SPATIAL DEPENDENCY NETWORKS: NEURAL LAYERS FOR IMPROVED GENERATIVE IMAGE MODELING

Anonymous authors
Paper under double-blind review

ABSTRACT

How to improve generative modeling by better exploiting spatial regularities and coherence in images? To answer this question, we introduce a novel neural layer for building image generators (decoders) and apply it to variational autoencoders (VAEs). Our spatial dependency networks (SDNs) compute feature maps in a spatially coherent way, utilizing a sequential gating-based mechanism to distribute contextual information across 2-D maps. Different SDN layers of a deep neural network represent spatial dependencies at different levels of abstraction. Augmenting the decoder of a hierarchical VAE by spatial dependencies considerably improves density estimation over baseline convolutional architectures and the state-of-the-art among the models within the same class. We show that SDN can be applied to large images by synthesizing samples of high quality and coherence. In a vanilla VAE setting, we find that a powerful SDN decoder also improves learning disentangled representations, indicating that neural architectures play an important role in this task. Our results suggest favoring SDNs over convolutional networks in various VAE settings.

1 INTRODUCTION

The abundance of data and computation are often identified as core facilitators of the deep learning revolution. In addition to this technological leap, historically speaking, most major algorithmic advancements critically hinged on the existence of inductive biases, incorporating prior knowledge in different ways. Main breakthroughs in image recognition (Cireşan et al., 2012; Krizhevsky et al., 2012) were preceded by the long-standing pursuit for shift-invariant pattern recognition (Fukushima & Miyake, 1982) which catalyzed the ideas of weight sharing and convolutions (Waibel et al., 1989; LeCun et al., 1989). Recurrent networks (exploiting temporal recurrence) and transformers (modeling the "attention" bias) revolutionized the field of natural language processing (Mikolov et al., 2011; Vaswani et al., 2017). Visual representation learning is also often based on priors e.g. independence of latent factors (Schmidhuber, 1992; Bengio et al., 2013) or invariance to input transformations (Becker & Hinton, 1992; Chen et al., 2020). Clearly, one promising strategy to move forward is to introduce more structure into learning algorithms, and more knowledge on the problems and data.

Along this line of thought, we explore a way to improve the architecture of deep neural networks that generate images, here referred to as (deep) image generators, by incorporating prior assumptions based on topological image structure. More specifically, we aim to integrate the priors on spatial dependencies in images. We would like to enforce these priors on all intermediate image representations produced by an image generator, including the last one from which the final image is synthesized. To that end, we introduce a neural layer designed specifically for building image generators – spatial dependency network (SDN).

Concretely, the proposed SDN layer incorporates two priors: (i) spatial coherence reflects our assumption that feature vectors should be dependent on each other in a spatially consistent, smooth way. Thus in SDN, the neighboring feature vectors tend to be more similar than the non-neighboring ones, where the similarity correlates with the 2-D distance. The graphical model of SDN captures this assumption (see Figure 1(a)). Note that SDN can nevertheless model long-range dependencies; (ii) spatial dependencies between feature vectors should not depend on their 2-D coordinates. Mathematically speaking, SDN should be equivariant to spatial translation, in the same way as con-

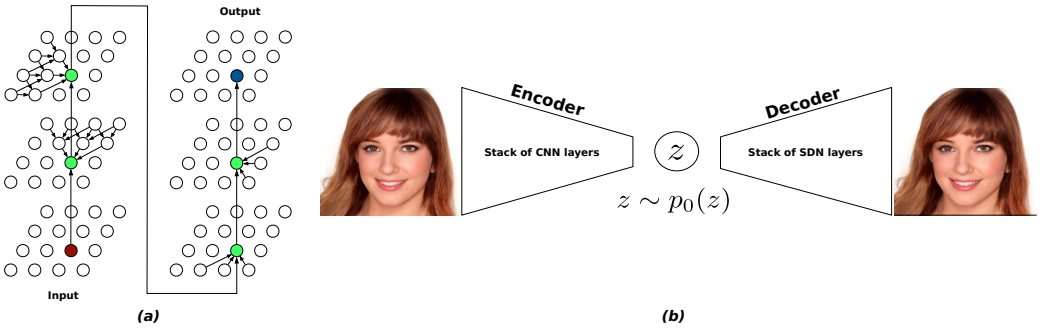


Figure 1: **(a) Directed graphical model of SDN.** The input feature vector (red node) is gradually refined (green nodes) as the computation progresses through the four sub-layers until the output feature vector is produced (blue node). In each sub-layer, the feature vector is corrected based on contextual information. Conditional maps within sub-layers are implemented as learnable deterministic functions with shared parameters; **(b) VAE with SDN decoder reconstructing a 'celebrity'.**

volutional networks (CNN) (Goodfellow et al., 2016). This property is achieved through parameter (weight) sharing both in SDN and CNN.

The particular focus of this paper is the application of SDN to variational autoencoders (VAEs) (Kingma & Welling, 2013). The main motivation is to improve the performance of VAE generative models by endowing their image decoders with SDNs (Figure 1(b)). While out of the scope of this work, SDN could also be applied to other generative models, e.g. generative adversarial networks (Goodfellow et al., 2014). More generally, SDN could be potentially used in any task in which image generation is required, such as image-to-image translation, super-resolution, image in-painting, image segmentation, or scene labeling.

SDN is experimentally examined in two different settings. In the context of real-life-image density modeling, SDN-empowered hierarchical VAE is shown to reach considerably higher test log-likelihoods than the baseline CNN-based architectures and can synthesize perceptually appealing and coherent images even at high sampling temperatures. In a synthetic data setting, we observe that enhancing a non-hierarchical VAE with an SDN facilitates learning of factorial latent codes, suggesting that unsupervised 'disentanglement' of representations can be bettered by using more powerful neural architectures, where SDN stands out as a good candidate model. The contributions and the contents of this paper can be summarized as follows.

CONTRIBUTIONS AND CONTENTS:

- The architecture of SDN is introduced and then contrasted to convolutional networks and self-attention. The relationship of SDN with autoregressive density models and normalizing flows is also explored.
- The architecture of SDN-VAE is introduced – the result of applying SDN to IAF-VAE (Kingma et al., 2016), with modest modifications to the original architecture. SDN-VAE is evaluated in terms of: *(a)* density estimation, where the performance is considerably improved both upon the baseline and related approaches to non-autoregressive modeling, on CIFAR10 and ImageNet32 data sets; *(b)* image synthesis, where images of competitively high perceptual quality are generated based on CelebAHQ256 data set.
- By integrating SDN into a relatively simple, non-hierarchical VAE, trained on the synthetic 3D-Shapes data set, we demonstrate in another comparative analysis substantial improvements upon convolutional networks in terms of: *(a)* optimization of the evidence lower bound (ELBO); *(b)* learning of disentangled representations with respect to two popular metrics, when β-VAE (Higgins et al., 2016) is used as an objective function.
- The implementation of the proposed architectures and of our experiments is disclosed in the frameworks of Pytorch (Paszke et al., 2019) and PytorchLightning (Falcon, 2019).

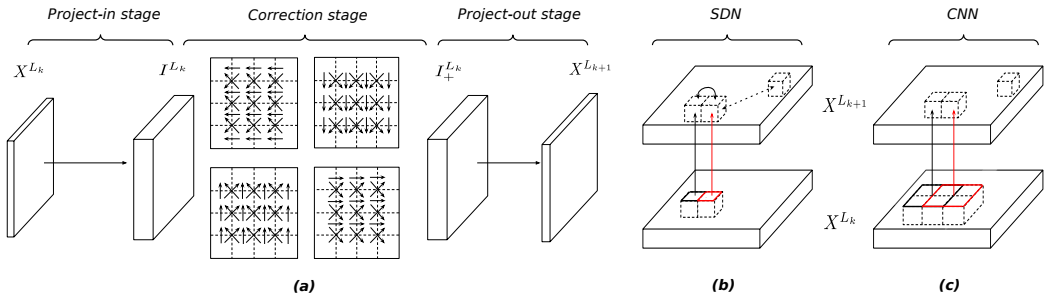


Figure 2: **SDN design.** **(a)** SDN layer as a 3-stage pipeline; **(b)** dependencies between the input X^{L_k} and the output $X^{L_{k+1}}$ modeled by an SDN, at the layer L_k. Solid arrows represent direct, and dashed indirect (non-neighboring) dependencies. Note that SDN sub-layers from Figure 1(a) are regarded as 'latent' here; **(c)** dependencies modeled by a 2×2 CNN.

2 SDN Architecture

The architectural design of SDN is motivated by our intent to improve modeling of spatial coherence and higher-order statistical correlations between pixels. Schematic overview of SDN is given in Figure 2. Technically speaking, SDN layer is a differentiable function which transforms input to output feature maps in a 3-stage procedure.

We first describe the stages of SDN in detail, then address practical and computational concerns.

Project-in stage. Using a simple feature vector-wise affine transformation, the input representation X^{L_k} at the layer L_k is transformed into the intermediate representation I^{L_k}:

$$I_{i,j}^{L_k} = X_{i,j}^{L_k} \mathbf{W}^{(\mathbf{1})} + \mathbf{b}^{(\mathbf{1})} \tag{1}$$

where i and j are the corresponding 2-D coordinates in X^{L_k} and I^{L_k}, $\mathbf{b}^{(\mathbf{1})}$ is a vector of learnable biases and $\mathbf{W}^{(\mathbf{1})}$ is a learnable matrix of weights whose dimensions depend on the number of channels of X^{L_k} and I^{L_k}. In the absence of overfitting, it is advisable to increase the number of channels to enhance the memory capacity of I^{L_k} and enable unimpeded information flow in the correction stage. Notice that described transformation corresponds to the convolutional (CNN) operator with a kernel of size 1. If the scales between the input X^{L_k} and the output $X^{L_{k+1}}$ differ (e.g. if converting an 8x8 to a 16x16 tensor), in this stage we additionally perform upsampling by applying the corresponding transposed CNN operator, also known as 'deconvolution' (Zeiler & Fergus, 2014).

Correction stage. The elements of I^{L_k} are 'corrected' in a 4-step procedure. In each step, a unidirectional sweep through I^{L_k} is performed in one of the 4 possible directions: *(i)* bottom-to-top; *(ii)* right-to-left; *(iii)* left-to-right; and *(iv)* top-to-bottom. During a single sweep, columns/rows are computed in parallel. The corresponding correction equations implement controlled updates of I^{L_k} using a gating mechanism, as done in popular recurrent units (Hochreiter & Schmidhuber, 1997; Cho et al., 2014). Specifically, we use a multiplicative update akin to the one found in GRUs (Cho et al., 2014), but adapted to the spatial setting. The procedure for the bottom-to-top sweep is given in Algorithm 1 where $\mathbf{W}^*$ denote learnable weights, $\mathbf{B}^*$ are learnable biases, i is the height (growing bottom-to-top), j is the width and c is the channel dimension. Gating elements $r_{i,j,c}$ and $z_{i,j,c}$ control the correction contributions of the *a priori* value of $I_{i,j,c}$ and the proposed value $n_{i,j,c}$, respectively. For the borderline elements of I^{L_k}, missing neighbors are initialized to zeros (hence the zero padding). The sweeps in the remaining 3 directions are performed analogously.

Project-out stage. Corrected (*a posteriori*) representation $I_+^{L_k}$ is then mapped to the output $X^{L_{k+1}}$, which has the same number of channels as the input X^{L_k}:

$$X_{i,j}^{L_{k+1}} = I_{+,i,j}^{L_k} \mathbf{W}^{(\mathbf{3})} + \mathbf{b}^{(\mathbf{3})} \tag{2}$$

where $\mathbf{W}^{(\mathbf{3})}$ is the stage-level learnable weight matrix and $\mathbf{b}^{(\mathbf{3})}$ is the corresponding bias vector.

Algorithm 1 Bottom-to-top sweep of the correction stage

Input: I^{L_k} – intermediate representation; N – the scale of I^{L_k};
Output: $I_+^{L_k}$ – corrected intermediate representation; # after all 4 sweeps
Complexity: $\mathcal{O}(N)$
$I^{L_k} = \tanh(I^{L_k})$ # done for the first direction only
$I^{L_k} = \textit{zero_padding}(I^{L_k})$ # done for the first direction only
for $i = 1$ **to** N and ($j = 1$ **to** N **in parallel**) **do**
  $r_{i,j,c} = \sigma([\mathbf{W^{r1}} I_{i,j}]_c + [\mathbf{W^{r2}} I_{i-1,j}]_c + [\mathbf{W^{r3}} I_{i-1,j-1}]_c) + [\mathbf{W^{r3}} I_{i-1,j+1}]_c + \mathbf{B^r}_{i,j,c})$ # reset gate
  $z_{i,j,c} = \sigma([\mathbf{W^{z1}} I_{i,j}]_c + [\mathbf{W^{z2}} I_{i-1,j}]_c + [\mathbf{W^{z3}} I_{i-1,j-1}]_c) + [\mathbf{W^{z3}} I_{i-1,j+1}]_c + \mathbf{B^z}_{i,j,c})$ #update gate

  $n_{i,j,c} = \tanh(r_{i,j,c}[\mathbf{W^{n1}} I_{i,j}]_c + [\mathbf{W^{n2}} I_{i-1,j}]_c + [\mathbf{W^{n3}} I_{i-1,j-1}]_c) + [\mathbf{W^{n3}} I_{i-1,j+1}]_c + \mathbf{B^n}_{i,j,c})$
  $I_{i,j,c} = z_{i,j,c} I_{i,j,c} + (1 - z_{i,j,c}) n_{i,j,c}$ # feature correction
end for

Computational considerations. A valid concern in terms of scaling SDN to large images is the computational complexity of $\mathcal{O}(N)$, N being the scale of the output. We make two key observations:

a. one can operate with less than 4 directions per layer, alternating them across layers to achieve the 'mixing' effect. This improves runtime with some sacrifice in performance;

b. in practice, we found it sufficient to apply SDN only at lower scales (e.g. up to 64x64 on CelebAHQ256), with little to no loss in performance. We speculate that this is due to redundancies in high-resolution images, which allows modelling of relevant spatial structure at lower scales;

To provide additional insights on the computational characteristics, we compared SDN to CNN in a unit test-like setting (Appendix A.3). Our implementation of a 2-directional SDN has roughly the same number of parameters as a 5×5CNN, and is around 10 times slower than a 3×3CNN. However, we also observed that in a VAE context, the overall execution time discrepancy is (about 2-3 times) smaller, partially because SDN converges in fewer iterations (Table 4 in Appendix).

Avoiding vanishing gradients. The problem might arise when SDN is plugged in a very deep neural network (such as the one discussed in Section 4). To remedy this, in our SDN-VAE experiments we used the variant of SDN with 'highway' connection, i.e. gated residual (Srivastava et al., 2015), to which we refer to as ResSDN (shown in Figure 3).

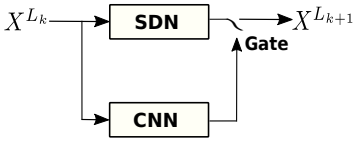


Figure 3: **Residual SDN (ResSDN).**

3 Related Work

Convolutional networks. Both SDN and CNN layers exploit locality (a feature vector in SDN is corrected based on its direct neighbors) and equivariance to translation (achieved through parameter sharing). Additionally, SDN enforces spatial coherence. There are also differences in the dependency modeling (depicted in Figures 2(b) and 2(c)). Firstly, when conditioned on the input X^{L_k}, the output feature vectors of CNN are statistically independent: $X_{i,j}^{L_{k+1}} \perp\!\!\!\perp X_{l,m}^{L_{k+1}} | X^{L_k}\ \forall (i,j) \neq (l,m)$, where i and j are feature map coordinates. In SDNs, all feature vectors are dependent: $X_{i,j}^{L_{k+1}} \not\perp\!\!\!\perp X_{l,m}^{L_{k+1}} | X^{L_k}\ \forall (i,j) \neq (l,m)$. Secondly, in the observed one-layer setup, the unconditional dependency neighborhood of CNN is bounded by the size of its receptive field – a conceptual limitation not present in SDN. Hence unlike CNN, SDN can model long-range spatial dependencies.

Autoregressive models and normalizing flows. SDN design is inspired by how autoregressive models capture spatial dependencies due to the pixel-by-pixel generation process (Theis & Bethge, 2015). SDN(-VAE) improves on this is by modeling spatial dependencies in multiple directions. Autoregressive models are inherently uni-directional; the generation order of pixels can be changed but is fixed for training and sampling. Thus in some sense, SDN transfers the ideas from autoregressive to non-autoregressive settings in which there are no ordering constraints. Also, most autoregressive

models use teacher forcing during training to avoid sequential computation, but sampling time complexity is quadratic. SDN has linear complexity in both cases and can operate at smaller scales only. Parallel computation of SDN is similar to the one found in PixelRNN (Van Oord et al., 2016), but instantiated in the non-autoregressive setting with no directionality or conditioning constraints.

One can also draw parallels with how autoregressive models describe normalizing flows, for example, IAF (Kingma et al., 2016) and MAF (Papamakarios et al., 2017). In this case, each flow in a stack of flows is described with an autoregressive model that operates in a different direction, or more generally, has a different output ordering. In its standard 4-directional form (Figure 1(a)), SDN creates dependencies between all input and output feature vectors; this renders a full, difficult-to-handle Jacobian matrix. For this reason, SDN is not directly suitable for parameterizing normalizing flows in the same way. In contrast, the main application domain of SDN is rather the parameterization of deterministic, unconstrained mappings.

Self-attention. The attention mechanism has been one of the most prominent models in the domain of text generation (Vaswani et al., 2017). Recently, it has also been applied to generative image modeling (Parmar et al., 2018; Zhang et al., 2019). Both SDN and self-attention can model long-range dependencies. The key difference is how this is done. In SDN, only neighboring feature vectors are dependent on each other (see Figures 1 and 2) hence the spatial locality is exploited. Gated units are used to ensure that the information is propagated across large distances. On the other hand, self-attention requires an additional mechanism to incorporate positional information, but the dependencies between non-neighboring feature vectors are direct. We believe that the two models should be treated as complementary; self-attention excels in capturing long-range dependencies while SDN is better in enforcing spatial coherence, equivariance to translation, and locality. Note finally that standard self-attention is costly, with quadratic complexity in time and space in the number of pixels, which makes it $\mathcal{O}(N^4)$ in the feature map scale, if implemented naively.

Other related work. SDNs are also notably reminiscent of Multi-Dimensional RNNs (Graves et al., 2007) which were used for image segmentation (Graves et al., 2007; Stollenga et al., 2015) and recognition (Visin et al., 2015), but not in the context of generative modeling. One technical advantage of our work is the linear training time complexity in the feature map scale. Another difference is that SDN uses GRU, which is less expressive than LSTM, but more memory efficient.

4 SDN-VAE: An Improved Variational Autoencoder for Images

As a first use case, SDN is used in the SDN-VAE architecture. The basis of SDN-VAE is the IAF-VAE model (Kingma et al., 2016). Apart from integrating SDNs, we introduce additional modifications for improved performance, training stability, and reduced memory requirements. We first cover the basics of VAEs, then briefly summarize IAF-VAE and the related work. Finally, we elaborate on the novelties in the SDN-VAE design.

Background on VAEs. VAEs assume a latent variable generative model $p_\theta(X) = \int p_\theta(X, Z)dZ$ where θ are model parameters. When framed in the maximum likelihood setting, the marginal probability of data is intractable due to the integral in the above expression. VAEs take a variational approach, approximating posterior $q_\phi(Z|X)$ using a learnable function q with parameters ϕ. Following Kingma & Welling (2019), we can derive the following equality:

$$\log p_\theta(X) = \underbrace{\mathbb{E}_{q_\phi(Z|X)}\left[\log\left[\frac{p_\theta(X,Z)}{q_\phi(Z|X)}\right]\right]}_{\mathcal{L}_{\phi,\theta}(X)=ELBO} + \underbrace{\mathbb{E}_{q_\phi(Z|X)}\left[\log\left[\frac{q_\phi(Z|X)}{p_\theta(Z|X)}\right]\right]}_{KL(q_\phi(Z|X)||p_\theta(Z|X))\geq 0} \tag{3}$$

$\mathcal{L}_{\phi,\theta}(X)$ is the evidence lower bound (ELBO), since it 'bounds' the marginal log-likelihood term $\log p_\theta(X)$, where the gap is defined by the KL-divergence term on the right. In VAEs, both q_ϕ and p_θ are parametrized by deep neural networks. The model is learned via backpropagation, and a low variance approximation of the gradient $\nabla_{\phi,\theta}\mathcal{L}_{\phi,\theta}(X)$ can be computed via the *reparametrization trick* (Kingma & Welling, 2019; Rezende et al., 2014).

Background on IAF-VAE. Main advancements of IAF-VAE in comparison to the vanilla VAE include: *(a)* leveraging on a hierarchical (ladder) network of stochastic variables (Sønderby et al.,

2016) for increased expressiveness and improved training; *(b)* introducing inverse autoregressive flows to enrich simple isotropic Gaussian-based posterior with a normalizing flow; *(c)* utilizing a structured bi-directional inference procedure to encode latent variables through an encoder-decoder interplay; *(d)* introducing a ResNet block as a residual-based (He et al., 2016) instantiation of the ladder layer. For more details, we refer the reader to the original paper (Kingma et al., 2016).

Related work. BIVA (Maaløe et al., 2019) is an extension of IAF-VAE which adds skip connections across latent layers, and a stochastic bottom-up path. In our experiments, however, we were not able to improve performance in a similar way. Concurrently to our work, NVAE (Vahdat & Kautz, 2020) reported significant improvements upon the IAF-VAE baseline, by leveraging on many technical advancements including: batch normalization, depth-wise separable convolutions, mixed-precision, spectral normalization and residual parameterization of the approximate posterior. Both methods use the discretized mixture of logistics (Salimans et al., 2017).

SDN-VAE ARCHITECTURE

Our main contribution to the IAF-VAE architecture is the integration of ResSDN layers (Figure 3). We replace the convolutional layers on the top-down (generation) path of the IAF-VAE ResNet block, up to a particular scale: up to 32x32 for images of that resolution, and up to 64x64 for 256x256 images. For the convenience of the reader, we borrowed the scheme from the original paper and clearly marked the modifications (Figure 4).

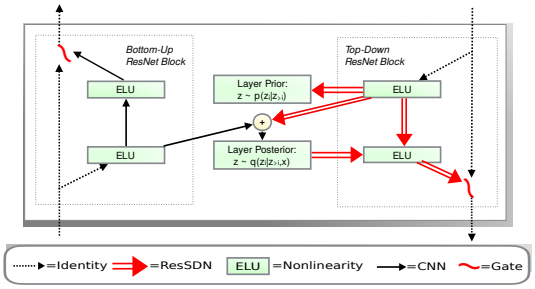


Figure 4: **SDN-VAE ResNet block as modified IAF-VAE ResNet block**. Two notable changes (marked in red) include applying: *(a)* ResSDN instead of convolutional networks on the top-down path; *(b)* gated rather than a sum residual; (figure adapted from Kingma et al. (2016))

Other notable modifications include: *(a)* gated residual instead of a sum residual in the ResNet block (Figure 4), for improved training stability; *(b)* more flexible observation model based on a discretized mixture of logistics instead of a discretized logistic distribution, for improved performance; and *(c)* mixed-precision (Micikevicius et al., 2018), which reduced memory requirements thus allowing training with larger batches. Without SDNs, we refer to this architecture as IAF-VAE+.

5 IMAGE DENSITY MODELING

Proposed SDN-VAE and the contributions of SDN layers to its performance were empirically evaluated in terms of: *(a)* density estimation, where SDN-VAE was in a quantitative fashion compared to the convolutional network baseline, and to the related state-of-the-art approaches; *(b)* image synthesis, where a learned distribution was analyzed in a qualitative way by sampling from it in different ways. Exact details of our experiments, additional ablation studies and additional results in image synthesis are documented in the Appendix sections A.1, A.2 and A.4 respectively.

Density estimation. From a set of i.i.d. images $\mathcal{D}_{train} = X_{1..N}$, the true probability density function $p(X)$ is estimated via a parametric model $p_\theta(X)$, whose parameters θ are learned using the maximum log-likelihood objective: $\arg\max_\theta \left[\log p_\theta(X) \approx \frac{1}{N}\sum_{i=1}^{N} \log p_\theta(X_i)\right]$. The test log-likelihood is computed on an isolated set of images $\mathcal{D}_{test} = X_{1..K}$, to evaluate learned $p_\theta(X)$.

SDN-VAE and the competing methods were tested on CIFAR-10 (Krizhevsky et al., 2009), ImageNet32 (Van Oord et al., 2016), and CelebAHQ256 (Karras et al., 2017). Quantitative comparison is given in Table 1. IAF-VAE+ denotes our technically enhanced implementation of IAF-VAE, but without the SDN modules (recall Section 4).

Type	Method	CIFAR-10	ImageNet32	CelebAHQ256
VAE-based	**SDN-VAE (ours)**	(2.87)	(3.85)	(0.70)
	IAF-VAE+ (ours)	3.05	4.00	0.71
	IAF-VAE (Kingma et al., 2016)	3.11	X	X
	BIVA (Maaløe et al., 2019)	3.08	X	X
	NVAE (Vahdat & Kautz, 2020)	2.91	3.92	(0.70)
Flow-based	GLOW (Kingma & Dhariwal, 2018)	3.35	4.09	1.03
	FLOW++ (Ho et al., 2019)	3.08	3.86	X
	ANF (Huang et al., 2020)	3.05	3.92	0.72
	SurVAE (Nielsen et al., 2020)	3.08	4.00	X
Autoregressive*	PixelRNN (Van Oord et al., 2016)	3.00	3.86	X
	PixelCNN (Van den Oord et al., 2016)	3.03	3.83	X
	PixelCNN++ (Salimans et al., 2017)	2.92	X	X
	PixelSNAIL (Chen et al., 2018)	**2.85**	3.80	X
	SPN (Menick & Kalchbrenner, 2018)	X	3.85	**0.61**
	IT (Parmar et al., 2018)	2.90	**3.77**	X

Table 1: **Density estimation results.** Negative test log-likelihood is measured in *bits per dimension (BPD)* - lower is better. As in previous works, only the most successful runs are reported. Circled are the best runs among non-autoregressive models and bolded are the best runs overall.
* by autoregressive we refer to the methods based on $p(X) = \prod_{i=1} p(X_i|X_1, ..., X_{i-1})$ factorization.

Figure 5: **Linear interpolation between test images.** Given a pair of images from the test dataset (on the far right and far left), linear interpolation was performed on the layer 5 at a temperature of 0.9 (the sequence above) and on the layer 4 at a temperature of 1.0 (the sequence below).

Image synthesis. We additionally inspected the trained $p_\theta(X)$ by: *(a)* unconditionally generating samples from the prior, at different temperatures (Figure 6 left); *(b)* sampling in the neighborhood of an image not seen during training time (Figure 6 right); *(c)* interpolating between two images not seen during training time (Figure 5). All technical details on these experiments are in Appendix A.1. Additional images are given in Appendix A.4.

Discussion. Our results clearly suggest that SDN-VAE is not only superior to the convolutional network baseline, but also to all previous related approaches for non-autoregressive density modeling. To further inspect whether the increase in performance is due to applying SDN or due to the implicit increase in the number of parameters, we conducted additional ablation studies (described in Appendix A.2). These results provide additional evidence that SDN is the cause of increased

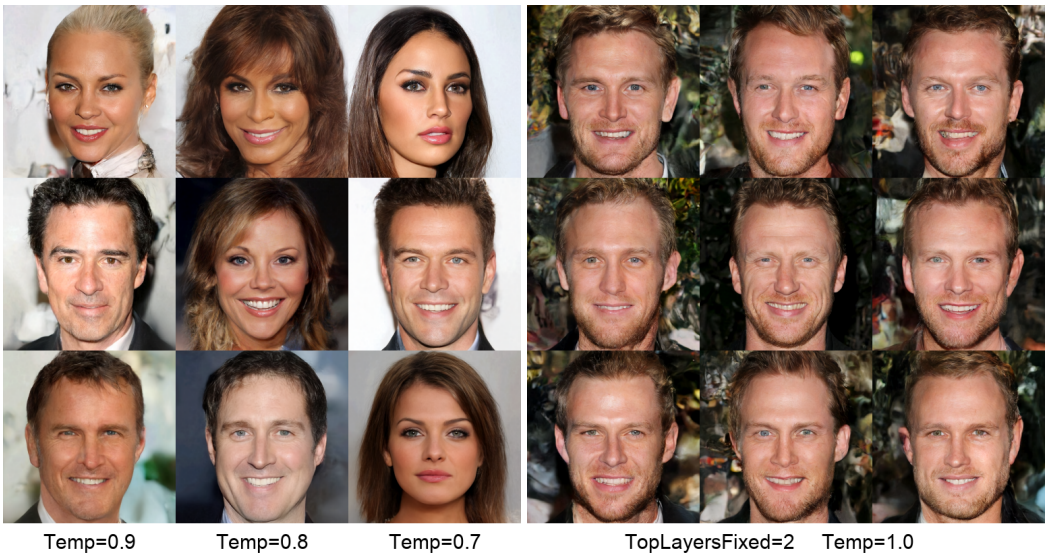


Figure 6: *(left)* **Unconditional sampling.** Samples were drawn from the SDN-VAE prior at varying temperatures; *(right)* **Sampling around a test image.** Conditioned on the top 2 layers of the latent code of the image in the center, the surrounding images were sampled at a temperature of 1.0.

performance, as even convolutional networks with more parameters, increased receptive field or more depth lead to inferior performance. In our image generation experiments we confirmed that SDN-VAE can be successfully scaled to large images. Generated samples are of exceptionally high quality. What is particularly challenging in VAE-based image generation, is sampling at high temperatures, however as shown in Figure 10 of our Appendix, SDN-VAE can successfully synthesize coherent images even in this case.

6 Learning Disentangled Representations

To study its performance impact in a more constrained setting, SDN was paired with a VAE architecturally much simpler than IAF-VAE. Apart from the implementation simplicity and shorter training time, non-hierarchical VAE is more suitable for disentangled representation learning, at least in the sense of (Higgins et al., 2016) where the aim is to decorrelate the dimensions of a latent vector. In particular, the gains in performance when using SDN were evaluated with respect to: *(a)* evidence lower bound (ELBO); *(b)* disentanglement of latent codes based on the corresponding metrics, to examine the effects of SDN decoder to the quality of learned latent representations.

Experiment setting. A relatively simple VAE architecture with a stack of convolutional networks in both the encoder and decoder (Higgins et al., 2016) was used as a baseline model. SDN decoder was constructed using a single non-residual, one-directional instance of SDN placed at the scale 32. The competing methods were evaluated on 3D-Shapes (Burgess & Kim, 2018), a synthetic dataset containing 64×64 images of scenes with rooms and objects of various colors and shapes. Following related literature (Locatello et al., 2019), there was no training-test split, so the reported ELBOs are measured on the training set. The remaining details are provided in Appendix A.5.

Training with an unmodified VAE objective. Using the original VAE objective (Kingma & Welling, 2013), competing architectures were compared in terms of ELBO (Table 2). SDN leads to substantially better estimates which come from better approximation of the conditional log-likelihood term, as shown in more detail in Figure 12 in the Appendix A.6.

VAE	KLD [BPD$\times 10^{-3}$]	**Neg. ELBO** [BPD]
CNN	7.40 ± 0.07	4.44 ± 0.04
SDN	7.95 ± 0.07	$\mathbf{2.66 \pm 0.04}$

Table 2: **Density estimation on 3D-Shapes.**

Training with β-VAE objective. The same models were trained using β-VAE objective (Higgins et al., 2016). In the β-VAE algorithm, the 'disentanglement' of latent dimensions is controlled by an additional hyperparameter denoted as β, which effectively acts as a regularizer. Modified ELBO reads as: $\mathcal{L}_{\phi,\theta}(X,\beta) = \mathbb{E}_{q_\phi(Z|X)}[\log p_\theta(X|Z)] - \beta\ \mathrm{KL}(q_\phi(Z|X)||p_0(Z))$. To investigate the effects of SDN decoder on the shaping of the latent space, we measured disentanglement for different values of β, for two popular disentanglement metrics: β-VAE (Higgins et al., 2016) and FactorVAE (Kim & Mnih, 2018). The results are shown in Figure 7.

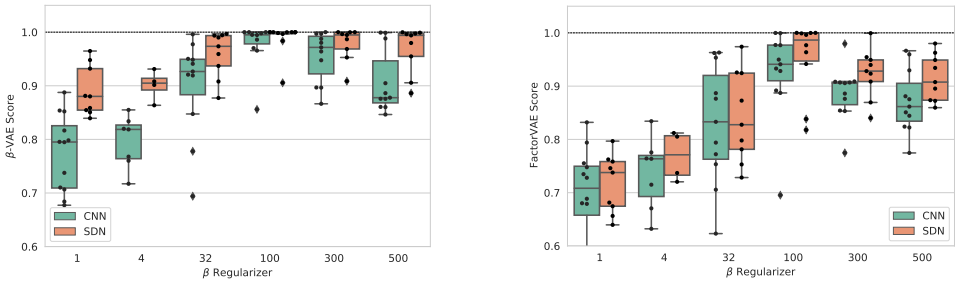


Figure 7: **β-VAE disentanglement results.** The plots compare the CNN and SDN-based VAE architectures, with respect to the β-VAE (left) and FactorVAE (right) disentanglement metrics.

Discussion. In a more constrained VAE setting, the beneficial effects of SDN to image modeling were confirmed. Perhaps surprisingly at first glance, we also found that improved VAE decoder facilitates factorization of latent codes. This may seem contradictory to the previous findings (Bowman et al., 2015; Chen et al., 2016) which suggest that a powerful decoder hurts representation learning. However, the difference here is that SDN decoders do not leverage on teacher forcing. 'Posterior collapse' can happen only very early in the training, but this is in our case easily solved using β annealing (warm-up) procedure (Bowman et al., 2015). Our results indicate the importance of good neural architectures in learning disentangled representations.

7 Conclusions

Summary. This paper introduced a novel neural layer suitable for deep neural networks that produce images – image generators. The proposed SDN is tailored to image generators that operate in a non-autoregressive way, i.e. synthesize all pixels 'at once'. SDN improves upon convolutional networks by modeling spatial coherence and long-range spatial dependencies. The main motivation behind introducing SDN is its application to VAE image decoders. SDN was analyzed in the context of: *(a)* a complex hierarchical VAE model, where the state-of-the-art performance was obtained in non-autoregressive density modeling; *(b)* a vanilla VAE, resulting in improvements in both density modeling and disentangled representation learning.

Implications. SDN is a simple-to-use module which can be easily integrated into any deep neural network, and as demonstrated in the paper, it is favorable to convolutional networks in multiple settings. SDN was shown to improve the performance of VAEs immensely, which is relevant since VAEs can be used in any density modeling task (unlike generative adversarial networks), e.g. outlier detection. VAEs are also favorable to autoregressive models in the settings where an explicit latent representation of data is needed, e.g. when the latent interpolation between test samples is necessary. We also provide an insight how VAE decoders can be changed for improved representation learning, suggesting an alternative way, and a concrete solution to tackle this important problem.

Limitations and future work. The main downside of SDN remains the computation time. However, we suspect that a more optimized implementation could substantially improve the runtime performance of SDN. In the future, it would be beneficial to explore the applicability of SDN in other settings. For example, one could apply SDN to other VAE architectures or to generative adversarial networks. SDN could also be applied to image processing tasks such as image super-resolution or image segmentation.

REFERENCES

Suzanna Becker and Geoffrey E Hinton. Self-organizing neural network that discovers surfaces in random-dot stereograms. *Nature*, 355(6356):161–163, 1992.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

Chris Burgess and Hyunjik Kim. 3d shapes dataset. https://github.com/deepmind/3dshapes-dataset/, 2018.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*, 2020.

Xi Chen, Diederik P Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. Variational lossy autoencoder. *arXiv preprint arXiv:1611.02731*, 2016.

Xi Chen, Nikhil Mishra, Mostafa Rohaninejad, and Pieter Abbeel. Pixelsnail: An improved autoregressive generative model. In *International Conference on Machine Learning*, pp. 864–872. PMLR, 2018.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Dan Cireşan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE conference on computer vision and pattern recognition*, pp. 3642–3649. IEEE, 2012.

WA Falcon. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning Cited by*, 3, 2019.

Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pp. 267–285. Springer, 1982.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. Multi-dimensional recurrent neural networks. In *International conference on artificial neural networks*, pp. 549–558. Springer, 2007.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. *arXiv preprint arXiv:1902.00275*, 2019.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Chin-Wei Huang, Laurent Dinh, and Aaron Courville. Augmented normalizing flows: Bridging the gap between generative flows and latent variable models. *arXiv preprint arXiv:2002.07101*, 2020.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

Hyunjik Kim and Andriy Mnih. Disentangling by factorising. volume 80 of *Proceedings of Machine Learning Research*, pp. 2649–2658, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL `http://proceedings.mlr.press/v80/kim18b.html`.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Diederik P Kingma and Max Welling. An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*, 2019.

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in neural information processing systems*, pp. 10215–10224, 2018.

Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in neural information processing systems*, pp. 4743–4751, 2016.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Alex Krizhevsky et al. Learning multiple layers of features from tiny images. 2009.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Raetsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. In *international conference on machine learning*, pp. 4114–4124, 2019.

Lars Maaløe, Marco Fraccaro, Valentin Liévin, and Ole Winther. Biva: A very deep hierarchy of latent variables for generative modeling. In *Advances in neural information processing systems*, pp. 6551–6562, 2019.

Jacob Menick and Nal Kalchbrenner. Generating high fidelity images with subscale pixel networks and multidimensional upscaling. *arXiv preprint arXiv:1812.01608*, 2018.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. In *International Conference on Learning Representations*, 2018.

Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 5528–5531. IEEE, 2011.

Didrik Nielsen, Priyank Jaini, Emiel Hoogeboom, Ole Winther, and Max Welling. Survae flows: Surjections to bridge the gap between vaes and flows. *arXiv preprint arXiv:2007.02731*, 2020.

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pp. 2338–2347, 2017.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. *arXiv preprint arXiv:1802.05751*, 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pp. 8026–8037, 2019.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in neural information processing systems*, pp. 901–909, 2016.

Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.

Jürgen Schmidhuber. Learning factorial codes by predictability minimization. *Neural computation*, 4(6):863–879, 1992.

Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. In *Advances in neural information processing systems*, pp. 3738–3746, 2016.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

Marijn F Stollenga, Wonmin Byeon, Marcus Liwicki, and Juergen Schmidhuber. Parallel multi-dimensional lstm, with application to fast biomedical volumetric image segmentation. In *Advances in neural information processing systems*, pp. 2998–3006, 2015.

Lucas Theis and Matthias Bethge. Generative image modeling using spatial lstms. In *Advances in Neural Information Processing Systems*, pp. 1927–1935, 2015.

Arash Vahdat and Jan Kautz. Nvae: A deep hierarchical variational autoencoder. *arXiv preprint arXiv:2007.03898*, 2020.

Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in neural information processing systems*, pp. 4790–4798, 2016.

Aaron Van Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *International Conference on Machine Learning*, pp. 1747–1756, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Francesco Visin, Kyle Kastner, Kyunghyun Cho, Matteo Matteucci, Aaron Courville, and Yoshua Bengio. Renet: A recurrent neural network based alternative to convolutional networks. *arXiv preprint arXiv:1505.00393*, 2015.

Alex Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J Lang. Phoneme recognition using time-delay neural networks. *IEEE transactions on acoustics, speech, and signal processing*, 37(3):328–339, 1989.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pp. 818–833. Springer, 2014.

Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In *International conference on machine learning*, pp. 7354–7363. PMLR, 2019.

A APPENDIX

A.1 DENSITY ESTIMATION – EXPERIMENT DETAILS

	CIFAR10	ImageNet32	CelebAHQ256
	$32^2 = 32$x32	32^2	256^2
# training samples	50K	1.281M	27K
# test samples	10K	50K	3K
Quantization	8 bits	8 bits	5 bits
Encoder/decoder depth	15	16	17
Deterministic #channels per layer	200	200	200
Stochastic #channels per layer	4	4	8
Scales	$8^2, 16^2, 32^2$	$4^2, 8^2, 16^2, 32^2$	$4^2, 8^2, 16^2, 32^2, 64^2, 128^2, 256^2$
SDN #channels per scale	120, 240, 260	424 for all	80, 160, 240, 320, 400, ~~SDN~~, ~~SDN~~
Layers per scale	5 for all	4 for all	2,2,2,2,2,3,4
Number of directions per SDN	2	3	2
Optimizer	Adamax	Adamax	Adamax
Learning rate	0.002	0.002	0.001
Learning rate annealing	Cosine	Cosine	Cosine
Batch size per GPU	32	32	4
Number of GPUs	8	8	8
GPU type	TeslaV100	TeslaV100	TeslaV100
GPU memory	32GB	32GB	32GB
Prior model	Gaussian	Gaussian	Gaussian
Posterior model	Gaussian	Gaussian	Gaussian
Posterior flow	1 IAF	1 IAF	1 IAF
Observation model	DML	DML	DML
DML Mixture components	5	10	30
Exponential Moving Average (EMA)	0.999	0.9995	0.999
Free bits	0.01	0.01	0.01
Number of importance samples	1024	1024	1024
Mixed-precision	Yes	Yes	Yes
Weight normalization	Yes	Yes	Yes
Horizontal flip data augmentation	Yes	No	Yes
Training time	45h	200h	90h

Table 3: **Experimental configurations for the density estimation tests.** DML is the discretized mixture of logistics (Salimans et al., 2017). Weight normalization, mixed-precision, free bits and Adamax are documented by Salimans & Kingma (2016); Micikevicius et al. (2018); Kingma et al. (2016); Kingma & Ba (2014) respectively. ~~SDN~~ means that SDN was not applied at the corresponding scale. The IAF (Kingma et al., 2016) contained 2 layers of masked 3x3 convolutional networks, with the context and number of CNN filters both of size 100. The importance sampling (Burda et al., 2015) was used for obtaining tighter lower bounds.

On configuring hyperparameters. Due to high computational demands, we only modestly optimized hyperparameters. In principle, our strategy was to scale up: (a) the number of deterministic and SDN filters; (b) batch size; (c) learning rate; (d) the number of SDN directions and (e) the number of DML components; as long as we found no signs of overfitting, had no memory or stability issues, and the training was not considerably slower. EMA coefficient values were taken from the previous related works (Kingma et al., 2016; Chen et al., 2018) – we tested 0.999 and 0.9995. We also swept through the values $\{2, 4, 8, 15\}$ for the number of latent stochastic filters, but saw no significant difference in the performance. Most extensively we explored the number of layers per scale, and found this to have relevant impact on runtime and over/underfitting, for both baseline and our architecture. We found that more downsampling improved runtime and reduced memory consumption, but made the tested models prone to overfitting.

A.2 Density Estimation – Ablation Studies

Additional ablation studies on SDN-VAE were conducted in order to explore in more detail whether the good performance on the density estimation experiments (Table 1) indeed comes from the proposed SDN layer. We used CIFAR-10 data on which both baseline and SDN-VAE models converged faster in comparison to two other data sets. Our main motivation was to understand whether the increase in performance comes from our architectural decisions or simply from the increase in the number of parameters. Namely, since ResSDN (from Figure 3) is a network with a larger number of parameters in comparison to the baseline 3x3 convolutional network, it would be justified to question if the increase in performance can be indeed attributed to ResSDN layers.

To that end, we designed multiple baseline blocks to replace CNN in the IAF-VAE+ architecture, by replicating the design protocol from Section 4 in which ResSDN was applied to create the SDN-VAE architecture. Baseline blocks are illustrated in Figure 8. The main idea is to explore whether possibly increased kernel of a CNN would be sufficient to model spatial dependencies, or whether a deeper network can bring the same performance to the basic VAE architecture.

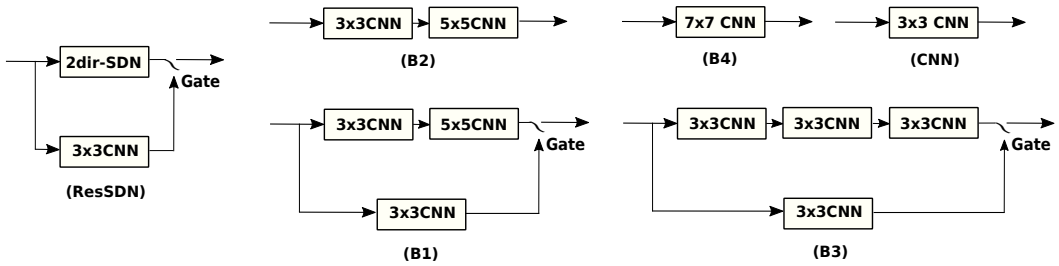


Figure 8: **ResSDN and baseline blocks.**

All baseline blocks were trained in the same setting as SDN-VAE i.e. all the parameters were kept fixed, except from the learning rate, which we halved after every 10 unsuccessful attempts to stabilize the training. IAF-VAE based on the CNN block was stable at the default learning rate of 0.002, B2 was stable at the learning rate of 0.001, while B4 and B1 were stable only at 0.0005. B2 was unstable. We also tested ResSDN at a learning rate of 0.001 to understand if there would be a drop in performance, but there wasn't any except from marginally slower convergence time.

The best runs in terms of negative evidence lower bound (ELBO) for each of the baseline blocks, along with our most successful run for SDN-VAE architecture are reported in Table 4. What we can observe is that the increase in capacity can indeed be correlated with good performance. However, even those VAE models which contained more parameters than the proposed SDN-VAE architecture were not able to converge to a higher negative ELBO. In fact, there exist a considerable performance gap between baseline blocks and the proposed ResSDN layer.

	Layer	Number of VAE parameters in millions	Time to converge in hours	Time to converge in iterations	**Best Neg. ELBO** in BPD
Baselines	CNN	42M	17h	140K	3.081
	B1	192M	21h	75K	3.080
	B2	104M		unstable	
	B3	126M	23h	90K	2.945
	B4	130M	22h	124K	3.120
Ours	ResSDN	121M	45h	60K	2.906

Table 4: **The comparison of ResSDN to the baselines from Figure 8.** We replaced CNN in the IAF-VAE+ architecture (from Section 4) with baseline blocks and compared the density estimation performance in terms of ELBO. The experiments were conducted on CIFAR-10 dataset.

A.3 Additional SDN Analysis

Number of parameters. For a filter size of 200 (a value used in the SDN-VAE experiments), we compare the number of parameters between CNN and SDN layers. Note that the input scale does not have any effect on the resulting numbers. The numbers are given in Table 5. Here 'dir' denotes the directions in SDN. 'Project phase' denotes the size of project-in and project-out SDN sub-layers which are in this experiment of the same size, since no upsampling is performed. We can observe that the 2-directional SDN is approximately of the same size as 5x5CNN in terms of number of free parameters.

	3x3CNN	5x5CNN	Project phase	SDN cell	1dir-SDN	2dir-SDN
# parameters	360200	1000200	40200	481200	561600	1042800

Table 5: **Number of parameters.**

Runtime unit tests. We measured the execution times of forward propagation of CNN and SDN, for different input scales. To obtain standard deviation estimates, each forward propagation was repeated 100 times. The batch size was set to 128. The results are given in Table 6. SDN is considerably slower than CNN, but in the reasonable limits. Note that more efficient implementation will likely improve SDN runtime performance.

Input scale vs. Layer	3x3CNN	5x5CNN	1dir-SDN	2dir-SDN
4	0.32±0.01	1.26±0.01	1.93±0.06	3.52±0.07
8	0.63±0.01	3.25±0.13	4.09±0.06	7.61±0.09
16	2.33±0.08	11.2±0.08	11.4±0.03	21.0±0.05
32	9.18±0.16	20.4±0.18	45.4±0.09	83.8±0.15

Table 6: **Runtime unit tests.** The execution time of forward propagation in ms.

A.4 IMAGE SYNTHESIS – ADDITIONAL RESULTS

Figure 9: **MSE-based nearest neighbors.** On the left, shown are images from Figure 6. On the right, shown are nearest neighbors in terms of mean squarred error (MSE), found in the training data set. There are no signs that generated samples are replicas of the training ones.

Figure 10: **Additional samples synthesized at different temperatures.** As we decrease the temperature, getting closer to the mean of the prior, the photographs become smoother i.e. more 'generic'.

Figure 11: **Sampling around a test image, for varying temperatures and number of fixed layers.** Presented is a 3x3 grid. In each grid there is a 3x3 sub-grid in which an image in the center is encoded, and the surrounding images are sampled conditioned on the specified number of layers from the encoded image and at the specified temperature. The text on the left hand side denotes the number of layers taken from an encoded image to condition sampling on. The text on the bottom denotes the temperature at which samples were drawn. Two key observations can be made: *(a)* lowering the temperature decreases the level of details, making photographs smoother; *(b)* Most of the information in SDN-VAE is encoded in the topmost layer. As we go further down the chain of the generator network, there is a gradual decrease in information content in latent stochastic variables.

A.5 LEARNING DISENTANGLED REPRESENTATIONS – EXPERIMENT DETAILS

The hyperparameter configuration is given in Table 7. In principle, we followed Higgins et al. (2016); Locatello et al. (2019) in configuring parameters. Notable changes include using of discretized logistics (Kingma et al., 2016) as the observation model and the Adamax optimizer. We also applied β-VAE annealing to avoid posterior collapse early in the training, an issue for both considered architectures. The base architecture was again taken from the same previous works. The exact description of architectures is given in Table 8 for the CNN-based VAE, and in Table 9 for the SDN-based VAE.

	3D Shapes $64^2 = 64 \times 64$
# data samples	448K
SDN #channels	200
Number of directions per SDN	1
Optimizer	Adamax
Learning rate	0.001
Batch size	128
Total number of training iterations	200k
Prior and posterior models	Gaussian
Observation model	Discretized Logistics

Table 7: **Configuration of disentangled representation learning experiments.**

Encoder	**Decoder**
4×4 conv, 32 ReLU, stride 2	FC, 256 ReLU
4×4 conv, 32 ReLU, stride 2	FC, $4 \times 4 \times 256$ ReLU
4×4 conv, 64 ReLU, stride 2	4×4 upconv, 64 ReLU, stride 2
4×4 conv, 64 ReLU, stride 2	4×4 upconv, 32 ReLU, stride 2
FC 256	4×4 upconv, 32 ReLU, stride 2
FC 2×10	4×4 upconv, 3 (number of channels) , stride 2

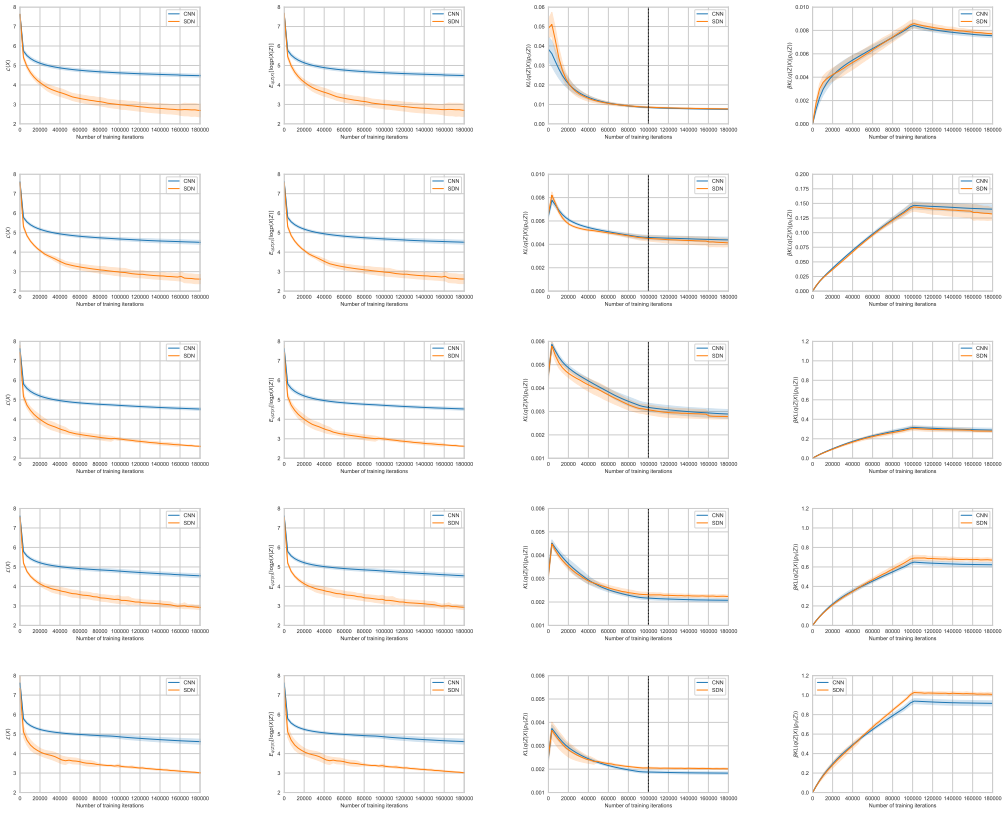
Table 8: **CNN-based vanilla VAE architecture.**

Encoder	**Decoder**
4×4 conv, 32 ReLU, stride 2	FC, 256 ReLU
4×4 conv, 32 ReLU, stride 2	FC, $4 \times 4 \times 256$ ReLU
4×4 conv, 64 ReLU, stride 2	4×4 upconv, 64 ReLU, stride 2
4×4 conv, 64 ReLU, stride 2	4×4 upconv, 32 ReLU, stride 2
FC 256	1dir-SDN with 200 channels
FC 2×10	4×4 upconv, 3 (number of channels) , stride 2

Table 9: **SDN-based vanilla VAE architecture.**

A.6 LEARNING DISENTANGLED REPRESENTATIONS – ADDITIONAL RESULTS

For the sake of completeness, we also provide learning curves for varying values of β and for different individual terms of β-VAE objective function. The plots are shown in Figure 12.



(a) (b) (c) (d)

Figure 12: **β-VAE learning curves for CNN and SDN-based VAE architectures.** Presented are the following quantities measured over the course of training: *(a)* evidence lower bound (ELBO); *(b)* conditional log-likelihood (reconstruction) term; *(c)* KL divergence term; *(d)* β-scaled KL divergence term; Top-to-bottom, the rows are related to $\beta = \{1, 32, 100, 300, 500\}$. Each pair ($\beta$ value, VAE architecture) is trained for 10 different random seeds. Linear β-annealing procedure was performed, where β was increased from 0 to its final value across the span of 100K training iterations (the end of the annealing procedure is denoted by the vertical line in the plots of the column (c)).