeating at regular intervals, but state-of-the-art models have difficulty generating these patterns. We propose to address this problem by incorporating programs representing global structure into generative models—e.g., a 2D for-loop may represent a repeating pattern of windows—along with a framework for learning these models by leveraging program synthesis to obtain training data. On both synthetic and real-world data, we demonstrate that our approach substantially outperforms state-of-the-art at both generating and completing images with global structure.

1. Introduction

There has been much interest recently in generative models, following the introduction of both variational autoencoders (VAEs) (Kingma & Welling, 2014) and generative adversarial networks (GANs) (Goodfellow et al., 2014). These models have successfully been applied to a range of tasks, including image generation (Radford et al., 2015), image completion (IIzuka et al., 2017), texture synthesis (Jetchev et al., 2017; Xian et al., 2018), sketch generation (Ha & Eck, 2017), and music generation (Dieleman et al., 2018).

Despite their successes, generative models still have difficulty capturing global structure. For example, consider the image completion task in Figure 1. The original image (left) is of a building, for which the global structure is a 2D repeating pattern of windows. Given a partial image (middle left), the goal is to predict the completion of the image. As can be seen, a state-of-the-art image completion algorithm has trouble reconstructing the original image (right) (IIzuka et al., 2017). Real-world data often contains such global structure, including repetitions, reflectional or rotational symmetry, or even more complex patterns.

In recent years, *program synthesis* (Solar-Lezama et al., 2006) has emerged as a promising approach to capturing patterns in data (Ellis et al., 2015; 2018; Valkov et al., 2018). The idea is that simple programs can capture global structure that evades state-of-the-art deep neural networks. A key benefit of using program synthesis is that we can design the space of programs to capture different kinds of structure—e.g., repeating patterns (Ellis et al., 2018), symmetries, or spatial structure (Deng et al., 2018)—depending on the application domain. The challenge is that for the most part, existing approaches have synthesized programs that operate directly over raw data. Since programs have difficulty operating over perceptual data, existing approaches have largely been limited to very simple data—e.g., detecting 2D repeating patterns of simple shapes (Ellis et al., 2018).

We propose to address these shortcomings by synthesizing programs that represent the underlying structure of high-dimensional data. In particular, we decompose programs into two parts: (i) a *sketch* $s \in S$ that represents the skeletal structure of the program (Solar-Lezama et al., 2006), with *holes* that are left unimplemented, and (ii) *components* $c \in C$ that can be used to fill these holes. We consider *perceptual components*—i.e., holes in the sketch are filled with raw perceptual data. For example, the program

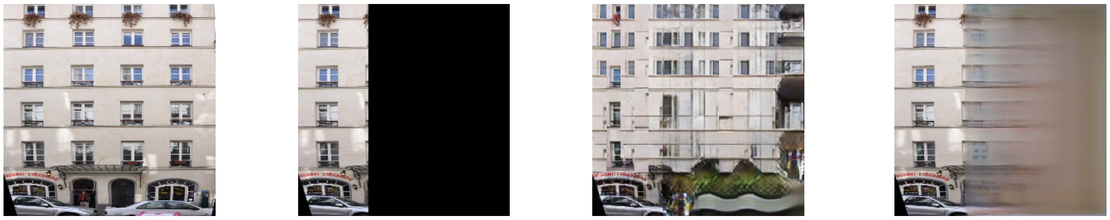
```
for i = 1..3
  for j = 1..4
    draw(i*2, j*3,       )
```

represents part of the structure in the original image x^* in Figure 1 (left). The code is the sketch, and the component is a sub-image from the given partial image. Together, we call such a program a *neurosymbolic program*.

Building on these ideas, we propose an approach called *Synthesis-guided Generative Models* (SGM) that combines neurosymbolic programs representing global structure with state-of-the-art deep generative models. By incorporating programmatic structure, SGM substantially improves the quality of these models. As can be seen, the completion produced using SGM (middle right of Figure 1) substantially outperforms state-of-the-art.

[1]University of Pennsylvania, USA. Correspondence to: Halley Young <halleyy@seas.upenn.edu>.

Proceedings of the 36^{th} *International Conference on Machine Learning*, Long Beach, California, PMLR 97, 2019.



original image x^* partial image x_{part} completion $\hat{x}$ (ours) completion $\hat{x}$ (baseline)

Figure 1. The task is to complete the partial image x_{part} (middle left) into an image that is close to the original image x^* (left). By incorporating programmatic structure, our approach (middle right) substantially outperforms state-of-the-art (IIzuka et al., 2017) (right).

SGM can be used for both generation and completion. The generation pipeline is shown in Figure 2. At a high level, SGM for generation operates in two phases:

- First, it generates a program that represents the global structure in the image to be generated. In particular, it generates a program $P = (s, c)$ representing the latent global structure in the image (left in Figure 2), where s is a sketch and c is a perceptual component.
- Second, our algorithm executes P to obtain a *structure rendering* x_{struct} representing the program as an image (middle of Figure 2). Then, our algorithm uses a deep generative model to complete x_{struct} into a full image (right of Figure 2). The structure in x_{struct} helps guide the deep generative model towards images that preserve the global structure.

The image-completion pipeline (see Figure 3) is similar.

Training these models end-to-end is challenging, since a priori, ground truth global structure is unavailable. To address this shortcoming, we leverage domain-specific program synthesis algorithms to produce examples of programs that represent global structure of the training data. In particular, we propose a synthesis algorithm tailored to the image domain, which extracts programs with nested for-loops that can represent multiple 2D repeating patterns in images. Then, we use these example programs as supervised training data.

Our programs can capture rich spatial structure in the training data. For example, in Figure 2, the program structure encodes a repeating structure of 0's and 2's on the whole image, and a separate repeating structure of 3's on the right-hand side of the image. Furthermore, in Figure 1, the generated image captures the idea that the repeating pattern of windows does not extend to the bottom portion of the image.

Contributions. We propose an architecture of generative models that incorporates programmatic structure, as well as an algorithm for training these models (Section 2). Our learning algorithm depends on a domain-specific program synthesis algorithm for extracting global structure from the training data; we propose such an algorithm for the image domain (Section 3). Finally, we evaluate our approach on synthetic data and on a real-world dataset of building facades (Tyleček & Šára, 2013), both on the task of generation from scratch and on generation from a partial image. We show that our approach substantially outperforms several state-of-the-art deep generative models (Section 4).

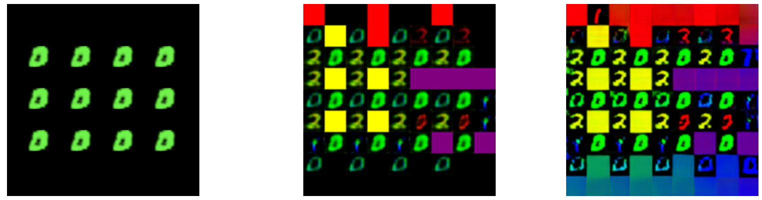
Related work. There has been growing interest in applying program synthesis to machine learning—e.g., for small data (Liang et al., 2010), interpretability (Wang & Rudin, 2015; Verma et al., 2018), safety (Bastani et al., 2018), and lifelong learning (Valkov et al., 2018). Most relevantly, there has been interest in using programs to capture structure that deep learning models have difficulty representing (Lake et al., 2015; Ellis et al., 2015; 2018; Pu et al., 2018). For instance, Ellis et al. (2015) proposes an unsupervised learning algorithm for capturing repeating patterns in simple line drawings; however, not only are their domains simple, but they can only handle a very small amount of noise. Similarly, Ellis et al. (2018) captures 2D repeating patterns of simple circles and polygons; however, rather than synthesizing programs with perceptual components, they learn a simple mapping from images to symbols as a preprocessing step. The closest work we are aware of is Valkov et al. (2018), which synthesizes programs with *neural components* (i.e., components implemented as neural networks); however, their application is to lifelong learning, not generation, and to learning with supervision (labels) rather than to unsupervised learning of structure.

There has been related work on synthesizing probabilistic programs (Hwang et al., 2011; Perov & Wood, 2014), including applications to learning structured ranking functions (Nori et al., 2015) and for learning design patterns (Talton et al., 2012). More recently, DeepProbLog (Manhaeve et al., 2018) has extended the probabilistic logic programming language ProbLog (De Raedt et al., 2007) to include learned neural components.

Additionally, there has been work extending neural module networks (Andreas et al., 2016) to generative models (Deng et al., 2018). These algorithms essentially learn a collection of neural components that can be composed together based on hierarchical structure. However, they require that the

```
for i = 1..3
  for j = 1..4
    draw(i*2 + 2, j*2 + 1,   )
        ...
```

sampled program P



single for loop rendering — structure rendering x_{struct} — generated image $\hat{x}$

Figure 2. SGM generation pipeline: (i) Sample a latent vector $z \sim p(z)$, and sample a program $P = (s, c) \sim p_\phi(s, c \mid z)$ (left: a single sampled for loop). (ii) Execute P to obtain a structure rendering (middle left: the rendering of the single for loop shown on the left, middle right: the structure rendering). (iii) Sample a completion $\hat{x} \sim p_\theta(x \mid s, c)$ of x_{struct} into a full image (right).

structure be available (albeit in natural language form) both for training the model and for generating new images.

Finally, there has been work incorporating spatial structure into models for generating textures (Jetchev et al., 2017); however, their work only handles a single infinite repeating 2D pattern. In contrast, we can capture a rich variety of spatial patterns parameterized by a space of programs—e.g., the image in Figure 1 generated by our approach contains different repeating patterns in different parts of the image.

2. Generative Models with Latent Structure

We describe our proposed architecture for generative models that incorporate programmatic structure. For most of this section, we focus on generation; we discuss how we adapt these techniques to image completion at the end. We illustrate our generation pipeline in Figure 2.

Let $p_{\theta,\phi}(x)$ be a distribution over a space $\mathcal{X}$ with unknown parameters θ, ϕ that we want to estimate. We study the setting where x is generated based on some latent structure, which consists of a *program sketch* $s \in \mathcal{S}$ and a *perceptual component* $c \in \mathcal{C}$, and where the structure is in turn generated conditioned on a latent vector $z \in \mathcal{Z}$—i.e.,

$$p_{\theta,\phi}(x) = \int_{\mathcal{Z}} \int_{\mathcal{C}} \sum_{s \in \mathcal{S}} p_\theta(x \mid s, c) p_\phi(s, c \mid z) p(z) dc dz.$$

Figure 2 shows an example of a sampled program $P = (s, c) \sim p_\phi(s, c \mid z)$ (left) and a sampled completion $\hat{x} \sim p_\theta(x \mid s, c)$ (right). To sample a completion, our model executes P to obtain a *structure rendering* $x_{\text{struct}} = \text{eval}(P)$ (middle), and then uses $p_\theta(x \mid s, c) = p_\theta(x \mid x_{\text{struct}})$.

We now describe our algorithm for learning the parameters θ, ϕ of $p_{\theta,\phi}$, followed by a description of our choices of architecture for $p_\phi(s, c \mid z)$ and $p_\theta(x \mid s, c)$.

Learning algorithm. Given training data $\{x^{(i)}\}_{i=1}^n \subseteq \mathcal{X}$, where $x^{(i)} \sim p_{\theta,\phi}(x)$, the maximum likelihood estimate is

$$\theta^*_{\text{MLE}}, \phi^*_{\text{MLE}} = \arg\max_{\theta,\phi} \sum_{i=1}^n \log p_{\theta,\phi}(x^{(i)}).$$

Since $\log p_{\theta,\phi}(x)$ is intractable to optimize, we use an approach based on the variational autoencoder (VAE). In particular, we use a variational distribution

$$q_{\tilde{\phi}}(s, c, z \mid x) = q_{\tilde{\phi}}(z \mid s, c) q(s, c \mid x),$$

which has parameters $\tilde{\phi}$. Then, we optimize $\tilde{\phi}$ while simultaneously optimizing θ, ϕ. Using $q_{\tilde{\phi}}(s, c, z \mid x)$, the *evidence lower bound* on the log-likelihood is

$$\begin{aligned} \log p_{\theta,\phi}(x) \geq\ & \mathbb{E}_{q(s,c,z|x)}[\log p_\theta(x \mid s, c)] \\ & - D_{\text{KL}}(q(s, c, z \mid x) \parallel p_\phi(s, c \mid z) p(z)) \\ =\ & \mathbb{E}_{q(s,c|x)}[\log p_\theta(x \mid s, c)] \\ & + \mathbb{E}_{q(s,c|x), q_{\tilde{\phi}}(z|s,c)}[\log p_\phi(s, c \mid z)] \\ & - \mathbb{E}_{q(s,c|x)}[D_{\text{KL}}(q_{\tilde{\phi}}(z \mid s, c) \parallel p(z))] \\ & - H(q(s, c \mid x)), \end{aligned} \tag{1}$$

where D_{KL} is the KL divergence and H is information entropy. Thus, we can approximate θ^*, ϕ^* by optimizing the lower bound (1) instead of $\log p_{\theta,\phi}(x)$. However, (1) remains intractable since we are integrating over all program sketches $s \in \mathcal{S}$ and perceptual components $c \in \mathcal{C}$. As we describe next, our approach is to synthesize a single point estimate $s_x \in \mathcal{S}$ and $c_x \in \mathcal{C}$ for each $x \in \mathcal{X}$.

Synthesizing structure. For a given $x \in \mathcal{X}$, we use *program synthesis* to infer a *single* likely choice $s_x \in \mathcal{S}$ and $c_x \in \mathcal{C}$ of the latent structure. The program synthesis algorithm must be tailored to a specific domain; we propose an algorithm for inferring for-loop structure in images in Section 3. Then, we use these point estimates in place of the integrals over $\mathcal{S}$ and $\mathcal{C}$—i.e., we assume that

$$q(s, c \mid x) = \delta(s - s_x) \delta(c - c_x),$$

where δ is the Dirac delta function. Plugging into (1) gives

$$\begin{aligned} \log p_{\theta,\phi}(x) \geq\ & \log p_\theta(x \mid s_x, c_x) \\ & + \mathbb{E}_{q_{\tilde{\phi}}(z|s_x,c_x)}[\log p_\phi(s_x, c_x \mid z)] \\ & - D_{\text{KL}}(q_{\tilde{\phi}}(z \mid s_x, c_x) \parallel p(z)). \end{aligned} \tag{2}$$

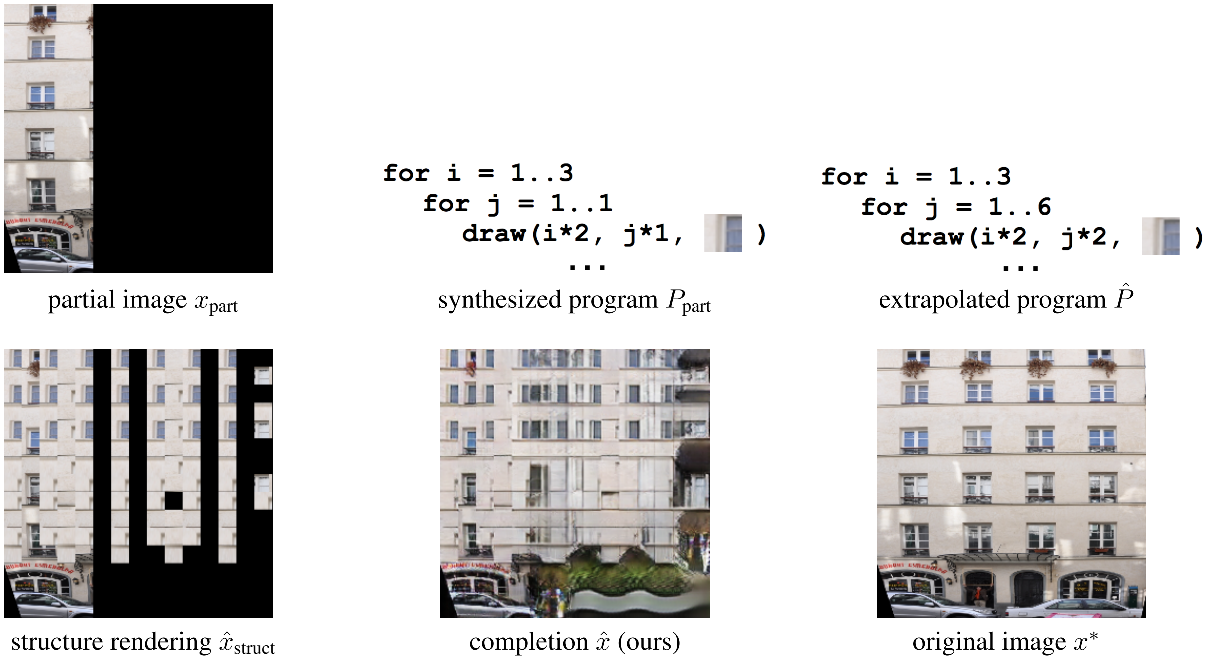


Figure 3. SGM image completion pipeline: (i) Given a partial image x_{part} (top left), use our program synthesis algorithm (Section 3) to synthesize a program P_{part} representing the structure in the partial image (top middle). (ii) Extrapolate P_{part} to a program $\hat{P} = f_\psi(P_{\text{part}})$ representing the structure of the full image. (iii) Execute $\hat{P}$ to obtain a rendering of the program structure $\hat{x}_{\text{struct}}$ (bottom left). (iv) Complete $\hat{x}_{\text{struct}}$ into an image $\hat{x}$ (bottom middle), which resembles the original image x^* (bottom right).

where we have dropped the degenerate terms $\log \delta(s - s_x)$ and $\log \delta(c - c_x)$ (which are constant with respect to the parameters $\theta, \phi, \tilde{\phi}$). As a consequence, (1) decomposes into two parts that can be straightforwardly optimized—i.e.,

$$\begin{aligned} \log p_{\theta,\phi}(x) \geq\ & \mathcal{L}(\theta; x) + \mathcal{L}(\phi, \tilde{\phi}; x) \\ \mathcal{L}(\theta; x) =\ & \log p_\theta(x \mid s_x, c_x) \\ \mathcal{L}(\phi, \tilde{\phi}; x) =\ & \mathbb{E}_{q_{\tilde{\phi}}(z \mid s_x, c_x)}[\log p_\phi(s_x, c_x \mid z)] \\ & - D_{\text{KL}}(q_{\tilde{\phi}}(z \mid s_x, c_x) \parallel p(z)), \end{aligned}$$

where we can optimize θ independently from $\phi, \tilde{\phi}$:

$$\theta^* = \arg\max_\theta \sum_{i=1}^n \mathcal{L}(\theta; x^{(i)})$$

$$\phi^*, \tilde{\phi}^* = \arg\max_{\phi,\tilde{\phi}} \sum_{i=1}^n \mathcal{L}(\phi, \tilde{\phi}; x^{(i)}).$$

Latent structure VAE. Note that $\mathcal{L}(\phi, \tilde{\phi}; x)$ is exactly equal to the objective of a VAE, where $q_{\tilde{\phi}}(z \mid s, c)$ is the encoder and $p_\phi(s, c \mid z)$ is the decoder—i.e., learning the distribution over latent structure is equivalent to learning the parameters of a VAE. The architecture of this VAE depends on the representation of s and c. In the case of for-loop structure in images, we use a sequence-to-sequence VAE.

Generating data with structure. The term $\mathcal{L}(\theta; x)$ corresponds to learning a probability distribution (conditioned on the latent structure s and c)—e.g., we can estimate this distribution using another VAE. As before, the architecture of this VAE depends on the representation of s and c. Rather than directly predicting x based on s and c, we can leverage the program structure more directly by first executing the program $P = (s, c)$ to obtain its output $x_{\text{struct}} = \text{eval}(P)$, which we call a *structure rendering*. In particular, x_{struct} is a more direct representation of the global structure represented by P, so it is often more suitable to use as input to a neural network. The middle of Figure 2 shows an example of a structure rendering for the program on the left. Then, we can train a model $p_\theta(x \mid s, c) = p_\theta(x \mid x_{\text{struct}})$.

In the case of images, we use a VAE with convolutional layers for the encoder q_ϕ and transpose convolutional layers for the decoder p_θ. Furthermore, instead of estimating the entire distribution $p_\theta(x \mid s, c)$, we also consider two non-probabilistic approaches that directly predict x from x_{struct}, which is an image completion problem. We can solve this problem using GLCIC, a state-of-the-art image completion model (IIzuka et al., 2017). We can also use CycleGAN (Zhu et al., 2017), which solves the more general problem of mapping a training set of structured renderings $\{x_{\text{struct}}\}$ to a training set of completed images $\{x\}$.

Image completion. In image completion, we are given a set of training pairs (x_{part}, x^*), and the goal is to learn a model that predicts the complete image x^* given a partial image x_{part}. Compared to generation, our likelihood is now conditioned on x_{part}—i.e., $p_{\theta,\phi}(x \mid x_{\text{part}})$. Now, we describe how we modify each of our two models $p_\theta(x \mid s, c)$ and $p_\phi(s, c \mid z)$ to incorporate this extra information.

First, the programmatic structure is no longer fully latent, since we can observe the structure in x_{part}. In particular, we leverage our program synthesis algorithm to help perform completion. We first synthesize programs P^* and P_{part} representing the global structure in x^* and x_{part}, respectively. Then, we train a model f_ψ that predicts P^* given P_{part}—i.e., it extrapolates P_{part} to a program $\hat{P} = f_\psi(P_{\text{part}})$ representing the structure of the full image. Thus, unlike generation, where we sample a program $\hat{P} = (s, c) \sim p_\phi(s, c \mid z)$, we use the extrapolated program $\hat{P} = f_\psi(P_{\text{part}})$. Second, when we execute $\hat{P} = (s, c)$ to obtain a structure rendering x_{struct}, we render it on top of the given partial image x_{part}. Finally, we sample a completion $\hat{x} \sim p_\theta(x \mid x_{\text{struct}})$ as before. Our image completion pipeline is shown in Figure 3.

3. Synthesizing Programmatic Structure

Image representation. Since the images we work with are very high dimensional, for tractability, we assume that each image $x \in \mathbb{R}^{NM \times NM}$ is divided into a grid containing N rows and N columns, where each grid cell has size $M \times M$ pixels (where $M \in \mathbb{N}$ is a hyperparameter). For example, this grid structure is apparent in Figure 3 (top right), where $N = 15$, $M = 17$ and $N = 9$, $M = 16$ for the facade and synthetic datasets respectively. For $t, u \in [N] = \{1, ..., N\}$, we let $x_{tu} \in \mathbb{R}^{M \times M}$ denote the sub-image at the (t, u) position in the $N \times N$ grid.

Program grammar. Given this structure, we consider programs that draw 2D repeating patterns of $M \times M$ sub-images on the grid. More precisely, we consider programs

$$P = ((s_1, c_1), ..., (s_k, c_k)) \in (S \times C)^k$$

that are length k lists of pairs consisting of a sketch $s \in S$ and a perceptual component $c \in C$; here, $k \in \mathbb{N}$ is a hyperparameter. [1] A sketch $s \in S$ has form

$$\begin{aligned} s = \ & \textbf{for } (i, j) \in \{1, ..., n\} \times \{1, ..., n'\} \textbf{ do} \\ & \qquad \text{draw}(a \cdot i + b,\ a' \cdot j + b',\ \textbf{??}) \\ & \textbf{end for} \end{aligned}$$

where $n, a, b, n', a', b' \in \mathbb{N}$ are undetermined parameters that must satisfy $a \cdot n + b \le N$ and $a' \cdot n' + b' \le N$, and where **??** is a hole to be filled by a perceptual component, which is an $M \times M$ sub-image $c \in \mathbb{R}^{M \times M}$. [2] Then, upon executing the (i, j) iteration of the for-loop, the program renders sub-image I at position $(t, u) = (a \cdot i + b, a' \cdot j + b')$ in the $N \times N$ grid. Figure 3 (top middle) shows an example of a sketch s where its hole is filled with a sub-image c, and Figure 3 (bottom left) shows the image rendered upon executing $P = (s, c)$. Figure 2 shows another such example.

Program synthesis problem. Given a training image $x \in \mathbb{R}^{NM \times NM}$, our program synthesis algorithm outputs the parameters $n_h, a_h, b_h, n'_h, a'_h, b'_h$ of each sketch s_h in the program (for $h \in [k]$), along with a perceptual component c_h to fill the hole in sketch s_h. Together, these parameters define a program $P = ((s_1, c_1), ..., (s_k, c_k))$.

The goal is to synthesize a program that faithfully represents the global structure in x. We capture this structure using a boolean tensor $B^{(x)} \in \{0, 1\}^{N \times N \times N \times N}$, where

$$B^{(x)}_{t,u,t',u'} = \begin{cases} 1 & \text{if } d(x_{tu}, x_{t'u'}) \le \epsilon \\ 0 & \text{otherwise,} \end{cases}$$

where $\epsilon \in \mathbb{R}_+$ is a hyperparameter, and $d(I, I')$ is a distance metric between on the space of sub-images. In our implementation, we use a weighted sum of earthmover's distance between the color histograms of I and I', and the number of SIFT correspondences between I and I'.

Additionally, we associate a boolean tensor with a given program $P = ((s_1, c_1), ..., (s_k, c_k))$. First, for a sketch $s \in S$ with parameters a, b, n, a', b', n', we define

$$\text{cover}(s) = \{(a \cdot i + b, a' \cdot j + b') \mid i \in [n], j \in [n']\},$$

i.e., the set of grid cells where sketch renders a sub-image upon execution. Then, we have

$$B^{(s)}_{t,u,t',u'} = \begin{cases} 1 & \text{if } (t, u), (t', u') \in \text{cover}(s) \\ 0 & \text{otherwise,} \end{cases}$$

i.e., $B^{(s)}_{t,u,t',u'}$ indicates whether the sketch s renders a sub-image at both of the grid cells (t, u) and (t', u'). Then,

$$B^{(P)} = B^{(s_1)} \vee ... \vee B^{(s_k)},$$

where the disjunction of boolean tensors is defined elementwise. Intuitively, $B^{(P)}$ identifies the set of pairs of grid cells (t, u) and (t', u') that are equal in the image rendered upon executing each pair (s, c) in P. [3]

Finally, our program synthesis algorithm aims to solve the following optimization problem:

$$P^* = \arg\max_P \ell(P; x) \tag{3}$$

$$\ell(P; x) = \|B^{(x)} \wedge B^{(P)}\|_1 + \lambda \|\neg B^{(x)} \wedge \neg B^{(P)}\|_1,$$

[1] So far, we have assumed that a program is a single pair $P = (s, c)$, but the generalization to a list of pairs is straightforward.

[2] For colored images, we have $I \in \mathbb{R}^{M \times M \times 3}$.

[3] Note that the covers of different sketches in P can overlap; ignoring this overlap does not significantly impact our results.

Algorithm 1 Synthesizes a program P representing the global structure of a given image $x \in \mathbb{R}^{NM \times NM}$.

Input: $X = \{x\} \subseteq \mathbb{R}^{NM \times NM}$
$\hat{C} \leftarrow \{x_{tu} \mid t, u \in [N]\}$
$P \leftarrow \varnothing$
for $h \in \{1, ..., k\}$ **do**
  $s_h, c_h = \arg\max_{(s,c) \in S \times \hat{C}} \ell(P_{h-1} \cup \{(s,c)\}; x)$
  $P \leftarrow P \cup \{(s_h, c_h)\}$
end for
Output: P

where $\wedge$ and $\neg$ are applied elementwise, and $\lambda \in \mathbb{R}_+$ is a hyperparameter. In other words, the objective of (3) is the number of true positives (i.e., entries where $B^{(P)} = B^{(x)} = 1$), and the number of false negatives (i.e., entries where $B^{(P)} = B^{(x)} = 0$), and computes their weighted sum. Thus, the objective of (3) measures for how well P represents the global structure of x. For tractability, we restrict the search space in (3) to programs of the form

$$P = ((s_1, c_1), ..., (s_k, c_k)) \in (S \times \hat{C})^k$$
$$\hat{C} = \{x_{tu} \mid t, u \in [N]\}.$$

In other words, rather than searching over all possible sub-images $c \in \mathbb{R}^{M \times M}$, we only search over the sub-images that actually occur in the training image x. This may lead to a slightly sub-optimal solution, for example, in cases where the optimal sub-image to be rendered is in fact an interpolation between two similar but distinct sub-images in the training image. However, we found that in practice this simplifying assumption still produced viable results.

Program synthesis algorithm. Exactly optimizing (3) is in general an NP-complete problem. Thus, our program synthesis algorithm uses a partially greedy heuristic. In particular, we initialize the program to $P = \varnothing$. Then, on each iteration, we enumerate all pairs $(s, c) \in S \times \hat{C}$ and determine the pair (s_h, c_h) that most increases the objective in (3), where $\hat{C}$ is the set of all sub-images x_{tu} for $t, u \in [N]$. Finally, we add (s_h, c_h) to P. We show the full algorithm in Algorithm 1.

Theorem 3.1. *If $\lambda = 0$, then $\ell(\hat{P}; x) \geq (1 - e^{-1})\ell(P^*; x)$, where $\hat{P}$ is returned by Algorithm 1 and P^* solves (3).*

Proof. If $\lambda = 0$, then optimizing $\ell(P; x)$ is equivalent to set cover, where the items are tuples

$$\{(t, u, t', u') \in [N]^4 \mid B^{(x)}_{t,u,t',u'} = 1\},$$

and the sets are $(s, c) \in S \times \hat{C}$. The theorem follows from (Hochbaum, 1997). □

In general, (3) is not submodular, but we find that the greedy heuristic still works well in practice.

4. Experiments

We perform two experiments—one for generation from scratch and one for image completion. We find substantial improvement in both tasks. Details on neural network architectures are in Appendix A, and additional examples for image completion are in Appendix B.

4.1. Datasets

Synthetic dataset. We developed a synthetic dataset based on MNIST. Each image consists of a 9×9 grid, where each grid cell is 16×16 pixels. Each grid cell is either filled with a colored MNIST digit or a solid color background. The program structure is a 2D repeating pattern of an MNIST digit; to add natural noise, we each iteration of the for-loop in a sketch s_h renders different MNIST digits, but with the same MNIST label and color. Additionally, we chose the program structure to contain correlations characteristic of real-world images—e.g., correlations between different parts of the program, correlations between the program and the background, and noise in renderings of the same component. Examples are shown in Figure 4. We give details of how we constructed this dataset in Appendix A. This dataset contains 10,000 training and 500 test images.

Facades dataset. Our second dataset consists of 1855 images (1755 training, 100 testing) of building facades.[4] These images were all scaled to a size of $256 \times 256 \times 3$ pixels, and were divided into a grid of 15×15 cells each of size 17 or 18 pixels. These images contain repeating patterns of objects such as windows and doors.

4.2. Generation from Scratch

Experimental setup. First, we evaluate our approach SGM applied to generation from scratch. We focus on the synthetic dataset—we found that our facades dataset was too small to produce meaningful results. For the first stage of SGM (i.e., generating the program $P = (s, c)$), we use a LSTM architecture for the encoder $p_\phi(s, c \mid z)$ and a feedforward architecture for the decoder $q_{\tilde{\phi}}(z \mid s, c)$. As described in Section 2, we use Algorithm 1 to synthesize programs $P_x = (s_x, c_x)$ representing each training image $x \in X_{\text{train}}$. Then, we train p_ϕ and $q_{\tilde{\phi}}$ on the training set of programs $\{P_x \mid x \in X\}$.

For the second stage of SGM (i.e., completing the structure rendering x_{struct} into an image x), we use a variational encoder-decoder (VED)

$$p_\theta(x \mid s, c) = \int p_\theta(x \mid w) \cdot q_\theta(w \mid x_{\text{struct}}) dw,$$

where $q_\theta(w \mid x_{\text{struct}})$ encodes a structure rendering x_{struct}

[4] We chose a large training set since our dataset is so small.

Model	Score
SGM (CycleGAN)	**85.51**
BL (SpatialGAN)	258.68
SGM (VED)	**59414.7**
BL (VAE)	60368.4
SGM (VED Stage 1 $p_\phi(s, c \mid z)$)	32.0
SGM (VED Stage 2 $p_\theta(x \mid s, c)$)	59382.6

Table 1. Performance of our approach SGM versus the baseline (BL) for generation from scratch. We report Fréchet inception distance for GAN-based models, and negative log-likelihood for the VAE-based models

into a latent vector w, and $p_\theta(x \mid w)$ decodes the latent vector to a whole image We train p_θ and q_θ using the reconstruction error $\|\hat{x} - x^*\|$. Additionally, we trained a Cycle-GAN model to map structure renderings to complete images, by giving the CycleGAN model unaligned pairs of x_{struct} and x^* as training data. We compare our VED model to a VAE (Kingma & Welling, 2014), and compare our CycleGAN model to a SpatialGAN (Jetchev et al., 2017).

Results. We measure performance for SGM with the VED and the baseline VAE using the variational lower bound on the negative log-likelihood (NLL) (Zhao et al., 2017) on a held-out test set. For our approach, we use the lower bound (2),[5] which is the sum of the NLLs of the first and second stages; we report these NLLs separately as well. Figure 4 shows examples of generated images. For SGM and SpatialGAN, we use Fréchet inception distance (Heusel et al., 2017). Table 1 shows these metrics of both our approach and the baseline.

Discussion. The models based on our approach quantitatively improve over the respective baselines. The examples of images generated using our approach with VED completion appear to contain more structure than those generated using the baseline VAE. Similarly, the images generated using our approach with CycleGAN clearly capture more complex structure than the unbounded 2D repeating texture patterns captured by SpatialGAN.

4.3. Image Completion

Experimental setup. Second, we evaluated our approach SGM for image completion, on both our synthetic and the facades dataset. For this task, we compare using three image completion models: GLCIC (IIzuka et al., 2017), CycleGAN (Zhu et al., 2017), and the VED architecture described in Section 4.2. GLCIC is a state-of-the-art image completion model. CycleGAN is a generic image-to-image transformer.

[5]Technically, $p_\theta(x \mid s_x, c_x)$ is lower bounded by the loss of the variational encoder-decoder).

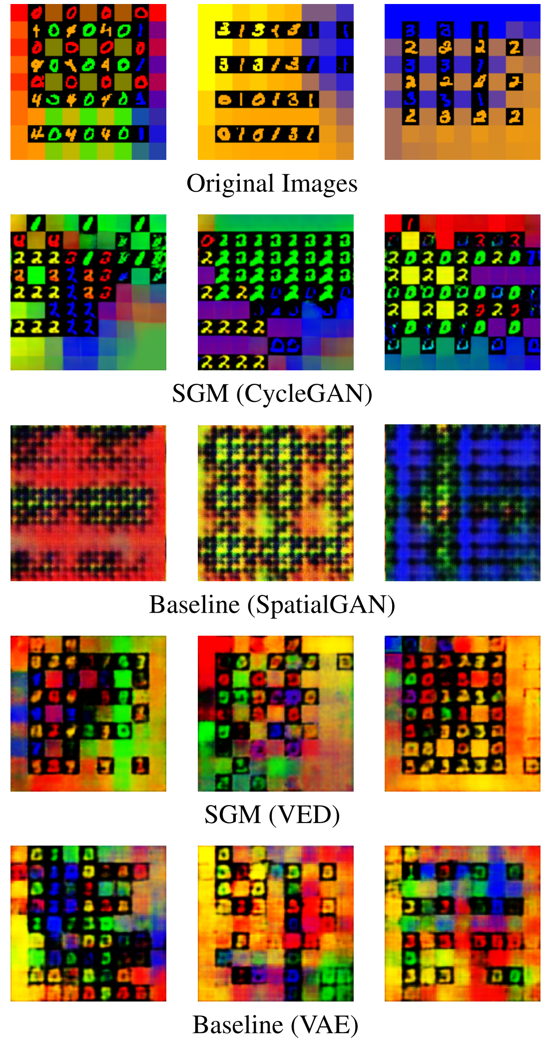


Figure 4. Examples of synthetic images generated using our approach, SGM (with VED and CycleGan), and the baseline (a VAE and a SpatialGAN). Images in different rows are unrelated since the task is generation from scratch.

It uses unpaired training data, but we found that for our task, it outperforms approaches such as Pix2Pix (Isola et al., 2017) that take paired training data. For each model, we trained two versions:

- **Our approach (SGM):** As described in Section 2 (for image completion), given a partial image x_{part}, we use Algorithm 1 to synthesize a program P_{part}. We extrapolate P_{part} to $\hat{P} = f_\psi(P_{\text{part}})$, and execute $\hat{P}$ to obtain a structure rendering x_{struct}. Finally, we train the image completion model (GLCIC, CycleGAN, or VED) to complete x_{struct} to the original image x^*.
- **Baseline:** Given a partial image x_{part}, we train the image completion model (GLCIC, CycleGAN, or VED) to directly complete x_{part} to the original image $x*$.

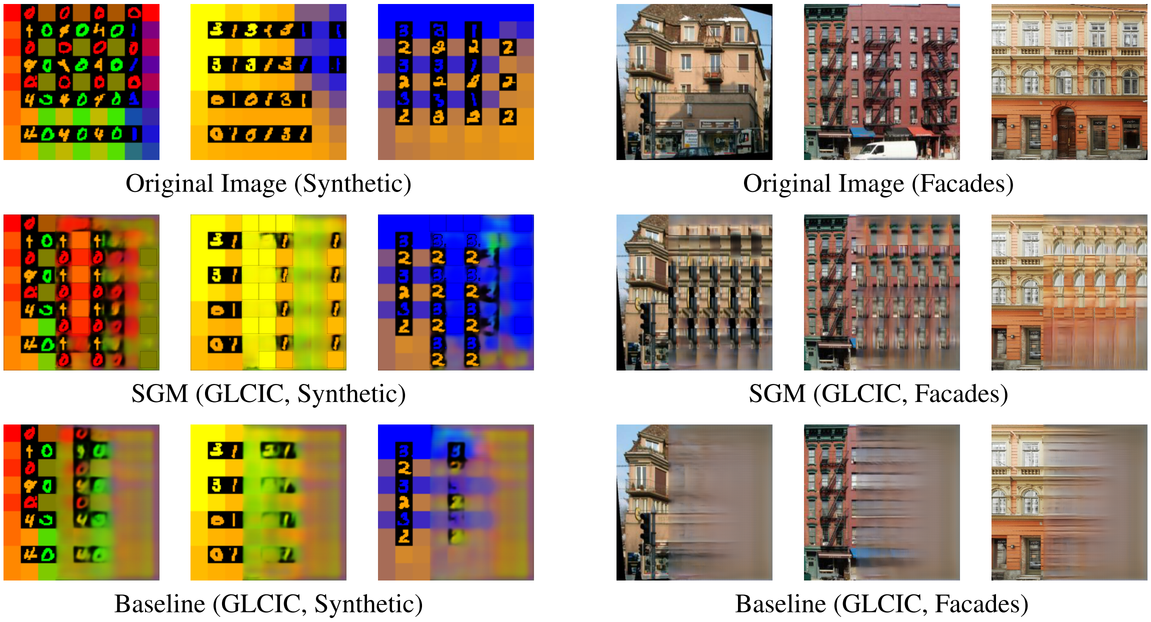


Figure 5. Examples of images generated using our approach (SGM) and the baseline, using GLCIC for image completion.

Model	Synthetic		Facades	
	SGM	BL	SGM	BL
GLCIC	**106.8**	163.66	**141.8**	195.9
CycleGAN	**91.8**	218.7	**124.4**	251.4
VED	**44570.4**	52442.9	8755.4	**8636.3**

Table 2. Performance of our approach SGM versus the baseline (BL) for image completion. We report Fréchet distance for GAN-based models, and negative log-likelihood (NLL) for the VED.

Results. As in Section 4.2, we measure performance using Fréchet inception distance for GLCIC and CycleGAN, and negative log-likelihood (NLL) for the VED, reported on a held-out test set. We show these results in Table 2. We show examples of completed image using GLCIC in Figure 5. We show additional examples of completed images, including those completed using CycleGAN and VED, in Appendix B.

Discussion. Our approach SGM outperforms the baseline in every case except the VED on the facades dataset. We believe the last result is since both VEDs failed to learn any meaningful structure (see Figure 7 in Appendix B).

A key reason why the baselines perform so poorly on the facades dataset is that the dataset is very small. Nevertheless, SGM substantially outperforms the baselines even on the larger synthetic dataset. Finally, generative models such as GLCIC are known to perform poorly away from the edges of the given partial image (IIzuka et al., 2017). A benefit of our approach is that it provides global context for models such as GLCIC that are good at performing local completion.

5. Conclusion

We have proposed a new approach to generation that incorporates programmatic structure into state-of-the-art deep learning models. In our experiments, we have demonstrated the promise of our approach to improve generation of high-dimensional data with global structure that current state-of-the-art deep generative models have difficulty capturing.

There are a number of directions for future work that could improve the quality of the images generated using our approach. Most importantly, we have relied on a relatively simple grammar of programs. Designing more expressive program grammars that can more accurately capture global structure could substantially improve our results. Examples of possible extensions include if-then-else statements and variable grids. Furthermore, it may be useful to incorporate spatial transformations so we can capture patterns that are distorted due to camera projection.

Correspondingly, more sophisticated synthesis algorithms may be needed for these domains. In particular, learning-based program synthesizers may be necessary to infer more complex global structure. Devising new learning algorithms—e.g., based on reinforcement learning—would be needed to learn these synthesizers in conjunction with the parameters of the SGM model.

Acknowledgements

We thank the anonymous reviewers for insightful feedback. This research was supported by NSF awards #1737858 and #1836936.

References

Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 39–48, 2016.

Bastani, O., Pu, Y., and Solar-Lezama, A. Verifiable reinforcement learning via policy extraction. In *NIPS*, 2018.

De Raedt, L., Kimmig, A., and Toivonen, H. Problog: A probabilistic prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pp. 2468–2473, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc. URL http://dl.acm.org/citation.cfm?id=1625275.1625673.

Deng, Z., Chen, J., Fu, Y., and Mori, G. Probabilistic neural programmed networks for scene generation. In *Advances in Neural Information Processing Systems*, pp. 4032–4042, 2018.

Dieleman, S., van den Oord, A., and Simonyan, K. The challenge of realistic music generation: modelling raw audio at scale. In *Advances in Neural Information Processing Systems*, pp. 8000–8010, 2018.

Ellis, K., Solar-Lezama, A., and Tenenbaum, J. Unsupervised learning by program synthesis. In *Advances in neural information processing systems*, pp. 973–981, 2015.

Ellis, K., Ritchie, D., Solar-Lezama, A., and Tenenbaum, J. Learning to infer graphics programs from hand-drawn images. In *Advances in Neural Information Processing Systems*, pp. 6060–6069, 2018.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Ha, D. and Eck, D. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*, 2017.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, pp. 6626–6637, 2017.

Hochbaum, D. S. Approximating covering and packing problems: set cover, vertex cover, independent set, and related problems. *Approximation Algorithms for NP-Hard Problem*, pp. 94–143, 1997.

Hwang, I., Stuhlmüller, A., and Goodman, N. D. Inducing probabilistic programs by bayesian program merging. *arXiv preprint arXiv:1110.5667*, 2011.

IIzuka, S., Simo-Serra, E., and Ishikawa, H. Globally and locally consistent image completion. In *ACM Trans. Graph.*, 2017.

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. Image-to-image translation with conditional adversarial networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5967–5976. IEEE, 2017.

Jetchev, N., Bergmann, U., and Vollgraf, R. Texture synthesis with spatial generative adversarial networks. 2017.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *ICLR*, 2014.

Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Liang, P., Jordan, M. I., and Klein, D. Learning programs: A hierarchical bayesian approach. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 639–646, 2010.

Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., and Raedt, L. D. Deepproblog: Neural probabilistic logic programming. *CoRR*, abs/1805.10872, 2018. URL http://arxiv.org/abs/1805.10872.

Nori, A. V., Ozair, S., Rajamani, S. K., and Vijaykeerthy, D. Efficient synthesis of probabilistic programs. In *PLDI*, volume 50, pp. 208–217. ACM, 2015.

Perov, Y. N. and Wood, F. Learning probabilistic programs. In *NIPS Probabilistic Programming Workshop*, 2014.

Pu, Y., Miranda, Z., Solar-Lezama, A., and Kaelbling, L. Selecting representative examples for program synthesis. In *International Conference on Machine Learning*, pp. 4158–4167, 2018.

Radford, A., Metz, L., and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2015.

Solar-Lezama, A., Tancau, L., Bodik, R., Seshia, S., and Saraswat, V. Combinatorial sketching for finite programs. In *ASPLOS*, volume 41, pp. 404–415. ACM, 2006.

Talton, J., Yang, L., Kumar, R., Lim, M., Goodman, N., and Měch, R. Learning design patterns with bayesian grammar induction. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, pp. 63–74. ACM, 2012.

Tyleček, R. and Šára, R. Spatial pattern templates for recognition of objects with regular structure. In *Proc. GCPR*, Saarbrucken, Germany, 2013.

Valkov, L., Chaudhari, D., Srivastava, A., Sutton, C., and Chaudhuri, S. Houdini: Lifelong learning as program synthesis. In *Advances in Neural Information Processing Systems*, pp. 8701–8712, 2018.

Verma, A., Murali, V., Singh, R., Kohli, P., and Chaudhuri, S. Programmatically interpretable reinforcement learning. In *ICML*, 2018.

Wang, F. and Rudin, C. Falling rule lists. In *Artificial Intelligence and Statistics*, pp. 1013–1022, 2015.

Xian, W., Sangkloy, P., Agrawal, V., Raj, A., Lu, J., Fang, C., Yu, F., and Hays, J. Texturegan: Controlling deep image synthesis with texture patches. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8456–8465, 2018.

Zhao, S., Song, J., and Ermon, S. Towards deeper understanding of variational autoencoding models. *arXiv preprint arXiv:1702.08658*, 2017.

Zhu, J.-Y., Park, T., and Efros, A. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV.*, 2017.