Journal of Machine Learning Research 23 (2022) 1-51 Submitted 3/20; Revised 8/21; Published 1/22

# Evolutionary Variational Optimization of Generative Models

**Jakob Drefs** JAKOB.DREFS@UOL.DE
*Machine Learning Lab*
*University of Oldenburg*
*26129 Oldenburg, Germany*

**Enrico Guiraud** ENRICO.GUIRAUD@CERN.CH
*CERN*
*1211 Geneva, Switzerland,*
*and Machine Learning Lab*
*University of Oldenburg*
*26129 Oldenburg, Germany*

**Jörg Lücke** JOERG.LUECKE@UOL.DE
*Machine Learning Lab*
*University of Oldenburg*
*26129 Oldenburg, Germany*

**Editor:** Frank Wood

## Abstract

We combine two popular optimization approaches to derive learning algorithms for generative models: variational optimization and evolutionary algorithms. The combination is realized for generative models with discrete latents by using truncated posteriors as the family of variational distributions. The variational parameters of truncated posteriors are sets of latent states. By interpreting these states as genomes of individuals and by using the variational lower bound to define a fitness, we can apply evolutionary algorithms to realize the variational loop. The used variational distributions are very flexible and we show that evolutionary algorithms can effectively and efficiently optimize the variational bound. Furthermore, the variational loop is generally applicable ("black box") with no analytical derivations required. To show general applicability, we apply the approach to three generative models (we use Noisy-OR Bayes Nets, Binary Sparse Coding, and Spike-and-Slab Sparse Coding). To demonstrate effectiveness and efficiency of the novel variational approach, we use the standard competitive benchmarks of image denoising and inpainting. The benchmarks allow quantitative comparisons to a wide range of methods including probabilistic approaches, deep deterministic and generative networks, and non-local image processing methods. In the category of "zero-shot" learning (when only the corrupted image is used for training), we observed the evolutionary variational algorithm to significantly improve the state-of-the-art in many benchmark settings. For one well-known inpainting benchmark, we also observed state-of-the-art performance across all categories of algorithms although we only train on the corrupted image. In general, our investigations highlight the importance of research on optimization methods for generative models to achieve performance improvements.

**Keywords:** Expectation Maximization, Variational Methods, Evolutionary Algorithms, Sparse Coding, Denoising, Inpainting

## 1. Introduction

Variational approximations (Saul et al., 1996; Saul and Jordan, 1995; Neal and Hinton, 1998; Jordan et al., 1999, and many more) are popular and successful approaches to efficiently train probabilistic generative models. In the common case when inference based on a generative data model is not tractable or not scalable to desired problem sizes, variational approaches provide approximations for an efficient optimization of model parameters. Here we focus on variational expectation maximization (variational EM) to derive learning algorithms (Neal and Hinton, 1998; Jordan et al., 1999), while acknowledging that variational approaches are also well suited and successful in the context of fully Bayesian (non-parametric) approaches (e.g. Ghahramani and Jordan, 1995; Zhou et al., 2009). Variational EM typically seeks to approximate intractable full posterior distributions by members of a specific family of variational distributions. Prominent such families are the family of Gaussian distributions (e.g., Opper and Winther, 2005; Rezende et al., 2014; Kingma and Welling, 2014) and the family of factored distributions (e.g., Jordan et al., 1999), with the latter often being referred to as *mean field* approaches. Variational approaches can, in general, be regarded as one of two large classes of approximations with the other being made up of sampling approximations. Variational training can be very efficient and has been shown to realize training of very large-scale models (Blei et al., 2003; Rezende et al., 2014; Kingma and Welling, 2014) with thousands or even millions (Sheikh and Lücke, 2016) of parameters. In terms of efficiency and scalability, they are often preferred to sampling approaches (see, e.g., Angelino et al., 2016, Section 6, for a discussion), although many successful algorithms often combine both variational and sampling techniques (e.g., Dayan et al., 1995; Shelton et al., 2011; Rezende et al., 2014; Kingma and Welling, 2014).

One drawback of variational approaches lies in their limits in modeling the posterior structure, which can result in biases introduced into the learned parameters. Gaussian variational distributions, by definition, only capture one posterior mode, for instance, and fully factored approaches (i.e., mean field) do not capture explaining-away effects including posterior correlations. Gaussian variational EM is consequently particularly popular for generative models known to result in posteriors with essentially one mode (Opper and Winther, 2005; Seeger, 2008; Opper and Archambeau, 2009). Mean field approaches are more broadly applied but the deteriorating effects of assuming a-posteriori independence have repeatedly been pointed out and discussed in different contexts (e.g., Mackay, 2001; Ilin and Valpola, 2005; Turner and Sahani, 2011; Sheikh et al., 2014; Vértes and Sahani, 2018). A further drawback of variational approaches, which they share with most sampling approaches, is the challenge of analytically deriving an appropriate approximation for any new generative model. While solutions specific to a generative model are often the best choice w.r.t. efficiency, it has been argued (see, e.g. Ranganath et al., 2014; Steinruecken et al., 2019) that the necessary expertise to derive such solutions is significantly taxing the application of variational approaches in practice.

The drawbacks of variational approaches have motivated novel research directions. Early on, mean field approaches were generalized, for instance, to contain mutual dependencies (e.g., structured mean field; Saul and Jordan, 1995; Bouchard-Côté and Jordan, 2009; Murphy, 2012; MacKay, 2003) and, more recently, methods to construct arbitrarily complex approximations to posteriors, e.g., using normalizing flows, were suggested (Rezende and Mohamed, 2015;

Kingma et al., 2016). Ideally, a variational approximation should be both very efficiently scalable as well as generally applicable. Also, with variational inference becoming increasingly popular for training deep unsupervised models (e.g., Rezende et al., 2014; Kingma et al., 2016), the significance of fast and flexible variational methods has further increased. Not surprisingly, however, increasing both scalability and generality represents a major challenge because, in general, a trade-off can be observed between the flexibility of the used variational method on the one hand, and its scalability and task performance on the other.

In order to contribute to novel methods that are both flexible and scalable, we consider here generative models with discrete latents and explore the combination of variational and evolutionary optimization. For our purposes, we use truncated posteriors as a family of variational distributions. In terms of scalability, truncated posteriors suggest themselves as their application has enabled training of very large generative models (Sheikh and Lücke, 2016; Forster and Lücke, 2018; Hirschberger et al., 2022). Furthermore, the family of truncated posteriors is directly defined by the joint distribution of a given generative model, which allows for algorithms that are generally applicable to generative models with discrete latents. While truncated posteriors have been used and evaluated in a series of previous studies (e.g. Lücke and Eggert, 2010; Dai and Lücke, 2014; Shelton et al., 2017), they have previously not been optimized variationally, i.e., rather than seeking the optimal members of the variational family, truncations were estimated by one-step feed-forward functions. Instead, we use here a fully variational optimization loop which improves the variational bound by using evolutionary algorithms. For mixture models, fully variational approaches based on truncated posteriors have recently been suggested (Forster et al., 2018; Hirschberger et al., 2022), but they exploit the non-combinatorial nature of mixtures to derive speed-ups for large scales (also compare Hughes and Sudderth, 2016). Here we, for the first time, apply a fully variational approach based on truncated posteriors in order to optimize more complex generative models (see Lücke et al., 2018 and Guiraud et al., 2018 for preliminary results).

## 2. Evolutionary Variational Optimization

A probabilistic generative model stochastically generates data points (here referred to as $\vec{y}$) using a set of hidden (or latent) variables (referred to as $\vec{s}$). The generative process can be formally defined by a joint probability $p(\vec{s}, \vec{y} \mid \Theta)$, where $\Theta$ is the set of all model parameters. We will introduce concrete models in Section 3. To start, let us consider general models with binary latents. In such case $p(\vec{y} \mid \Theta) = \sum_{\vec{s}} p(\vec{s}, \vec{y} \mid \Theta)$, where the sum is taken over all possible configurations of the latent variable $\vec{s} \in \{0,1\}^H$ with $H$ denoting the length of the latent vector. Given a set of $N$ data points $\mathcal{Y} = \{\vec{y}^{(n)}\}_{n=1,\ldots,N}$, we seek parameters $\Theta$ that maximize the data likelihood $\mathcal{L}(\Theta) = \prod_{n=1}^{N} p(\vec{y}^{(n)} \mid \Theta)$. Here we use an approach based on Expectation Maximization (EM; e.g. Gupta and Chen, 2011, for a review). Instead of maximizing the (log-)likelihood directly, we follow, e.g., Saul and Jordan (1995) and Neal and Hinton (1998), and iteratively increase a variational lower bound (referred to as *free energy* or *ELBO*), which is given by:

$$\mathcal{F}(q, \Theta) = \sum_{n=1}^{N} \sum_{\vec{s}} q^{(n)}(\vec{s}) \, \log\big(p(\vec{y}^{(n)}, \vec{s} \mid \Theta)\big) \, + \sum_{n=1}^{N} H[q^{(n)}]. \tag{1}$$

$q^{(n)}(\vec{s})$ are variational distributions and $H[q^{(n)}] = -\sum_{\vec{s}} q^{(n)}(\vec{s}) \log\big(q^{(n)}(\vec{s})\big)$ denotes the entropy of these distributions. We seek distributions $q^{(n)}(\vec{s})$ that approximate the intractable posterior distributions $p(\vec{s} \mid \vec{y}^{(n)}, \Theta)$ as well as possible and that, at the same time, result in tractable parameter updates. If we denote the parameters of the variational distributions by $\Lambda$, then a variational EM algorithm consists of iteratively maximizing $\mathcal{F}(\Lambda, \Theta)$ w.r.t. $\Lambda$ in the variational E-step and w.r.t. $\Theta$ in the M-step. In this respect, the M-step can maintain the same functional form as for exact EM but expectation values now have to be computed with respect to the variational distributions.

## 2.1 Evolutionary Optimization of Truncated Posteriors

Instead of using specific functional forms such as Gaussians or factored (mean field) distributions for $q^{(n)}(\vec{s})$, we choose, for our purposes, truncated posterior distributions (see, e.g., Lücke and Eggert, 2010; Sheikh et al., 2014; Shelton et al., 2017; Lücke and Forster, 2019; Hirschberger et al., 2022):

$$q^{(n)}(\vec{s} \mid \mathcal{K}^{(n)}, \hat{\Theta}) := \frac{p\left(\vec{s} \mid \vec{y}^{(n)}, \hat{\Theta}\right)}{\sum\limits_{\vec{s}\,' \in \mathcal{K}^{(n)}} p\left(\vec{s}\,' \mid \vec{y}^{(n)}, \hat{\Theta}\right)} \delta(\vec{s} \in \mathcal{K}^{(n)}) = \frac{p\left(\vec{s}, \vec{y}^{(n)} \mid \hat{\Theta}\right)}{\sum\limits_{\vec{s}\,' \in \mathcal{K}^{(n)}} p\left(\vec{s}\,', \vec{y}^{(n)} \mid \hat{\Theta}\right)} \delta(\vec{s} \in \mathcal{K}^{(n)}), \tag{2}$$

where $\delta(\vec{s} \in \mathcal{K}^{(n)})$ is equal to 1 if a state $\vec{s}$ is contained in the set $\mathcal{K}^{(n)}$ and 0 otherwise. The variational parameters are now given by $\Lambda = (\mathcal{K}, \hat{\Theta})$ where $\mathcal{K} = \{\mathcal{K}^{(n)}\}_{n=1,\ldots,N}$. With this choice of variational distributions, expectations w.r.t. the full posterior can be approximated by efficiently computable expectations w.r.t. truncated posteriors (2):

$$\langle g(\vec{s}) \rangle_{q^{(n)}} = \frac{\sum\limits_{\vec{s} \in \mathcal{K}^{(n)}} g(\vec{s})\ p(\vec{s}, \vec{y}^{(n)} \mid \hat{\Theta})}{\sum\limits_{\vec{s}\,' \in \mathcal{K}^{(n)}} p(\vec{s}\,', \vec{y}^{(n)} \mid \hat{\Theta})}. \tag{3}$$

Instead of estimating the relevant states of $\mathcal{K}^{(n)}$ using feedforward (preselection) functions (e.g., Lücke and Eggert, 2010; Sheikh et al., 2014, 2019), we here apply a fully variational approach, i.e., we define a variational loop to optimize the variational parameters. Based on the specific functional form of truncated posteriors, optimal variational parameters $\Lambda = (\mathcal{K}, \hat{\Theta})$ are given by setting $\hat{\Theta} = \Theta$ and by seeking $\mathcal{K}$ which optimize (see Lücke, 2019, for details):

$$\mathcal{F}(\mathcal{K}^{(1\ldots N)}, \Theta) = \sum_{n=1}^{N} \log\Big( \sum_{\vec{s} \in \mathcal{K}^{(n)}} p\left(\vec{y}^{(n)}, \vec{s} \mid \Theta\right) \Big). \tag{4}$$

Equation 4 represents a reformulation of the variational lower bound (1) for $\hat{\Theta} = \Theta$. Because of the specific functional form of truncated posteriors, this reformulation does not contain an explicit entropy term, which allows for an efficient optimization using pairwise comparisons. More concretely, the variational bound is provably increased in the variational E-step if the sets $\mathcal{K}^{(n)}$ are updated by replacing a state $\vec{s}$ in $\mathcal{K}^{(n)}$ with a new state $\vec{s}^{\,\mathrm{new}}$ such that:

$$p(\vec{s}^{\,\mathrm{new}}, \vec{y}^{(n)} \mid \Theta)\ >\ p(\vec{s}, \vec{y}^{(n)} \mid \Theta)\,, \tag{5}$$

where we make sure that any new state $\vec{s}^{\,\mathrm{new}}$ is not already contained in $\mathcal{K}^{(n)}$. By successively replacing states (or bunches of states), we can keep the size of each set $\mathcal{K}^{(n)}$ constant. We use the same constant size $S$ for all sets (i.e., $|\mathcal{K}^{(n)}| = S$ for all $n$), which makes $S$ an important parameter for the approximation. The crucial remaining question is how new states can be found such that the lower bound (4) is increased efficiently.

A distinguishing feature when using the family of truncated posteriors as variational distributions is the type of variational parameters, which are given by sets of latent states (i.e. by the sets $\mathcal{K}^{(n)}$). As these states, for binary latents, are given by bit vectors ($\vec{s} \in \{0,1\}^H$), we can here interpret them as genomes of individuals. Evolutionary algorithms (EAs) then emerge as a very natural choice: we can use EAs to mutate and select the bit vectors $\vec{s} \in \{0,1\}^H$ of the sets $\mathcal{K}^{(n)}$ in order to maximize the lower bound (4). In the EA terminology, the variational parameters $\mathcal{K}^{(n)}$ then become *populations* of individuals, where each *individual* is defined by its latent state $\vec{s} \in \{0,1\}^H$ (in the following, we will use *individual* to also refer to its genome/latent state).

For each population $\mathcal{K}^{(n)}$, we will use EAs with standard genetic operators (parent selection, mutation and crossover) in order to produce offspring, and we will then use the offspring in order to improve each population. The central function for an EA is the fitness function it seeks to optimize. For our purposes, we will define the fitness $f(\vec{s}; \vec{y}^{\,(n)}, \Theta)$ of the individuals $\vec{s}$ to be a monotonic function of the joint $p(\vec{s}, \vec{y}^{\,(n)} \mid \Theta)$ of the given generative model, i.e., we define the fitness function to satisfy:

$$f(\vec{s}^{\,\mathrm{new}}; \vec{y}^{\,(n)}, \Theta) \; > \; f(\vec{s}; \vec{y}^{\,(n)}, \Theta) \quad \Leftrightarrow \quad p(\vec{s}^{\,\mathrm{new}}, \vec{y}^{\,(n)} \mid \Theta) \; > \; p(\vec{s}, \vec{y}^{\,(n)} \mid \Theta)\,. \tag{6}$$

A fitness satisfying (6) will enable the selection of parents that are likely to produce offspring with high joint probability $p(\vec{s}^{\,\mathrm{new}}, \vec{y}^{\,(n)} \mid \Theta)$. Before we detail how the offspring is used to improve a population $\mathcal{K}^{(n)}$, we first describe how the EA generates offspring. For each population $\mathcal{K}^{(n)}$, we set $\mathcal{K}^{(n)}_0 = \mathcal{K}^{(n)}$ and then iteratively generate new generations $\mathcal{K}^{(n)}_g$ by successive application of Parent Selection, Crossover, and Mutation operators:

*Parent Selection.* To generate offspring, $N_p$ parent states are first selected from the initial population $\mathcal{K}^{(n)}_0$ (see Figure 1). Ideally, the parent selection procedure should be balanced between exploitation of parents with high fitness (which might be expected to be more likely to produce children with high fitness) and exploration of mutations of poor performing parents (which might be expected to eventually produce children with high fitness while increasing population diversity). Diversity is crucial, as the $\mathcal{K}^{(n)}$ are sets of unique individuals and therefore the improvement of the overall fitness of the population depends on generating as many as possible *different* children with high fitness. In our numerical experiments, we explore both random uniform selection of parents and fitness-proportional selection of parents (i.e., parents are selected with a probability proportional to their fitness). For the latter case, we define fitness here as follows:

$$f(\vec{s}; \vec{y}^{\,(n)}, \Theta) := \widetilde{\log p}(\vec{s}, \vec{y}^{\,(n)} \mid \Theta) + \mathrm{const.} \tag{7}$$

The term $\widetilde{\log p}(\vec{s}, \vec{y}^{\,(n)} \mid \Theta)$ denotes a monotonously increasing function of the joint $p(\vec{s}, \vec{y}^{\,(n)} \mid \Theta)$ which is more efficiently computable and which has better numerical stability than the joint itself. $\widetilde{\log} p(\vec{s}, \vec{y}^{\,(n)} \mid \Theta)$ is defined as the logarithm of the joint where summands that do not depend on $\vec{s}$ have been elided (see Appendix A for examples). As the fitness (7) satisfies (6), the parents are likely to have high joint probabilities if they are selected proportionally to $f(\vec{s}; \vec{y}^{\,(n)}, \Theta)$. The constant term in (7) is introduced to ensure that the fitness function always takes positive values for fitness proportional selection. The constant is defined to be const $= \left|2 \min_{\vec{s} \in \mathcal{K}_g^{(n)}} \widetilde{\log} p(\vec{s}, \vec{y}^{\,(n)} \mid \Theta)\right|$. For a given population of individuals $\mathcal{K}_g^{(n)}$, the value of const is thus constant throughout the generation of the next population $\mathcal{K}_{g+1}^{(n)}$. According to (7), the fitness function can be evaluated efficiently given that the joint $p(\vec{s}, \vec{y}^{\,(n)} \mid \Theta)$ can efficiently be computed (which we will assume here).

*Crossover.* For the crossover step, all the parent states are paired in each possible way. Each pair is then assigned a number $c$ from 1 to $H - 1$ with uniform probability ($c$ defines the single crossover point). Finally, each pair swaps the last $H - c$ bits to generate offspring. The procedure generates $N_c = N_p(N_p - 1)$ children. The crossover step can be skipped, making the EA more lightweight but decreasing variety in the offspring.

*Mutation.* Finally, each child undergoes one random bitflip to further increase offspring diversity. In our experiments, we compare results of random uniform selection of the bits to flip with a more refined sparsity-driven bitflip algorithm (the latter scheme assigns to 0's and 1's different probabilities of being flipped in order to produce children with a sparsity compatible with the one learned by a given model; details in Appendix B). If the crossover step is skipped, the parent states are copied $N_m$ times ($N_m$ can be chosen between $1 \leq N_m \leq H$) and offspring is produced by mutating each copy by one random bitflip, resulting in $N_c = N_p N_m$ children.

The application of the genetic operators is iteratively repeated. The offspring $\mathcal{K}_{g+1}^{(n)}$ produced by the population $\mathcal{K}_g^{(n)}$ becomes the new population from which parents are selected (Figure 1). After $N_g$ iterations, we obtain a large number of new individuals (i.e., all the newly generated populations $\mathcal{K}_g^{(n)}$ with $g = 1, \ldots, N_g$). The new populations of individuals can now be used to improve the original population $\mathcal{K}^{(n)}$ such that a higher value for the lower bound (4) is obtained. To find the best new population for a given $n$, we collect all generated populations (all $\mathcal{K}_g^{(n)}$ with $g = 1, \ldots, N_g$) and unite them with $\mathcal{K}^{(n)}$. We then reduce this larger set back to the population size $S$, and we do so by only keeping those $S$ individuals with the highest[1] joints $p(\vec{s}, \vec{y}^{\,(n)} \mid \Theta)$. By only keeping the states/individuals with highest joints, the update procedure for $\mathcal{K}^{(n)}$ is guaranteed to monotonically increase the variational lower bound (4) since the variational bound is increased if criterion (5) is satisfied.

Algorithm 1 summarizes the complete variational algorithm; an illustration is provided in Figure 1. As the variational E-step is realized using an evolutionary algorithm, Algorithm 1 will from now on be referred to as *Evolutionary Variational Optimization* (EVO). The EVO algorithm can be trivially parallelized over data-points. For later numerical experiments, we will use a parallelized implementation of EVO that can be executed on several hundreds of

1. Instead of selecting the $S$ elements with largest joints, we in practice remove the $(|\mathcal{K}^{(n)}| - S)$ elements of $\mathcal{K}^{(n)}$ with the lowest joint probabilities. Both procedures are equivalent.

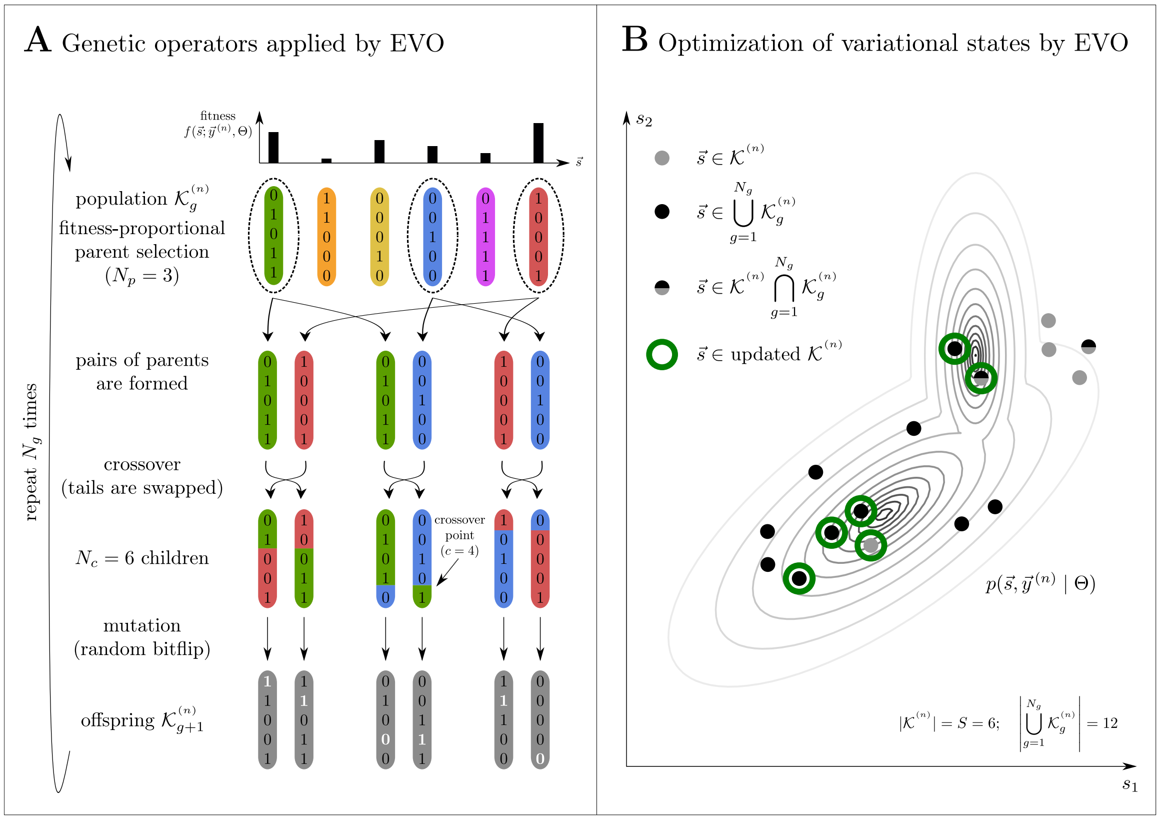

A Genetic operators applied by EVO
fitness $f(\vec{s};\vec{y}^{(n)},\Theta)$
$\vec{s}$
population $\mathcal{K}_g^{(n)}$
fitness-proportional parent selection ($N_p = 3$)
pairs of parents are formed
crossover (tails are swapped)
$N_c = 6$ children
crossover point ($c = 4$)
mutation (random bitflip)
offspring $\mathcal{K}_{g+1}^{(n)}$
repeat $N_g$ times
B Optimization of variational states by EVO
$s_2$
$s_1$
$\vec{s} \in \mathcal{K}^{(n)}$
$\vec{s} \in \bigcup_{g=1}^{N_g} \mathcal{K}_g^{(n)}$
$\vec{s} \in \mathcal{K}^{(n)} \bigcap_{g=1}^{N_g} \mathcal{K}_g^{(n)}$
$\vec{s} \in$ updated $\mathcal{K}^{(n)}$
$p(\vec{s},\vec{y}^{(n)} \mid \Theta)$
$|\mathcal{K}^{(n)}| = S = 6;\quad \left|\bigcup_{g=1}^{N_g} \mathcal{K}_g^{(n)}\right| = 12$

Figure 1: Illustration of the variational optimization using genetic operators.

computing cores (for further technical details see Appendix F.1 and also compare Salimans et al., 2017)[2]. For the numerical experiments, we will indicate which parent selection procedure ("fitparents" for fitness-proportional selection, "randparents" for random uniform selection) and which bitflip algorithm ("sparseflips" or "randflips") are used in the variational loop. The term "cross" will be added to the name of the EA when crossover is employed.

## 2.2 Related Work on Applications of Evolutionary Approaches to Learning

Evolutionary algorithms (EAs) have repeatedly been applied to learning algorithms (e.g. Goldberg, 1989). In the context of reinforcement learning, for instance, evolution strategies have successfully been used as an optimization alternative to Q-learning and policy gradients (Salimans et al., 2017). EAs have also previously been applied to unsupervised learning based on generative models, also in conjunction with (or as substitute for) expectation maximization (EM). Myers et al. (1999) used EAs to learn the structue of Bayes Nets, for instance. Pernkopf and Bouchaffra (2005) have used EAs for clustering based on Gaussian mixture models (GMMs), where GMM parameters are updated relatively conventionally using EM; EAs are used to select the best GMM models for the clustering problem (using a minimum description length criterion). Work by Tohka et al. (2007) uses EAs for a more elaborate initialization of Finite Mixture Models, which is followed by EM. Work by (Tian et al., 2011) goes further and combines EAs with a variational Bayes approach for

2. The source code can be accessed via `https://github.com/tvlearn`.

**Algorithm 1:** Evolutionary Variational Optimization (EVO).

```
define selection, crossover and mutation operators;
set hyperparameters $S$, $N_g$, etc.;
initialize model parameters $\Theta$;
for each $n$: populate set $\mathcal{K}^{(n)}$ with $S$ distinct latent states ($|\mathcal{K}^{(n)}| = S$);
repeat
    for $n = 1, \ldots, N$ do
        set $\mathcal{K}_0^{(n)} = \mathcal{K}^{(n)}$;
        for $g = 1, \ldots, N_g$ do
            $\mathcal{K}_g^{(n)} = \mathrm{mutation}\Big(\mathrm{crossover}\Big(\mathrm{selection}\Big(\mathcal{K}_{g-1}^{(n)}\Big)\Big)\Big)$;
            $\mathcal{K}^{(n)} = \mathcal{K}^{(n)} \cup \mathcal{K}_g^{(n)}$;
        remove those ($|\mathcal{K}^{(n)}| - S$) elements $\vec{s}$ in $\mathcal{K}^{(n)}$ with lowest $p(\vec{s}, \vec{y}^{(n)} \mid \Theta)$;
    use M-steps with (3) to update $\Theta$;
until parameters $\Theta$ have sufficiently converged;
```

GMMs. The algorithm they suggested is based on a fully Bayesian GMM approach with hyperpriors in the place of standard GMM parameters. The algorithm first uses an EA to find good hyperparameters; given the hyperparameters, the GMM is then trained using standard (but variational) EM iterations. The use of EAs by Pernkopf and Bouchaffra (2005) and Tian et al. (2011) is similar to their use for deep neural network (DNN) training. For DNNs, EAs are applied to optimize network hyperparameters in an outer loop (Stanley and Miikkulainen, 2002; Loshchilov and Hutter, 2016; Real et al., 2017; Suganuma et al., 2017; Oehmcke and Kramer, 2018, etc.) while DNN weight parameters are at the same time trained using standard back-propagation algorithms. Yet other approaches have applied EAs to directly optimize a clustering objective, using EAs to *replace* EM approaches for optimization (compare Hruschka et al., 2009). Similarly, in a non-probabilistic setting, EAs have been used to replace back-propagation for DNN training (see, e.g., Prellberg and Kramer, 2018, and references therein). Also in a non-probabilistic setting, for sparse coding with weights optimized based on a $l_1$-penalized reconstruction objective (e.g. Olshausen and Field, 1996), an EA tailored to this objective has been used to address maximum a-posteriori optimization (Ahmadi and Salari, 2016). In contrast to all such previous applications, the EVO approach (Algorithm 1) applies EAs as an integral part of variational EM, i.e., EAs address the key optimization problem of variational EM in the inner variational loop.

## 2.3 Data Estimator

As part of the numerical experiments, we will apply EVO to fit generative models to corrupted data which we will aim to restore; for instance, we will apply EVO to denoise images corrupted by additive noise or to restore images with missing data (see Section 4). Here we illustrate how an estimator for such data reconstruction can be derived (details are given in Appendix C).

Generally speaking, our aim is to compute estimates $\vec{y}^{\,\mathrm{est}}$ based on observations $\vec{y}^{\,\mathrm{obs}}$. If data points are corrupted by noise but not by missing values, we define the observations $\vec{y}^{\,\mathrm{obs}}$ to be equal to the data points $\vec{y} \in \mathbb{R}^D$ and compute estimates $y_d^{\mathrm{est}}$ for every $d = 1, \ldots, D$ with $D$ denoting the dimensionality of the data. In the presence of missing values, we define data points to comprise an observed part $\vec{y}^{\,\mathrm{obs}}$ and a missing part $\vec{y}^{\,\mathrm{miss}} = \vec{y} \backslash \vec{y}^{\,\mathrm{obs}}$, and we aim to compute estimates $\vec{y}^{\,\mathrm{est}}$ for the missing part. A data estimator can be derived based on the posterior predictive distribution $p(\vec{y}^{\,\mathrm{est}} \mid \vec{y}^{\,\mathrm{obs}})$. Here we will sketch how to derive an estimator for general models with binary latents and continuous observables (in Appendix C we provide a detailed derivation and show how to extend it to models with binary-continuous latents). The posterior predictive distribution for a model with binary latents and continuous observables is given by:

$$p(\vec{y}^{\,\mathrm{est}} \mid \vec{y}^{\,\mathrm{obs}}, \Theta) = \sum_{\vec{s}} p(\vec{y}^{\,\mathrm{est}} \mid \vec{s}, \Theta)\; p(\vec{s} \mid \vec{y}^{\,\mathrm{obs}}, \Theta). \tag{8}$$

We choose to take expectations w.r.t. the posterior predictive distribution, which leads to the following estimator:

$$\langle y_d^{\mathrm{est}} \rangle_{p(\vec{y}^{\,\mathrm{est}} \mid \vec{y}^{\,\mathrm{obs}}, \Theta)} = \Big\langle \langle y_d^{\mathrm{est}} \rangle_{p(y_d^{\mathrm{est}} \mid \vec{s}, \Theta)} \Big\rangle_{p(\vec{s} \mid \vec{y}^{\,\mathrm{obs}}, \Theta)}. \tag{9}$$

The inner expectation in (9) can be identified with the mean of the distribution of the observables given the latents. This distribution is part of the definition of the generative model (compare Section 3.1). The outer expectation in (9) is taken w.r.t. the posterior distribution of the latents. Such expectation can efficiently be approximated using truncated expectations (3). Details and examples of data estimators for concrete models are given in Appendix C.

## 3. Application to Generative Models, Verification and Scalability

Considering Algorithm 1, observe that probabilistic inference for the algorithm is fully defined by the joint $p(\vec{s}, \vec{y}^{\,(n)} \mid \Theta)$ of a given generative model and by a set of hyperparameters for the optimization procedure. No additional and model specific derivations are required for the variational E-step, which suggests Algorithm 1 for "black box" inference for models with binary latents. Using three different generative models, we here (A) verify this "black box" applicability, (B) numerically evaluate the algorithm's ability to recover generating parameters, and (C) show scalability of the approach by applying it to large-scale generative models. To be able to study the properties of the novel variational optimization procedure, we consider generative models for which parameter update equations (M-steps) have been derived previously.

### 3.1 Used Generative Models

The generative models we use are Noisy-OR Bayes Nets (binary latents and binary observables), Binary Sparse Coding (binary latents and continuous observables), and Spike-and-Slab Sparse Coding (binary-continuous latents and continuous observables).

*Noisy-OR Bayes Nets.* A Noisy-OR Bayes Net (NOR; e.g., Singliar and Hauskrecht, 2006;

Jernite et al., 2013; Rotmensch et al., 2017) is a non-linear bipartite data model with all-to-all connectivity between the layer of hidden and observed variables (and no intra-layer connections). Latents and observables both take binary values, i.e., $\vec{s} \in \{0,1\}^H$ and $\vec{y} \in \{0,1\}^D$ with $H$ and $D$ denoting the dimensions of the latent and observed spaces. The model assumes a Bernoulli prior for the latents, and non-zero latents are then combined via the noisy-OR rule:

$$p\,(\vec{s} \mid \Theta) = \prod_{h=1}^{H} \mathcal{B}(s_h; \pi_h), \qquad p\,(\vec{y} \mid \vec{s}, \Theta) = \prod_{d=1}^{D} N_d(\vec{s})^{y_d}(1 - N_d(\vec{s}))^{1-y_d}, \tag{10}$$

where $\mathcal{B}(s_h; \pi_h) = \pi_h^{s_h}(1-\pi_h)^{1-s_h}$ and where $N_d(\vec{s}) := 1 - \prod_{h=1}^{H}(1 - W_{dh}s_h)$. In the context of the NOR model, $\Theta = \{\vec{\pi}, W\}$, where $\vec{\pi}$ is the set of values $\pi_h \in [0,1]$ representing the prior activation probabilities for the hidden variables $s_h$, and $W$ is a $D{\times}H$ matrix of values $W_{dh} \in [0,1]$ representing the probability that an active latent variable $s_h$ activates the observable $y_d$. M-step update rules for the NOR model can be derived by inserting (10) into the lower bound (1) and by then taking derivatives of the resulting expression w.r.t. all model parameters. The update rule for the $W$ parameter does not allow for a closed-form solution and a fixed-point equation is employed instead. The resulting M-step equations are shown in Appendix D.

*Binary Sparse Coding.* Binary Sparse Coding (BSC; Haft et al., 2004; Henniges et al., 2010) is an elementary sparse coding model for continuous data with binary latent and continuous observed variables. Similar to standard sparse coding (Olshausen and Field, 1997; Lee et al., 2007), BSC assumes that, given the latents, the observables follow a Gaussian distribution. BSC and standard sparse coding differ from each other in their latent variables. In standard sparse coding, latents take continuous values and are typically modeled, e.g., with Laplace or Cauchy distributions. BSC uses binary latents $\vec{s} \in \{0,1\}^H$ which are assumed to follow a Bernoulli distribution $\mathcal{B}(s_h; \pi)$ with the same activation probability $\pi$ for each hidden unit $s_h$. The combination of the latents is described by a linear superposition rule. Given the latents, the observables $\vec{y} \in \mathbb{R}^D$ are independently and identically drawn from a Gaussian distribution:

$$p\,(\vec{s} \mid \Theta) = \prod_{h=1}^{H} \mathcal{B}(s_h; \pi), \qquad p\,(\vec{y} \mid \vec{s}, \Theta) = \prod_{d=1}^{D} \mathcal{N}(y_d; \sum_{h=1}^{H} W_{dh}s_h, \sigma^2). \tag{11}$$

The parameters of the BSC model are $\Theta = (\pi, \sigma, W)$. $W$ is a $D \times H$ matrix whose entries $W_{dh}$ contain the weights associated with each hidden unit $s_h$ and obervable $y_d$; $\sigma$ determines the standard deviation of the Gaussian. The M-step equations for BSC can be derived analogously to the NOR model by inserting (11) into (1) and by then optimizing the resulting expression w.r.t. each model parameter (compare, e.g., Henniges et al., 2010). As opposed to NOR, each of the BSC update equations can be derived in closed-form. The explicit expressions are listed in Appendix D.

*Spike-and-Slab Sparse Coding.* As a final example, we consider a more expressive data model and use EVO to optimize the parameters of a spike-and-slab sparse coding (SSSC) model. SSSC extends the generative model of BSC in the sense that it uses a spike-and-slab instead of a Bernoulli prior distribution. The SSSC model has been used in a number of previous studies and in a number of variants (Titsias and Lázaro-Gredilla, 2011; Lücke and Sheikh, 2012; Goodfellow et al., 2012; Sheikh et al., 2014, and many more). It can be formulated using two sets of latents, $\vec{s} \in \{0,1\}^H$ and $\vec{z} \in \mathbb{R}^H$, which are combined via pointwise multiplication s.t. $(\vec{s} \odot \vec{z})_h = s_h z_h$. Here we take the continuous latents to be distributed according to a multivariate Gaussian with mean $\vec{\mu}$ and a full covariance matrix $\Psi$. The binary latents follow a Bernoulli distribution with individual activation probabilities $\pi_h$, $h = 1, \ldots, H$:

$$p\,(\vec{s} \mid \Theta) = \prod_{h=1}^{H} \mathcal{B}(s_h; \pi_h), \qquad p\,(\vec{z} \mid \Theta) = \mathcal{N}(\vec{z}; \vec{\mu}, \Psi). \tag{12}$$

As in standard sparse coding, SSSC assumes components to combine linearly. The linear combination then determines the mean of a univariate Gaussian for the observables:

$$p\,(\vec{y} \mid \vec{s}, \vec{z}, \Theta) = \mathcal{N}(\vec{y}; \sum_{h=1}^{H} \vec{W}_h s_h z_h, \sigma^2 \mathbb{1}) \tag{13}$$

where $W$ is a $D \times H$ matrix with columns $\vec{W}_h$ and where $\mathbb{1}$ is the unit matrix. The M-step update rules of the SSSC model can be derived by taking derivatives of the free energy

$$\mathcal{F}(q, \Theta) = \sum_{n=1}^{N} \big\langle \log p(\vec{y}^{\,(n)}, \vec{s}, \vec{z} \mid \Theta) \big\rangle_{q^{(n)}} + \sum_{n=1}^{N} H[q^{(n)}] \tag{14}$$

w.r.t. each of the model parameters $\Theta = (\vec{\pi}, \sigma, W, \vec{\mu}, \Psi)$. The term $\big\langle f(\vec{s}, \vec{z}\,) \big\rangle_{q^{(n)}}$ in (14) denotes the expectation of $f(\vec{s}, \vec{z}\,)$ w.r.t. a variational distribution $q^{(n)}(\vec{s}, \vec{z}\,)$:

$$\big\langle f(\vec{s}, \vec{z}\,) \big\rangle_{q^{(n)}} = \sum_{\vec{s}} \int q^{(n)}(\vec{s}, \vec{z}\,) f(\vec{s}, \vec{z}\,)\, \mathrm{d}\vec{z}. \tag{15}$$

The term $H[q^{(n)}]$ in (14) denotes the entropy $-\big\langle \log q^{(n)}(\vec{s}, \vec{z}\,) \big\rangle_{q^{(n)}}$ of the variational distributions. For a detailed derivation of the SSSC M-step equations see Sheikh et al. (2014) and compare Goodfellow et al. (2012). For SSSC, all M-step updates (A) have closed-form solutions and (B) contain as arguments expectation values (15). Importantly for applying EVO, all these expectation values can be reformulated as expectations w.r.t. the posterior over the binary latent space. For more details see Appendix D and compare Sheikh et al. (2014). Based on the reformulations, the expectations w.r.t. the posterior over the binary latent space can be approximated using the truncated expectations (3). Since the joint $p(\vec{y}, \vec{s} \mid \Theta)$ of the SSSC model is computationally tractable and given by

$$p(\vec{y}, \vec{s} \mid \Theta) = \mathcal{B}(\vec{s}; \vec{\pi})\; \mathcal{N}(\vec{y};\, \tilde{W}_{\vec{s}}\, \vec{\mu}, C_{\vec{s}}), \tag{16}$$

with $C_{\vec{s}} = \sigma^2 \mathbb{1} + \tilde{W}_{\vec{s}}\, \Psi\, \tilde{W}_{\vec{s}}^{\mathrm{T}}$, $(\tilde{W}_{\vec{s}})_{dh} = W_{dh} s_h$ s.t. $W\,(\vec{s} \odot \vec{z}\,) = \tilde{W}_{\vec{s}}\, \vec{z}$, the variational lower bound (4) can be efficiently optimized using Algorithm 1. The EVO algorithm consequently provides a novel, evolutionary approach to efficiently optimize the parameters of the SSSC data model.

### 3.2 Verification: Recovery of Generating Parameters

First, we verified EVO by training NOR, BSC and SSSC models on artificial data for which the ground-truth generative parameters were known. We used the bars test as a standard setup for such purposes (Földiák, 1990; Hoyer, 2003; Lücke and Sahani, 2008). We started with a standard bars test using a weight matrix $W^{\mathrm{gen}}$ whose $H^{\mathrm{gen}} = 2\sqrt{D}$ columns represented generative fields in the form of horizontal and vertical bars. We considered a dictionary of $H^{\mathrm{gen}} = 10$ (Lücke and Sahani, 2008; Lücke and Eggert, 2010; Lücke et al., 2018; Sheikh and Lücke, 2016) components. For NOR, we set the amplitudes of the bars and of the background to a value of 0.8 and 0.1, respectively; for BSC and SSSC, the bar amplitudes were uniformly randomly set to 5 and -5, and the background was set to 0. For BSC and SSSC, we chose $(\sigma^{\mathrm{gen}})^2 = 1$ as ground-truth variance parameter; for SSSC, we furthermore defined $\vec{\mu}^{\mathrm{gen}} = \vec{0}$ and $\Psi^{\mathrm{gen}} = \mathbb{1}$.

For each model, 30 data sets of $N = 5{,}000$ samples were generated. We used $\pi_h^{\mathrm{gen}} = \frac{2}{H^{\mathrm{gen}}}\, \forall\, h = 1, \ldots, H^{\mathrm{gen}}$ (see Figure 2 for a few examples). We then applied EVO to train NOR, BSC and SSSC models with $H = H^{\mathrm{gen}} = 10$ components (EVO hyperparameters are listed in Table 4 in Appendix F.2). For NOR, priors $\pi_h$ were initialized to an arbitrary sparse value (typically $\frac{1}{H}$) while the initial weights $W^{\mathrm{init}}$ were sampled from the standard uniform distribution. The $\mathcal{K}^{(n)}$ sets were initialized by sampling bits from a Bernoulli distribution that encouraged sparsity. In our experiments, the mean of the distribution was set to $\frac{1}{H}$. For BSC, the initial value of the prior was defined $\pi^{\mathrm{init}} = \frac{1}{H}$; the value of $(\sigma^{\mathrm{init}})^2$ was set to the variance of the data points averaged over the observed dimensions. The columns of the $W^{\mathrm{init}}$ matrix were initialized with the mean of the data points to which some Gaussian noise with a standard deviation of $0.25\,\sigma^{\mathrm{init}}$ was added. The initial values of the $\mathcal{K}^{(n)}$ sets were drawn from a Bernoulli distribution with $p(s_h = 1) = \frac{1}{H}$. To initialize the SSSC model, we uniformly randomly drew $\vec{\pi}^{\mathrm{init}}$ and $\vec{\mu}^{\mathrm{init}}$ from the interval $[0.1, 0.5]$ and $[1, 5]$, respectively (compare Sheikh et al., 2014) and set $\Psi^{\mathrm{init}}$ to the unit matrix. For SSSC, we proceeded as we had done for the BSC model to initialize the dictionary $W$, the variance parameter $\sigma^2$ and the sets $\mathcal{K}^{(n)}$. The evolution of the model parameters during learning is illustrated in Figure 2 for an exemplary run of the "fitparents-randflip" EA (illustrations of the parameters $\sigma$ of the BSC model and $\Psi$, $\vec{\mu}$ and $\sigma$ of the SSSC model are depicted in Figure 8 in Appendix F).

Figure 2 illustrates that the ground-truth generative parameters can accurately be recovered for each model using EVO. For all models, the bar-like structure of the ground-truth generative fields (GFs) is correctly captured (in permuted order) in the learned GFs. For the SSSC model, the GFs, latent means and latent covariances are recovered in scaled versions compared to the respective ground-truth values. This effect can be considered as a consequence of the spike-and-slab prior which allows for multiple equivalent solutions for the amplitudes of $W$ and $\vec{z}$ given a data point. We further used the setup with artificial data and known ground-truth generative parameters to explore EAs with different types of combinations of genetic operators. Different combinations of genetic operators resulted in different degrees of ability to recover the ground-truth parameters. For instance, for NOR trained with EVO, the combination of random uniform parent selection, single-point crossover and sparsity-driven bitflip ("randparents-cross-sparseflips") resulted in the best behavior for bars test data (also compare Guiraud et al., 2018). For BSC and SSSC trained with EVO, the best operator combinations were "fitparents-cross-sparseflips" and "fitparents-randflips",

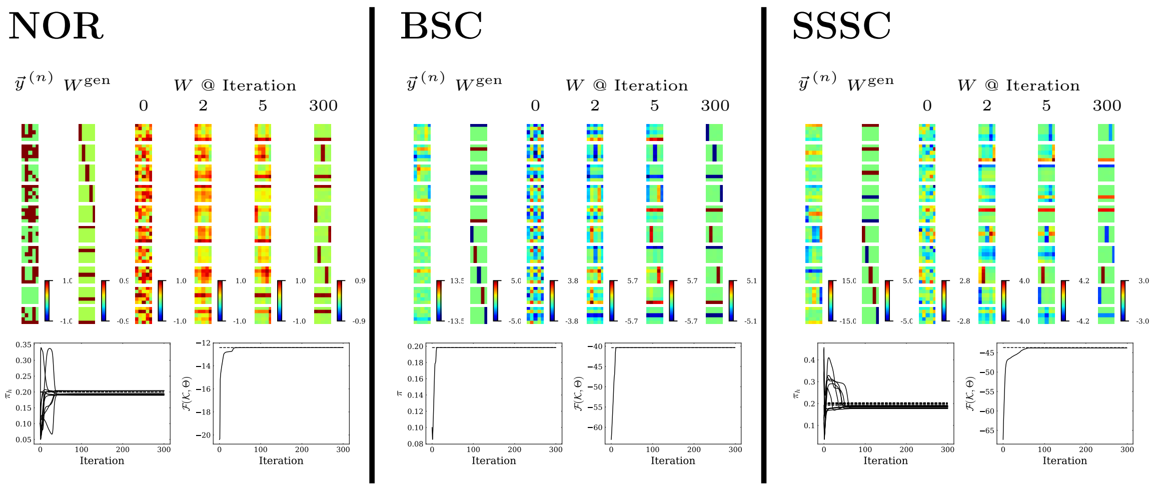


Figure 2: Recovering ground-truth generative parameters of NOR, BSC and SSSC models on artificial data using EVO (see text for details).

respectively (compare Figure 9 in Appendix F). In general, the best combination of operators will depend on the data model and the data set used.

### 3.3 Scalability to Large Generative Models

Having verified the ability of EVO to recover ground-truth generating parameters for the three generative models of Section 3.1, we continued with investigating how well the evolutionary approximation was able to optimize large-scale models. Scalability is a property of crucial importance for any approximation method, and large-scale applicability is essential to accomplish many important data processing tasks based on generative models. To investigate scalability, we used natural image patches which are known to be richly structured and which require large models to appropriately model their distribution. We used image patches extracted from a standard image database (van Hateren and van der Schaaf, 1998). We trained the NOR, BSC and SSSC data models using the "fitparents-cross-sparseflips" variant of EVO as we observed this operator combination to perform well across all models for this data.

For NOR, we considered raw image patches, i.e., images which were not mean-free or whitened and reflected light intensities relatively directly. We used image patches that were generated by extracting random $10 \times 10$ subsections of a single $255 \times 255$ image of overlapping grass wires (part of image 2338 of the database). We clamped the brightest 1 % pixels from the data set, and scaled each patch to have gray-scale values in the range $[0, 1]$. From these patches, we then created $N = 30{,}000$ data points $\vec{y}^{(n)}$ with binary entries by repeatedly choosing a random gray-scale image and sampling binary pixels from a Bernoulli distribution with parameter equal to the pixel values of the patches with $[0, 1]$ gray-scale. Because of mutually occluding grass wires in the original image, the generating components of the binary data could be expected to follow a non-linear superimposition, which motivated the application of a non-linear generative model such as NOR. For the data with $D = 10{\times}10 = 100$ observables, we applied a NOR model with $H = 100$ latents (EVO hyperparameters are listed in Table 4 in Appendix F.2). During training, we clamped the priors $\pi_h$ to a minimum value of $1/H$ to encourage the model to make use of all generative fields. Figure 3 (top) shows a random selection of 50 generative fields learned by the NOR model (the full dictionary is displayed in Figure 10 in Appendix F). Model parameter initialization followed the same procedure as for the bars tests. As can be observed, EVO is efficient for large-scale NOR on real data. Many of the generative fields of Figure 3 (top) resemble curved edges, which is in line with expectations and with results, e.g., as obtained by Lücke and Sahani (2008) with another non-linear generative model.

For the BSC generative model, we are not restricted to positive data. Therefore, we can use a whitening procedure as is customary for sparse coding approaches (Olshausen and Field, 1997). We employed $N = 100{,}000$ image patches of size $D = 16 \times 16 = 256$ which were randomly picked from the van Hateren data set. The highest 2 % of the pixel values were clamped to compensate for light reflections, and patches without significant structure were removed to prevent amplifications of very small intensity differences. The whitening procedure we subsequently applied is based on ZCA whitening (Bell and Sejnowski, 1997) retaining 95 % of the variance (compare Exarchakis and Lücke, 2017). We then applied EVO to fit a BSC model with $H = 300$ components to the data (EVO hyperparameters are listed in Table 4 in Appendix F.2). To initialize the model and the variational parameters, we proceeded as we had done for the bars test (Section 3.2). Figure 3 (middle) depicts some of the generative fields learned by the BSC model. These fields correspond to the 60 hidden units with highest prior activation probability (see Figure 10 in Appendix F for the full dictionary). The obtained generative fields primarily take the form of the well-known Gabor functions with different locations, orientations, phase, and spatial frequencies.

We collected another $N = 100{,}000$ image patches of size $D = 12 \times 12 = 144$ from the van Hateren data set to fit a SSSC model with $H = 512$ components (EVO hyperparameters listed in Table 4 in Appendix F.2). We applied the same preprocessing as for the BSC model and initialized the model and the variational parameters similarly to the bars test. Figure 3 (bottom) shows a random selection of 60 out of the 512 generative fields learned by the SSSC model (the full dictionary is displayed in Figure 10 in Appendix F). The experiment shows that EVO can be applied to train complex generative models on large-scale realistic data sets; the structures of the learned generative fields (a variety of Gabor-like and a few globular fields) are in line with observations made in previous studies which applied SSSC models to whitened image patches (see, e.g., Sheikh and Lücke, 2016).

## 4. Performance Comparison on Denoising and Inpainting Benchmarks

So far, we have verified that EVO can be applied to the training of elementary generative models such as Noisy-OR Bayes Nets (NOR) and Binary Sparse Coding (BSC) but also to the more expressive model of spike-and-slab sparse coding (SSSC) which features a more flexible prior than BSC or standard sparse coding. To perform variational optimization for these models, no additional analytical steps were required and standard settings of optimization parameters were observed to work well. However, an important question is how well EVO is able to optimize model parameters compared to other methods. And more generally,

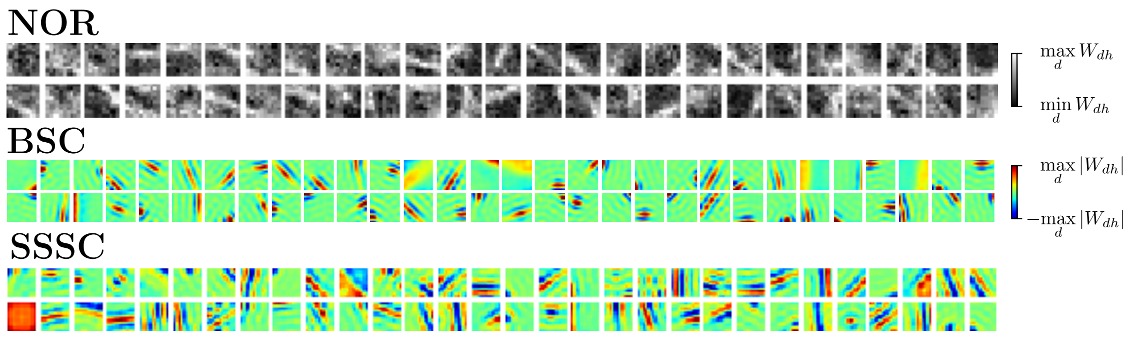


Figure 3: Dictionary elements learned by NOR (top), BSC (middle) and SSSC (bottom) models on natural image patches. The models were trained on separate data sets that had been obtained using different preprocessing schemes (see text). For NOR, a random selection of 50 out of 100 fields is displayed; for BSC the fields corresponding to the 60 out of 300 most active hidden units are shown; for SSSC a random selection of 60 out of 512 fields is displayed. The full dictionaries learned by the individual models are depicted in Figure 10 in Appendix F.

how well do generative model algorithms perform in benchmarks if their parameters are optimized by EVO? Here we will address these questions using two standard and popular benchmarks: image denoising and image inpainting. These two benchmarks offer themselves for performance evaluation as benchmark data is available for a large range of different algorithms (including algorithms based on generative models among many others). The data for standard denoising and inpainting benchmarks is continuous. We will consequently focus on the BSC and SSSC generative models. In the following, we will refer to EVO applied to BSC as *Evolutionary Binary Sparse Coding* (EBSC) and to EVO applied to SSSC as *Evolutionary Spike-and-Slab Sparse Coding* (ES3C). As genetic operator combination for the EA, we will use "fitparents-randflips" for all subsequent experiments. The main reason to prefer this combination over combinations that also use crossover is computational efficiency (which is a major limiting factor for the following experiments).

## 4.1 Algorithm Types and Comparison Categories

We will use standard benchmarks to compare EBSC and ES3C w.r.t. (A) algorithms based on generative models equal or similar to the ones used in this work, (B) other probabilistic methods including mixture models and Markov random fields approaches, (C) non-local image specific methods and (D) deep neural networks. Table 1 gives an overview of the methods we compare to, sorted by algorithm type. To the knowledge of the authors, the listed papers represent the best performing approaches while we additionally included standard baselines such as BM3D, EPLL or MLP (see Table 1 for references).

The algorithms listed in Table 1 have different requirements that have to be fulfilled for them to be applicable. To facilitate comparison, we have grouped the benchmarked algorithms based on the prior knowledge they require/use (see Table 2 and Appendix E for a related discussion). As can be observed, some algorithms (e.g., KSVD or BM3D) require the

| Acronym | Algorithm Name / Type | Reference |
|---|---|---|
| | **Sparse Coding / Dictionary Learning** | |
| MTMKL | Spike-and-Slab Multi-Task and Multiple Kernel Learning | Titsias and Lázaro-Gredilla (2011) |
| BPFA | Beta Process Factor Analysis | Zhou et al. (2012) |
| GSC | Gaussian Sparse Coding | Sheikh et al. (2014) |
| KSVD | Sparse and Redundant Representations Over Trained Dictionaries | Elad and Aharon (2006) |
| cKSVD | Sparse Representation for Color Image Restoration | Mairal et al. (2008) |
| LSSC | Learned Simultaneous Sparse Coding | Mairal et al. (2009) |
| BKSVD | Bayesian K-SVD | Serra et al. (2017) |
| | **Other Probabilistic Methods (GMMs, MRFs)** | |
| EPLL | Expected Patch Log Likelihood | Zoran and Weiss (2011) |
| PLE | Piecewise Linear Estimators | Yu et al. (2012) |
| FoE | Fields of Experts | Roth and Black (2009) |
| MRF | Markov Random Fields | Schmidt et al. (2010) |
| NLRMRF | Non-Local Range Markov Random Field | Sun and Tappen (2011) |
| | **Non-Local Image Processing Methods** | |
| BM3D | Block-Matching and 3D Filtering | Dabov et al. (2007) |
| NL | Patch-based Non-Local Image Interpolation | Li (2008) |
| WNNM | Weighted Nuclear Norm Minimization | Gu et al. (2014) |
| | **Deep Neural Networks** | |
| MLP | Multi Layer Perceptron | Burger et al. (2012) |
| DnCNN-B | Denoising Convolutional Neural Network | Zhang et al. (2017) |
| TNRD | Trainable Nonlinear Reaction Diffusion | Chen and Pock (2017) |
| MemNet | Deep Persistent Memory Network | Tai et al. (2017) |
| IRCNN | Image Restoration Convolutional Neural Network | Chaudhury and Roy (2017) |
| GSVAE | Gumbel-Softmax Variational Autoencoder | Jang et al. (2017) |
| FFDNet | Fast and Flexible Denoising Convolutional Neural Network | Zhang et al. (2018) |
| DIP | Deep Image Prior | Ulyanov et al. (2018) |
| DPDNN | Denoising Prior Driven Deep Neural Network | Dong et al. (2019) |
| BDGAN | Image Blind Denoising Using Generative Adversarial Networks | Zhu et al. (2019) |
| BRDNet | Batch-Renormalization Denoising Network | Tian et al. (2020) |

Table 1: Algorithms used for comparsion on denoising and/or inpainting benchmarks. The algorithms have been grouped into four types of approaches. The table includes the best performing algorithms and standard baselines.

noise level to be known a-priori, for instance. Others require (often large amounts of) clean images for training, which typically applies for feed-forward deep neural networks (DNNs). If an algorithm is able to learn without information other than the corrupted image itself, it is commonly referred to as a "zero-shot" learning algorithm (compare, e.g., Shocher et al., 2018; Imamura et al., 2019). Algorithms of the "zero-shot" category we compare to are, e.g., MTMKL, BPFA, DIP and GSVAE. The EVO-based algorithms EBSC and ES3C also fall into the "zero-shot" category. In Table 2, we have labeled the categories DE1-DE6 for denoising and IN1-IN6 for inpainting. DE2 could also be referred to as "zero-shot + noise level" as the algorithms of that category also do not need additional images.

**A** Denoising

| Clean training data required/used | | Further a-priori assumptions | | |
|---|---|---|---|---|
| | | None | Noise Level | Test-Train Match |
| No | | DE1<br>MTMKL<br>BPFA<br>GSC<br>GSVAE<br>BKSVD<br>EBSC<br>ES3C | DE2<br>KSVD<br>LSSC<br>BM3D<br>WNNM | DE3<br>n/a |
| Yes | | DE4<br>n/a | DE5<br>EPLL | DE6<br>MLP<br>DnCNN-B<br>TNRD<br>MemNet<br>FFDNet<br>DPDNN<br>BDGAN<br>BRDNet |

**B** Inpainting

| Clean training data required/used | | Further a-priori assumptions | | |
|---|---|---|---|---|
| | | None | Noise Level | Test-Train Match |
| No | | IN1<br>MTMKL<br>BPFA<br>BKSVD<br>DIP<br>ES3C | IN2<br>NL*<br>PLE | IN3<br>n/a |
| Yes | | IN4<br>FoE<br>MRF | IN5<br>cKSVD | IN6<br>NLRMRF<br>IRCNN |

Table 2: Algorithms for denoising and inpainting use different degrees and types of prior knowledge. In panel A, the algorithms listed in the column “Noise Level” assume access to the ground-truth noise level of the test data. The approaches listed in the column “Test-Train Match” in panel A are optimized either for one single or for several particular noise levels. In panel B, the algorithms listed in the column "Noise Level" (NL, PLE, cKSVD) use heuristics for noise level and/or sparsity estimation (*NL internally makes use of BM3D which requires a noise level (which is in turn set by a patch-by-patch heuristics); PLE and cKSVD employ manually set noise level/sparsity estimates). The column “Test-Train Match” in panel B lists algorithms that are optimized for a particular percentage of missing pixels (e.g. for images with 80% randomly missing values) or for a particular missing value pattern. The algorithms of category IN6 can be applied without providing a mask that indicates the pixels that are to be filled (see Appendix E for details).

### 4.2 Image Denoising

To apply EBSC and ES3C for image denoising, we first preprocessed the noisy images that we aimed to restore by cutting the images into smaller patches: given an image of size $\mathcal{H} \times \mathcal{W}$ and using a patch size of $D = P_x \times P_y$, we cut all possible $N = (\mathcal{W} - \mathcal{P}_\mathrm{x} + 1) \times (\mathcal{H} - \mathcal{P}_\mathrm{y} + 1)$ patches by moving a sliding window over the noisy image. Patches were collected individually for each image and corresponded to the data sets $\mathcal{Y}$ which EBSC or ES3C were trained on. In other words, EBSC and ES3C leveraged nothing except of the information of the noisy image itself (no training on other images, no training on clean images, noise unknown a-priori). After model parameters had been inferred using EBSC and ES3C, we took expectations w.r.t. the posterior predictive distribution of the model in order to estimate the non-noisy image pixels. For each noisy patch, we estimated all its non-noisy pixel values, and, as commonly done, we repeatedly estimated the same pixel value based on different (mutually overlapping) patches that shared the same pixel. The different estimates for the same pixel were then gathered to determine the non-noisy value using a weighted average (for details see Section 2.3 and Appendix C; also see, e.g., Burger et al., 2012). Finally, we were interested in how well EBSC and ES3C could denoise a given image in terms of the standard peak-signal-to-noise ratio (PSNR) evaluation measure.

#### 4.2.1 Relation between PSNR measure and variational lower bound

We first investigated whether the value of the learning objective (the variational lower bound) of EBSC and ES3C was instructive about the denoising performance of the algorithms in terms of PSNR. While the PSNR measure requires access to the clean target image, the learning objective can be evaluated without such ground-truth knowledge. If learning objective and PSNR measure were reasonably well correlated, one could execute the algorithms several times on a given image and determine the best run based on the highest value of the learning objective without requiring access to the clean image. For the experiment, we used the standard “House” image degraded by additive white Gaussian (AWG) noise with a standard deviation of $\sigma = 50$ (the clean and noisy images are depicted in Figure 5). We applied ES3C for denoising using patches of size $D = 8 \times 8$ and a dictionary with $H = 256$ elements (EVO hyperparameters listed in Table 4 in Appendix F.2). At several iterations during execution of the algorithm, we measured the value of the learning objective and the PSNR of the reconstructed image. The experiment was repeated five times, each time using the same noisy image but a different initialization of the algorithm. Figure 4 illustrates the result of this experiment. As can be observed, PSNR values increase with increasing value of the objective $\mathcal{F}(\mathcal{K}, \Theta) \,/\, N$ during learning. The truncated distributions which give rise to the lower bound are consequently well suited. Only at the end of learning, PSNR values slightly decrease while the lower bound still increases (Figure 4, inset). The final PSNR decreases are likely due to the PSNR measure being not perfectly correlated with the data likelihood. However, decreases of the likelihood while the lower bound still increases cannot be excluded. While the final decreases of PSNR are small (see scale of inset figure of Figure 4), their consequence is that the run with the highest final value for the variational bound is not necessarily the run with the highest PSNR. For Figure 4, for instance, Run 1 has the highest final variational bound but Run 4 the highest final PSNR.

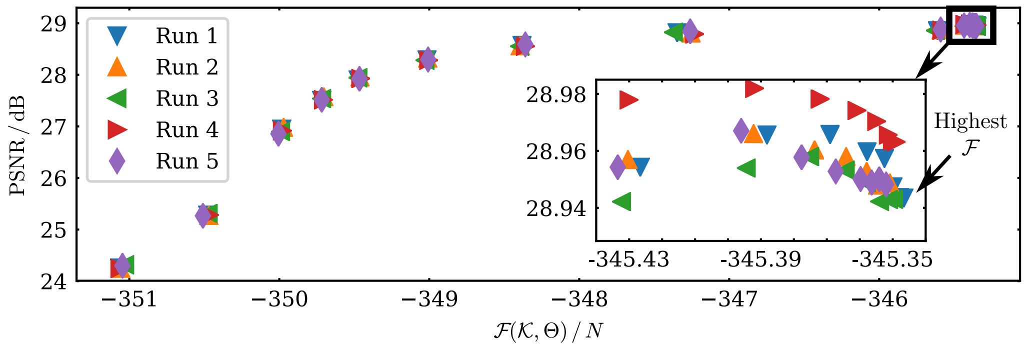


Figure 4: Variational bound and PSNR value pairs obtained from applying ES3C to the noisy "House" image (AWG noise with $\sigma = 50$). 2,000 EVO iterations were performed and PSNR values were measured at iterations 1, 2, 5, 10, 20, 50, 100, 200, 400, 600, 800, 1,000, 1,200, 1,400, 1,600, 1,800, 2,000. The experiment was repeated five times using the same noise realization in each run. The PSNR values at the last iteration were 28.94, 28.95, 28.94, 28.96, 28.95 (avg. rounded $28.95 \pm 0.01$; all values in dB).

### 4.2.2 Comparison of Generative Models and Approximate Inference

Before comparing EBSC and ES3C with a broad range of different denoising algorithms (further below), we first focused on approaches that are all based on essentially the same data model. Differences in measured performance can then be attributed more directly to differences in the method used for parameter optimization. Concretely, we first considered algorithms that are all based on a spike-and-slab sparse coding (SSSC) model. Namely, we compared the following algorithms:

- the ES3C algorithm which is trained using variational optimization of truncated posteriors (EVO),
- the MTMKL algorithm (Titsias and Lázaro-Gredilla, 2011) which uses a factored variation approach (i.e., mean field) for parameter optimization (also compare Goodfellow et al., 2012),
- the BPFA algorithm (Zhou et al., 2012) which uses sampling for parameter optimization,
- the GSC algorithm (Sheikh et al., 2014) which uses truncated posteriors constructed using preselection (i.e., no variational loop).

In addition, we included EBSC into the comparison. EBSC is based on the BSC generative model, and the BSC generative model can be optimized using different optimization approaches. Goodfellow et al. (2012), for instance, discuss the BSC model as a boundary case of their mean-field optimization of the SSSC generative model. Furthermore, BSC can be optimized using encoding neural networks in conjunction with reparameterization and the Gumbel-Softmax trick (Jang et al., 2017). Using the Gumbel-Softmax trick, standard optimization techniques developed for VAEs can be maintained also if discrete latent variables are used[3]. Optimization using gradient ascent via amortized inference, reparameterization and Gumbel-Softmax trick can consequently also be used to optimize the BSC data model. We included such a Gumbel-Softmax optimized BSC model for comparison, and refer to it as GSVAE$^{\text{lin}}$ (for *Gumbel-Softmax Variational Autoencoder*). More concretely, we used the VAE setup suggested by Jang et al. (2017) but used a linear decoder to match the linear generative model of BSC (the superscript "lin" indicates the linearity); to match the binary latents of BSC, we used latents with only two categories and ensured that the category corresponding to "0" did not contribute to data generation. Our implementation of GSVAE$^{\text{lin}}$ is based on the source code provided by the original publication (Jang, 2016), and we conducted further control experiments also using GSVAE versions more closely aligned with the original VAE architectures (see Appendix F.3 for details).

For best comparability of the different optimization approaches, we first investigated the denoising performance of the algorithms under controlled conditions: We considered a fixed task (the "House" benchmark with $\sigma = 50$ AWG noise), and we compared algorithms for three different configurations of $D$ and $H$ (see Figure 5 B). PSNR values for BPFA, MTMKL and GSC are taken from the corresponding original publications. The publication for GSC only reports values for $D = 8 \times 8$ and $H = 64$ as well as for $D = 8 \times 8$ and $H = 256$; MTMKL only for the former and BPFA only for the latter setting. We additionally cite the performance of MTMKL for $D = 8 \times 8$ and $H = 256$ as reported by Sheikh et al. (2014). For GSVAE$^{\text{lin}}$, EBSC and ES3C, PSNR values were obtained via patch averaging as described above (compare Section 4.2); the values reported correspond to average PSNRs of three runs of the respective algorithm (standard deviations were smaller or equal 0.13 dB PSNR; results of the individual runs are listed in Table 3 in Appendix F). For EBSC and ES3C, the same EVO hyperparameters were used (hyperparameters, including those used for GSVAE$^{\text{lin}}$, are listed in Table 4 in Appendix F.2).

Considering Figure 5 B, BPFA and MTMKL (which both represented the state-of-the-art at the time of their publication) perform well but, in comparison, ES3C shows significant further improvements. As the major difference between the considered approaches is the parameter optimization method rather than the generative model, the performance increases of ES3C compared to the other algorithms can be attributed to the used evolutionary variational optimization (EVO). An important factor of the performance increases for ES3C is presumably its ability to leverage the available generative fields more effectively. Titsias and Lázaro-Gredilla (2011) report for MTMKL, for instance, that larger dictionaries ($H > 64$) do not result in performance increases; similarly Zhou et al. (2012) report that BPFA models with 256 and 512 dictionary elements yield similar performance. GSC has been shown to make better use of larger dictionaries than MTMKL (Sheikh et al., 2014). Performance of GSC is, however, significantly lower than ES3C for all investigated dictionary sizes, which provides strong evidence for the advantage of the fully variational optimization applied by the EVO algorithm. The evolutionary variational optimization also results in a notably strong performance of EBSC compared to previous approaches: it outperforms MTMKL (as well as GSC for $H = 64$). The strong performance may be unexpected as the underlying

3. Essentially, a categorical distribution is annealed, i.e. becomes continuous. Reparameterization (which is otherwise not applicable to discrete latents) can now be used to estimate gradients. Reversal of the annealing process then recovers discrete latent values in the limit of low temperatures (e.g., Jang et al., 2017, for details).

BSC model is less expressive than the SSSC data model. The performance gains using EVO thus outweigh the potentially better data model in this case.

In comparison, optimization of the BSC model using VAE-like optimization (GSVAE$^{\text{lin}}$) resulted in much lower denoising performance: In all conditions (i.e. for all considered settings of $D$ and $H$), the PSNR values obtained with GSVAE$^{\text{lin}}$ were significantly lower compared to all other algorithms used for comparison (see Figure 5 B). A reason for the lower PSNR values of GSVAE$^{\text{lin}}$ compared to approaches such as MTMKL or EBSC is a much denser encoding, which, for the considered setting, is less advantageous than the sparse code learned by EBSC and other approaches (we elaborate further below). Finally, while PSNR values for EBSC can be higher than for methods based on spike-and-slab models, ES3C performs (given the same set of hyperparameters) much stronger than EBSC in all settings. The spike-and-slab-based ES3C algorithm can also make much more use of larger dictionaries and larger patches (see Figure 5 B).

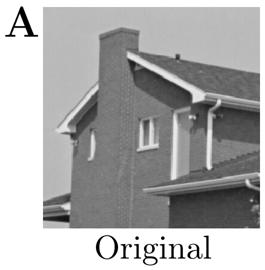
### 4.2.3 General comparison of denoising approaches

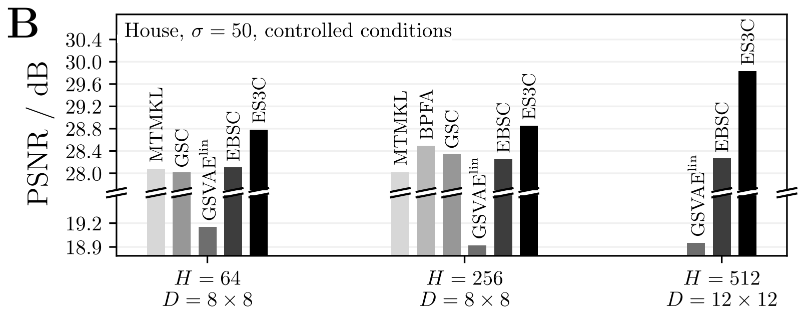
After having compared a selected set of algorithms under controlled conditions, we investigated denoising performance much more generally. We compared the best performing algorithms on standard denoising benchmarks irrespective of the general approach they follow, allowing for different task categories they require, and allowing for any type of hyperparameters they use. The methods we have used for comparison are listed in Table 2 A, where the algorithms are grouped according to the prior information they assume (compare Section 4.1 and Appendix E).

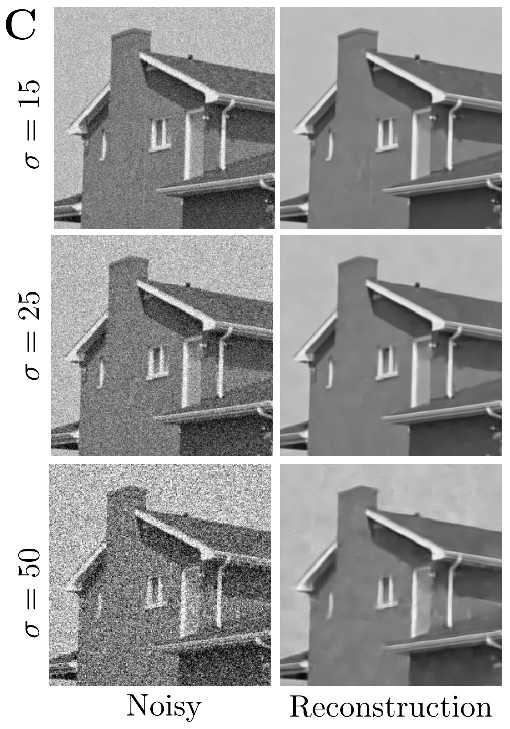
A limitation of most of the investigated algorithms is the computational resources they require. For EBSC and ES3C, model size (i.e., patch and dictionary sizes) are limited by available computational resources and so is the number of variational states $S$ (Section 2.1) used to approximate posteriors. For ES3C, a good trade-off between performance vs. computational demand was observed for instance for $D = 12 \times 12$ and $H = 512$ with $S = 60$ (for the experiment of Figure 5 D below); for EBSC a good trade-off between performance and computational demand was observed for $D = 8 \times 8$ and $H = 256$ with $S = 200$. We thus have used more variational states $S$ for EBSC than for ES3C. Technical details including EVO hyperparameters are provided in Appendix F.

Figures 5 D and 6 list denoising performances obtained with algorithms from the different categories of Table 2. The listed PSNR values are taken from the respective original publications with the following exceptions: For WNNM, EPLL and MLP, values are taken from Zhang et al. (2017) and for TNRD and MemNet values equal the ones in Dong et al. (2019). PSNR values for GSVAE$^{\text{lin}}$ were obtained based on the source code provided by the original publication (Jang, 2016) but using the same linear decoder as for Figure 5 B. We observed the patch size and the number of binary latents to be significant hyperparameters for GSVAE$^{\text{lin}}$; therefore, compared to the setting in Figure 5 B, we used larger patch sizes and a higher number of latents for GSVAE$^{\text{lin}}$ in the experiments of Figures 5 D and 6 (see Table 4 and Appendix F.3 for the hyperparameters used).

PSNR values reported in the original papers are often single PSNR values which may correspond to a single run of the investigated algorithm or to the best observed PSNR value. For deterministic methods with essentially no PSNR variations for a fixed input, it might



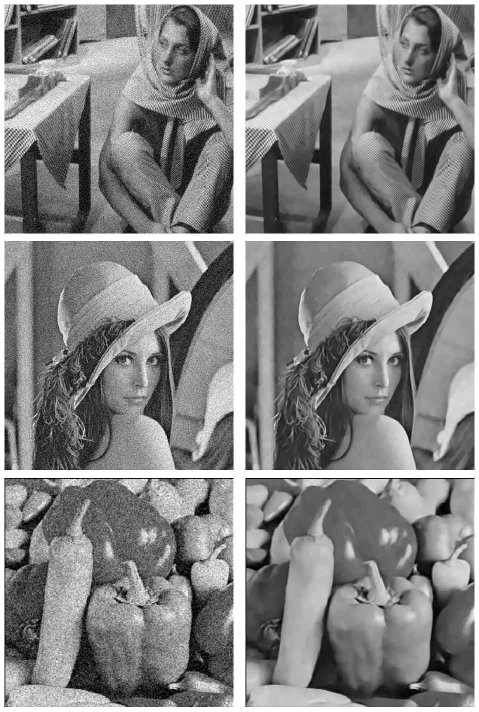




**D**

| | House $\sigma = 15$ | House $\sigma = 25$ | House $\sigma = 50$ |
|---|---|---|---|
| DE1 MTMKL | 34.29 | 31.88 | 28.08 |
| DE1 BPFA | 34.52 | 32.24 | 28.49 |
| DE1 GSC | 33.78 | 32.01 | 28.35 |
| DE1 GSVAE$^{\text{lin}}$ | 29.39$^{\varnothing}$ | 28.32$^{\varnothing}$ | 25.73$^{\varnothing}$ |
| DE1 EBSC | 33.66$^{\varnothing}$ | 32.40$^{\varnothing}$ | 28.98$^{\varnothing}$ |
| DE1 ES3C | **34.90**$^{\varnothing}$ | **33.15**$^{\varnothing}$ | **29.83**$^{\varnothing}$ |
| DE2 KSVD | 34.32$^{\varnothing}$ | 32.15$^{\varnothing}$ | 27.95$^{\varnothing}$ |
| DE2 LSSC | **35.35**$^{\varnothing}$ | 33.15$^{\varnothing}$ | 30.04$^{\varnothing}$ |
| DE2 BM3D | 34.94 | 32.86 | 29.37 |
| DE2 WNNM | 35.13 | **33.22** | **30.33** |
| DE5 EPLL | 34.17$^{\dagger}$ | 32.17$^{\dagger}$ | 29.12$^{\dagger}$ |
| DE6 MLP | n/a | 32.56 | 29.64 |
| DE6 DnCNN-B | 34.93 | 33.05 | 30.02 |
| DE6 TNRD | 34.55 | 32.54 | 29.48 |
| DE6 MemNet | 35.10 | 33.25 | 30.70 |
| DE6 FFDNet | 35.01 | 33.27 | 30.43 |
| DE6 DPDNN | **35.40** | **33.54** | **31.04** |
| DE6 BDGAN | 34.57 | 33.28 | 30.61 |
| DE6 BRDNet | 35.27 | 33.41 | 30.53 |

Figure 5: Denoising results for the "House" image (clean original depicted in **A**). **B** Performance comparison for AWG noise with $\sigma = 50$ under controlled conditions (see text for further details). **D** Performance comparison w.r.t. to state-of-the-art denoising approaches and standard baselines using different optimized hyperparameters (see text for further details). Grouping and labeling according to Table 2; in each group the highest PSNR value is marked bold. Numbers marked with $^{\varnothing}$correspond to averages over multiple independent realizations of the experiment using different realizations of the noise (see text for further details). For EPLL, model selection was performed based on the learning objective of the algorithm ($^{\dagger}$markers; personal communication with D. Zoran). Panel **C** illustrates the reconstructed images obtained with ES3C in the run with the highest PSNR value.



| | Barbara $\sigma = 25$ | Lena $\sigma = 25$ | Peppers $\sigma = 25$ |
|---|---|---|---|
| DE1 BPFA | 29.88 | 31.63 | 30.00 |
| DE1 GSVAE$^{\text{lin}}$ | 25.12$^{\varnothing}$ | 28.01$^{\varnothing}$ | 25.53$^{\varnothing}$ |
| DE1 EBSC | 28.38$^{\varnothing}$ | 30.88$^{\varnothing}$ | 28.93$^{\varnothing}$ |
| DE1 ES3C | **30.19**$^{\varnothing}$ | **31.95**$^{\varnothing}$ | **30.27**$^{\varnothing}$ |
| DE2 KSVD | 29.60$^{\varnothing}$ | 31.32$^{\varnothing}$ | 29.73$^{\varnothing}$ |
| DE2 LSSC | 30.47$^{\varnothing}$ | 31.87$^{\varnothing}$ | 30.21$^{\varnothing}$ |
| DE2 BM3D | 30.72 | 32.08 | 30.16 |
| DE2 WNNM | **31.24** | **32.24** | **30.42** |
| DE5 EPLL | 28.61$^{\dagger}$ | 31.73$^{\dagger}$ | 30.17$^{\dagger}$ |
| DE6 MLP | 29.21 | 32.12 | 30.25 |
| DE6 DnCNN-B | 29.69 | 32.42 | 30.84 |
| DE6 TNRD | 29.41 | 32.00 | 30.55 |
| DE6 MemNet | 29.98 | 32.51 | 30.87 |
| DE6 FFDNet | 29.98 | 32.59 | 30.79 |
| DE6 DPDNN | 30.30 | 32.69 | 30.90 |
| DE6 BDGAN | 29.95 | **32.77** | 30.88 |
| DE6 BRDNet | **30.34** | 32.65 | **31.04** |

Figure 6: Denoising results for "Barbara", "Lena" and "Peppers". The table shows a performance comparison w.r.t. to state-of-the-art denoising approaches and standard baselines using different optimized hyperparameters (see text for further details). Grouping and labeling according to Table 2; in each group the highest PSNR value is marked bold. Numbers marked with $^{\varnothing}$correspond to averages over multiple independent realizations of the experiment using different realizations of the noise (see text for further details). For EPLL, model selection was performed based on the learning objective of the algorithm ($^{\dagger}$markers; personal communication with D. Zoran). On the left we illustrate the reconstructed images obtained with ES3C in the run with the highest PSNR value.

seem natural to report a single PSNR value. For stochastic methods, reporting the best observed value is instructive about how well a given method can in principle perform. In practice, however, the run with highest PSNR value is less relevant than the average PSNR as selecting the run with best PSNR usually requires ground-truth (compare discussion in Appendix E). For KSVD and LSSC as well as for GSVAE$^{\text{lin}}$, EBSC and ES3C, the PSNR values listed in Figures 5 D and 6 correspond to averages. For the two former approaches, averages were obtained from five runs of the algorithms with different noise realizations (standard deviations were reported to be negligibly small; see Elad and Aharon, 2006, Mairal et al., 2009). For GSVAE$^{\text{lin}}$, EBSC and ES3C, we performed three runs of the algorithms with different noise realizations (observed standard deviations of the resulting PSNRs were not larger than 0.06 dB for EBSC and ES3C and 0.23 dB for GSVAE$^{\text{lin}}$; compare Table 3).

Considering Figures 5 D and 6, it can be observed that the performance tends to increase the more a-priori information is used by an algorithm (as can be expected). Feed-forward DNNs report overall the best PSNR values but also use the most a-priori information (see category DE6, Table 2). In the "zero-shot" category with no a-priori information (DE1), ES3C shows the best performance (Figure 5 D). ES3C significantly increases the state-of-the-art PSNR values across all investigated settings. EBSC is very competitive to previous approaches in the "House" benchmark (but a bit less so for the images of Figure 6). While being based on the same data model as EBSC, GSVAE$^{\text{lin}}$ was observed to result in much lower PSNRs than EBSC. Again, we observed a much denser encoding for GSVAE$^{\text{lin}}$ compared to other approaches used for the experiments. For the here considered benchmarks, such dense codes do not seem advantageous (while we remark that GSVAE$^{\text{lin}}$ and other GSVAE versions are observed to perform well on other data, see Appendix F.3 for details).

Notably, the spike-and-slab based ES3C approach is still competitive when compared to algorithms in categories DE2 and DE5 that use more a-priori information, but it is outperformed by some of these methods, especially at lower noise levels (Figures 5 D and 6). Differences in performance between the categories get smaller for larger and more complex images (Figure 6). For the image "Barbara", for instance, ES3C outperforms all DNNs except DPDNN and BRDNet, which are slightly better, while the best result is obtained for the image specific non-local method WNNM.

PSNR values achieved by the previous methods of BPFA or MTMKL have notably proven to be difficult to further improve. Only systems using more a-priori information (such as WNNM in category DE2 or DNNs in category DE6) have more recently reported higher PSNR values (also see discussion of difficult to improve BM3D values by Gu et al., 2014). In category DE1, the only more recent competitive approach is a Bayesian version of K-SVD (BKSVD; Serra et al., 2017). By extending K-SVD using Bayesian methodology, the noise level can be estimated from data. The BKSVD approach can thus be applied without a-priori information on the noise level, which makes it applicable in the "zero-shot" category (see Serra et al., 2017, for a discussion). PSNR values for BKSVD reported by Serra et al. (2017) are computed on downscaled images, and are therefore not directly comparable to those of Figures 5 D and 6. However, comparisons of BKSVD with MTMKL, K-SVD and BPFA reported by Serra et al. (2017) show that BKSVD performs competitively, especially for lower noise levels. For higher noise levels, though, (e.g. $\sigma = 25$ for "Lena" and "Peppers") BPFA results in higher PSNR values (and in similar PSNRs for "Barbara"). As ES3C outperforms BPFA for $\sigma = 25$ on "Barbara", "Lena" and "Peppers" on the full resolution images, BKSVD also does not seem to establish a new state-of-the-art in the DE1 category. We can, therefore, conclude that ES3C establishes a novel state-of-the-art in the "zero-shot" category since a relatively long time.

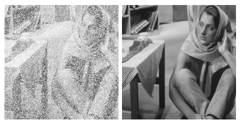
### 4.3 Image Inpainting

The final benchmark we used is image inpainting, i.e. the reconstruction of missing values in images. A standard setting for this task is the restoration of images that have been artificially degraded by uniformly randomly deleting pixels. Here we considered the standard test images "Barbara", "Cameraman", "Lena", "House" and "Castle" as examples for gray-scale and RGB images. For denoising, ES3C consistently performed better than EBSC, and the same we observed for inpainting. The performance difference between ES3C and EBSC on inpainting was, however, larger (sometimes several dB). We therefore focus on ES3C for comparison as EBSC is not competitive to the best performing methods we compare to.

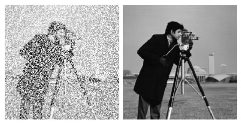
For inpainting with ES3C, we employed the same partitioning and averaging scheme as described in Section 4.2. For the RGB image, ES3C was applied jointly to all color channels, as this had been reported to be more beneficial than training individual models on different color layers (Mairal et al., 2008). We trained the model and restored missing values exclusively based on the incomplete data which we aimed to restore (following, e.g., Fadili and Starck, 2005; Little and Rubin, 1987); uncorrupted pixels were not modified. Details about the missing value estimator applied by ES3C are provided in Section 2.3 and in Appendix C. For inpainting, we used a SSSC model with fixed $\vec{\mu} = \vec{1}$ and $\Psi = \mathbb{1}$ as these parameters were observed to grow exponentially during learning which led to numerical instabilities of the training process. For each test scenario, we performed three runs of ES3C and used a different realization of missing values in each run (EVO hyperparameters are listed in Table 4 in Appendix F.2). Compared to the denoising experiments (Section 4.2.3), we observed for inpainting slightly larger variations of the resulting PSNR values (standard deviations were smaller or equal 0.18 dB; results of individual runs listed in Table 3 in Appendix F).

The inpainting results for "Barbara", "Cameraman", "Lena" (each 50% missing values) and "House" and "Castle" (each 50% and 80% missing values) are depicted in Figure 7 A-E . The reference PSNR values are taken from the original publications except for NL and FoE for which we cite the numbers as reported by Yu et al. (2012). In general, inpainting is a less wide-spread benchmark than denoising presumably because additional estimation routines have to be developed and missing values pose, more generally, a considerable challenge to many approaches. DNN-based approaches have to define, for instance, how a missing pixel of the input image is treated (which can be challenging, e.g., if the positions of the missing pixels are different from patch to patch as for this benchmark). The challenge is faced by approaches such as GSVAE (and VAE approaches in general) because DNNs are used for VAE encoders. Considering Figure 7 A-E, first observe that performance differences between categories are larger than for denoising. Compared to the best performing approaches in the literature, ES3C shows to be competitive in many settings. On the "Castle" benchmark, ES3C performs better than all other approaches in the literature, with the exception of PLE, which uses the noise level as a-priori information. For the "House" benchmark, ES3C establishes a novel state-of-the-art in general: it performs better than all other algorithms for 50% lost pixels and equal to PLE for 80% lost pixels (i.e., within PSNR standard deviation

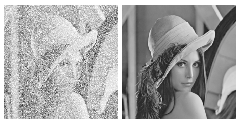
**A** Barbara



**B** Cameraman

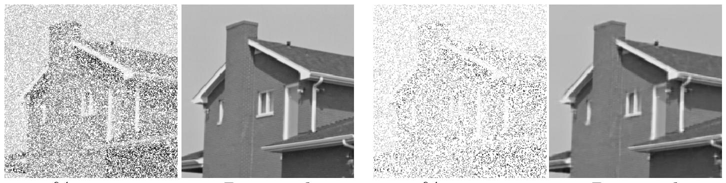


**C** Lena



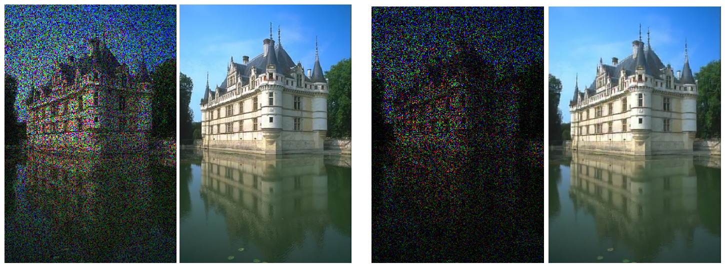
| | Barbara | C.man | Lena |
|---|---|---|---|
| IN1 BPFA | 33.17 | 28.90 | 36.94 |
| IN1 DIP | 32.22 | 29.80 | 36.16 |
| IN1 ES3C | **35.44**$^{\varnothing}$ | **30.69**$^{\varnothing}$ | **37.54**$^{\varnothing}$ |
| IN2 NL | 36.40 | n/a | **37.82** |
| IN2 PLE | **37.03** | n/a | 37.78 |
| IN4 FoE | 29.47 | n/a | 36.66 |

**D** House



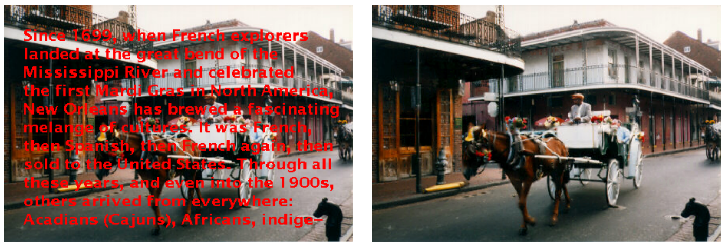
| | 50% | 80% |
|---|---|---|
| IN1 BPFA | 38.02 | 30.12 |
| IN1 DIP | 39.16 | n/a |
| IN1 ES3C | **39.59**$^{\varnothing}$ | **33.06**$^{\varnothing}$ |
| IN2 NL | **39.30** | 32.87 |
| IN2 PLE | 38.97 | **33.05** |
| IN4 FoE | 37.99 | 31.28 |

**E** Castle



| | 50% | 80% |
|---|---|---|
| IN1 MTMKL | n/a | 28.94 |
| IN1 BPFA | 36.45 | 29.12 |
| IN1 ES3C | **38.23**$^{\varnothing}$ | **29.66**$^{\varnothing}$ |
| IN2 PLE | 38.34 | 30.07 |
| IN5 cKSVD | n/a | 29.65 |
| IN6 IRCNN | n/a | 28.74 |

**F** New Orleans


Since 1699, when French explorers
landed at the great bend of the
Mississippi River and celebrated
the first Mardi Gras in North America,
New Orleans has brewed a fascinating
melange of cultures. It was French,
then Spanish, then French again, then
sold to the United States. Through all
these years, and even into the 1900s,
others arrived from everywhere:
Acadians (Cajuns), Africans, indige-
Corrupted
Restored

| | |
|---|---|
| IN1 ES3C | 32.17 |
| IN4 FoE | **32.39** |
| IN4 MRF | 31.69 |
| IN5 cKSVD | 32.45 |
| IN6 NLRMRF | **32.59** |
| IN6 IRCNN | 30.95 |

Figure 7: Inpainting results for “Barbara”, “Cameraman”, “Lena”, “House”, “Castle” and “New Orleans”. For the performance comparison, different approaches are grouped and labeled according to Table 2. In each group the highest PSNR value is marked bold. Numbers marked with $^{\varnothing}$in **A** to **E** correspond to averages over multiple independent runs of the experiment using different realizations of missing values (see text for details; the performance of ES3C in the individual runs is reported in Table 3 in Appendix F). On the left, we illustrate the reconstructed images obtained with ES3C in the run with the highest PSNR value.

of ES3C). This novel state-of-the-art is notably reached without requiring clean training data or a-priori knowledge about noise level or sparsity (ES3C requires to know which pixels are missing; compare Table 2 and Appendix E).

Finally, we also report inpainting performance on the well-known "New Orleans" inpainting benchmark (Figure 7 F left-hand-side). The benchmark serves as an inpainting example where pixels are lost non-randomly. For the benchmark, the original image is artificially corrupted by adding overlaid text, which is then removed by applying inpainting. Compared to inpainting with randomly missing values (e.g., the benchmarks in Figure 7 A-E), the "New Orleans" image contains relatively large contiguous regions of missing values. For our measurements, we used a publicly available data set[4]. The result of the experiment is illustrated in Figure 7 F. Reference PSNR values are taken from respective original publications with exceptions of FoE and MRF for which we use values reported by Sun and Tappen (2011). Due to extensive runtimes, we report the result of a single run of ES3C for this benchmark (see Appendix F for further details; EVO hyperparameters listed in Table 4). As can be observed, ES3C also performs well on this task. We observed higher PSNR values for ES3C than for MRF and IRCNN. Values for FoE and cKSVD are higher, but more a-priori information is also used. Especially FoE is less sensitive to larger areas of lost pixels than ES3C (compare the lower performance of FoE in Figure 7 A-D); presumably FoE can make better use of larger contexts. The state-of-the-art on this benchmark is established by NLRMRF.

## 5. Discussion

Efficient approximate inference is of crucial importance for training expressive generative models. Consequently, there exists a range of different methods with different assumptions and different features. As most generative models can only be trained using approximations, and usually many locally optimal solutions exist, the question of how well a generative model can potentially perform on a given task is difficult to answer. Sophisticated, mathematically grounded approaches such as sampling or variational optimization have been developed in order to derive sufficiently precise and efficient learning algorithms. As a contribution of this work, we have proposed a general purpose variational optimization that directly and intimately combines evolutionary algorithms (EAs) and EM. Considering Algorithm 1, EAs are an integral part of variational EM where they address the key optimization problem (the variational loop) arising in the training of directed generative models. The EVO algorithm is defined by a set of optimization hyperparameters for the EA. Given the hyperparameters, EVO is applicable to a given generative model solely by using the model's joint probability $p(\vec{s}, \vec{y} \mid \Theta)$. No analytical derivations are required to realize variational optimization.

4. We downloaded the clean and the corrupted image from `https://lear.inrialpes.fr/people/mairal/resources/KSVD_package.tar.gz` and the text mask from `https://www.visinf.tu-darmstadt.de/media/visinf/software/foe_demo-1_0.zip`; all images were provided as PNG files. We verified that the corrupted image was consistent w.r.t. the mask and the original image contained in the data set: We applied the mask to the original image and compared the unmasked parts to the unmasked parts of the corrupted image. These were identical. After filling masked areas with red (i.e., replacing the corresponding pixels with (255,0,0) in RGB space) we measured a PSNR value of the corrupted image of 13.48 dB. This value slightly deviates from numbers reported in other studies; Chaudhury and Roy, for example, measured a PSNR of 15.05 dB.

*Relation to Other Approximate Inference Approaches.* To show large-scale applicability of EVO, we considered high-dimensional image data and generative models with large (combinatorial) state spaces. Aside from the elementary generative models of Noisy-OR Bayes Nets (Singliar and Hauskrecht, 2006; Jernite et al., 2013; Rotmensch et al., 2017) and binary sparse coding (Haft et al., 2004; Henniges et al., 2010), we used spike-and-slab sparse coding (SSSC) as a more expressive example. The SSSC data model has been of considerable interest due to its strong performance for a range of tasks (Zhou et al., 2009, 2012; Titsias and Lázaro-Gredilla, 2011; Goodfellow et al., 2012; Sheikh et al., 2014). However, its properties require sophisticated probabilistic approximation methods for parameter optimization. Essentially three types of approximations have previously been applied to train the model: mean field (Titsias and Lázaro-Gredilla, 2011; Goodfellow et al., 2012), truncated posterior approximations (Sheikh et al., 2014), and MCMC sampling (Zhou et al., 2009, 2012; Mohamed et al., 2012). If we consider the BSC data model as a boundary case of a variational autoencoder with categorical latents (Jang et al., 2017), we can also include Gumbel-Softmax-based VAE optimization as a fourth optimization approach. EVO shares features with most of these previous approaches: like work by Sheikh et al. (2014), EVO uses truncated posterior distributions to approximate full posteriors, which contrasts with factored posterior approximations (i.e., mean field) used by Titsias and Lázaro-Gredilla (2011); Goodfellow et al. (2012). However, like work by Titsias and Lázaro-Gredilla, 2011, and Goodfellow et al., 2012, EVO is a fully variational EM algorithm which is guaranteed to monotonically increase the variational lower bound. This contrasts with Sheikh et al. (2014) or Shelton et al. (2017) who used feed-forward estimates to construct approximate posteriors (without a guarantee to monotonically increase a variational bound). Furthermore, like sampling-based approaches (Zhou et al., 2009, 2012; Mohamed et al., 2012), EVO computes estimates of posterior expectations based on a finite set of latent states. However, unlike for sampling approximations, these states are not samples but are themselves variational parameters that are evolved using EAs (with a fitness defined by a variational bound).

VAE optimization using the Gumbel-Softmax trick is the most different to EVO and to all previous approaches. The investigated GSVAE algorithms use sampling-based approximations such as Zhou et al. (2009, 2012) or Mohamed et al. (2012), and they optimize a variational lower bound such as ES3C and EBSC. GSVAE is also similar to GSC as both methods use deterministic feed-forward mappings in their definition of variational distributions (amortization). However, GSVAE is, in contrast to the previously discussed methods, a further development of standard VAE optimization. More concretely, it uses a DNN for encoding and a "softening" (i.e. annealing) of discrete latents (the Gumbel-Softmax trick) in order to apply standard VAE optimization based on reparameterization and gradient ascent. Standard VAE optimization was, on the other hand, originally defined for latents with Gaussian prior, which typically gives rise to a dense code (and such codes will be of relevance for the discussion of denoising experiments below).

*Denoising and Inpainting Benchmarks and Comparison with Other Approaches.* For comparison with a range of other methods, we used standard denoising and inpainting benchmarks and evaluated EVO for the SSSC model (termed *ES3C*) and EVO for the BSC model (termed *EBSC*). Compared to competing methods in the "zero-shot" category (see category DE1 and IN1 in Table 2), we observed ES3C to achieve the highest PSNR values on all of the benchmarks of Figures 5, 6, and 7 that we performed. Furthermore, ES3C also shows improvements w.r.t. methods that use more a-priori information in many test scenarios:

- The denoising results (Figures 5 and 6) show that ES3C can outperform state-of-the-art sparse coding and non-local approaches including, e.g., LSSC and BM3D, which require ground-truth noise level information. ES3C also improves on the popular GMM-based EPLL method, which additionally uses clean training data. Furthermore, ES3C shows improvements compared to a number of recent, intricate DNNs which are tailored to image denoising and require (large amounts of) clean data for training. For instance, on the "Barbara" image (Figure 6), ES3C is only outperformed by the very recent networks DPDNN (Dong et al., 2019) and BRDNet (Tian et al., 2020). At the same time, we note that the image noise for the benchmarks is Gaussian. Gaussian noise is commonly encountered in real data, but it also makes the SSSC model well suited for this task in general.

- For image inpainting, we observed ES3C to provide state-of-the-art results also in general (i.e. across all categories) in some settings: compared to the best performing algorithms on the "House" benchmark for inpainting, ES3C achieves the highest PSNR value (Figure 7 D, best value shared with PLE for 80% lost pixels). The inpainting benchmark is, however, much less frequently applied than its denoising counterpart for the same image because a treatment of missing data is required.

The observation that probabilistic approaches such as SSSC-based algorithms perform well on the inpainting benchmarks of Figure 7 A-E may not come as too much of a surprise. In the case of randomly missing values, the information provided by relatively small patches contains valuable information which an SSSC model can leverage well. Larger areas of missing values, as encountered in the "New Orleans" benchmark, require larger contexts, whose information may be better leveraged by less local methods such as Markov Random Fields (Figure 7 F).

To answer the question of why EVO results in improvements, direct comparison with other optimization methods for the same or similar models are most instructive (Figure 5 B). In general, the type of variational optimization can have a strong influence on the type of generative fields that are learned, e.g., for the SSSC model. Mean field methods (i.e., fully factored variational distributions) have a tendency to bias generative fields to be orthogonal (Mackay, 2001; Ilin and Valpola, 2005). Such effects are also discussed for other generative models (e.g. Turner and Sahani, 2011; Vértes and Sahani, 2018). Biases introduced by a variational approximation can thus lead to increasingly suboptimal representations (see, e.g., Titsias and Lázaro-Gredilla, 2011, Sheikh et al., 2014, for discussions). Sampling is in general more flexible but practical implementations may (e.g., due to insufficient mixing) also bias parameter learning towards learning posteriors with single modes. The denoising benchmark under controlled conditions shows that the sampling-based BPFA approach (Zhou et al., 2012) performs, for instance, better than the mean field approach MTMKL (Titsias and Lázaro-Gredilla, 2011) and the truncated EM approach GSC (Sheikh et al., 2014); but BPFA performs worse than ES3C, which is based on EVO. Likewise, also VAE-like training using Gumbel-Softmax is a specific optimization which impacts the learned representation. In our experiments, VAE-like optimization showed the strongest differences not only to EVO but also to all other approaches that were based on an SSSC or a BSC-like generative model.

In contrast to, e.g., MTMKL, GSC, ES3C and EBSC, the GSVAE approach resulted in a dense encoding of image patches. Concretely, instead of learning generative fields that resemble image structures such as edges or texture components, GSVAE learned fields with much less interpretable structures (and many of the learned fields represented relatively high spatial frequencies, see Figure 11). For reconstruction, GSVAE used large fractions of these fields (usually about half on average). In other words, GSVAE uses dense codes, while MTMKL, GSC, ES3C or EBSC use sparse codes for encoding (few fields are combined for reconstruction). Notably, ES3C or EBSC are by construction not constrained to learn a sparse code (parameters for the Bernoulli prior are learned). Neither is GSVAE by construction constrained to a dense code (prior parameters of GSVAE that correspond to a sparse code can be learned in principle). Still, in all applications to natural image data, ES3C and EBSC learned sparse codes; and in all applications to natural image data, GSVAE was observed to learn a dense code (for all GSVAE versions investigated, compare Appendix F.3). Comparison on denoising benchmarks clearly shows that the dense codes learned by GSVAE$^{\mathrm{lin}}$ are much less competitive than the sparse codes of the other approaches (Figures 5 and 6). At the same time, we remark that the used versions and implementations of GSVAE are competitive on other benchmarks. Compared to other methods in the literature (e.g. Park et al., 2019), we observed dense codes of GSVAE to result in good benchmarking performance on data sets with many images of whole objects, for instance. On the other hand, for such data, algorithms giving rise to sparse codes such as EBSC are not learning edge or texture like components and are observed to be less competitive (see Appendix F.3 for such controls).

The differences between algorithms with dense and sparse codes may be an important reason why deep generative models have not been reported to perform competitive in denoising. For the benchmarks in Figures 5 and 6, the recent BDGAN approach is, besides GSVAE, the only deep generative model. Except for BDGAN, neither for generative adversarial nets (GANs; Goodfellow et al., 2014) nor for GSVAE or other variational autoencoders competitive denoising or inpainting results are reported on these standard benchmarks (for GANs and VAEs, benchmarks other than those in Figures 5 to 7 are often considered; see Appendix F.3 for further discussion). For BDGAN, Zhu et al. (2019) report performances on standard denoising benchmarks; the algorithm shows competitive performance but, like feed-forward DNNs, requires large image corpora for training. At least in principle, GANs and VAEs are also applicable to the “zero-shot” category (DE1 and IN1 in Figures 5 - 7), however (and GSVAE represents one such example). The BDGAN approach uses large and intricate DNNs whose parameters are presumably difficult to train using just one image, which may explain why it was not applied in the “zero-shot” category. Recent work by, e.g. Shocher et al. (2018), explicitly discusses DNN sizes and the use of small DNNs for “zero-shot” super-resolution.

For VAEs, there is (except of the results reported here for GSVAE) neither data for the “zero-shot” category nor for the other categories available for the standard benchmarks we used (to the knowlege of the authors; but compare Prakash et al., 2021). A reason may be that (like for BDGAN) very intricate DNNs as well as sophisticated sampling and training methods are required to be competitive. Experiments with standard (Gaussian) VAE setups that we conducted did, e.g. for denoising, not result in PSNR values close to those reported in Figures 5 and 6. Another possible reason for the absence of competitive values for VAEs may, however, be related to the variational approximation used. VAEs for continuous data usually use Gaussians as variational distributions (Rezende et al., 2014; Kingma and Welling, 2014) that are in addition fully factored (i.e., mean field). The parameters of the VAE decoder may thus suffer from similar biasing effects as described for mean field approaches as used, e.g., by MTMKL (Titsias and Lázaro-Gredilla, 2011) for the SSSC model. That standard VAEs *do* show such biases has recently been pointed out, e.g., by Vértes and Sahani (2018). Furthermore, the dense codes learned by VAEs due to their usual Gaussian priors seem to lead to representations less suitable for denoising (but potentially well suited for other tasks). Even though sparse codes could be learned in principle (e.g., GSVAE can in principle learn sparse codes), the usual optimization methods for VAEs seem to favor dense codes.

Like other generative models, VAEs (which use DNNs as part of their encoders and decoders) are, consequently, likely to strongly profit from a more flexible encoding of variational distributions. In the context of deep learning, such flexible variational distributions have been suggested, e.g., in the form of normalizing flows (Rezende and Mohamed, 2015). In its standard version (Rezende and Mohamed, 2015) normalizing flows are making training significantly more expensive, however. Approximate learning using normalizing flows is centered around a sampling-based approximation of expectation values w.r.t. posteriors (Rezende and Mohamed, 2015; and also compare related approaches, e.g., Huang et al., 2018; Bigdeli et al., 2020). By following such a strategy, samples of increasingly complex variational distributions can be generated by successively transforming samples of an elementary distribution. The idea of successive transformations has also been used to directly parameterize the data distribution in what is now termed flow-based models (e.g. Dinh et al., 2017; Kingma and Dhariwal, 2018). An early example (Dinh et al., 2014) is a form of non-linear ICA (Hyvärinen and Pajunen, 1999). By using EAs, the EVO approach does, in contrast, follow a strategy that is notably very different from normalizing flows and other approaches. A consequence of these different strategies are very different sets of hyperparameters: EVO hyperparameters are centered around parameters for the EA such as population size as well as mutation, crossover and selection operators; hyperparameters of normalizing flows and related methods are centered around parameterized successive mappings (e.g., type of mapping, number of transformations) and (if applicable) hyperparameters for sampling. Furthermore, and most importantly for practical applications, the types of generative models addressed by EVO and normalizing flows are essentially complementary: EVO focuses on generative models with discrete latents while normalizing flows and others focus on models with continuous latents.

*Conclusion.* More generally, EVO can be regarded as an example of the significant potential in improving the accuracy of posterior approximations. Many current research efforts are focused on making parameterized models increasingly complex: usually an intricate DNN is used either directly (e.g., feed-forward DNNs/CNNs for denoising) or as part of a generative model. At least for tasks such as denoising or inpainting, our results suggest that improving the flexibility of approximation methods may be as effective in improving performance as increasing model complexity. While EVO shares the goal of a flexible and efficient posterior approximation with many other powerful and successful methods, it has a distinguishing feature: a direct link from variational optimization to a whole other research field focused on optimization: evolutionary algorithms. For generative models with discrete latents, new results in the field of EAs can consequently be leveraged to improve the training of future (elementary and complex) generative models.

## Acknowledgments

We acknowledge support by the German Research Foundation (DFG) under Grant 352015383 (SFB 1330, B2). Furthermore, we acknowledge the support of Enrico Guiraud through a Wolfgang Gentner scholarship of the German Federal Ministry of Education and Research (BMBF, Grant 13E18CHA), and support through computing infrastructure by the HPC Cluster CARL of Oldenburg University (DFG, Grant INST 184/157-1 FUGG) and the North-German Supercomputing Alliance (HLRN, Project nim00006).

## Appendix A. Log-Pseudo Joint Formulation

For efficient computation of log-joint probabilities $\log p\,(\vec{y}, \vec{s} \mid \Theta)$ (which are required to compute truncated expectations (3) and the variational lower bound (4)), we introduce the following reformulation. When computing $\log p\,(\vec{y}, \vec{s} \mid \Theta)$ for a concrete model, we identify the terms that only depend on the model parameters $\Theta$ (and not on the data points $\vec{y}$ and the variational states $\vec{s}$). These terms, denoted $\mathrm{C}(\Theta)$ in the following, only need to be evaluated once when computing $\log p\,(\vec{y}, \vec{s} \mid \Theta)$ for different data points and for different variational states. We refer to the remaining terms that depend on both $\vec{s}$, $\vec{y}$ and $\Theta$ as *log-pseudo joint* and denote this quantity by $\widetilde{\log p}(\vec{s}, \vec{y} \mid \Theta)$. The log-joint probability $\log p\,(\vec{y}, \vec{s} \mid \Theta)$ can then be reformulated as:

$$\log p\,(\vec{y}, \vec{s} \mid \Theta) = \widetilde{\log p}(\vec{s}, \vec{y} \mid \Theta) + \mathrm{C}(\Theta). \tag{17}$$

The concrete expressions of $\mathrm{C}(\Theta)$ and $\widetilde{\log p}(\vec{s}, \vec{y} \mid \Theta)$ for the models introduced in Section 3.1 are listed below.

*NOR.*

$$\mathrm{C}(\Theta) = \sum_{h=1}^{H} \log(1 - \pi_h), \tag{18}$$

$$\widetilde{\log p}(\vec{s}, \vec{y} \mid \Theta) = \sum_{d=1}^{D} \Big( y_d \log\big(N_d(\vec{s})\big) + (1 - y_d) \log\big(1 - N_d(\vec{s})\big) \Big) + \sum_{h=1}^{H} s_h \log\Big(\frac{\pi_h}{1 - \pi_h}\Big), \tag{19}$$

$$N_d(\vec{s}) = 1 - \prod_{h=1}^{H} (1 - W_{dh} s_h), \tag{20}$$

with the special exception of $\vec{s} = \vec{0}$ for which

$$\widetilde{\log p}(\vec{s} = \vec{0}, \vec{y}\,) = \begin{cases} 0 & \text{if } \vec{y} = \vec{0}, \\ -\inf & \text{otherwise.} \end{cases} \tag{21}$$

For practical computations, we set $\widetilde{\log p}$ to an arbitrarily low value rather than the floating point infinite representation.

*BSC.*

$$\mathrm{C}(\Theta) = H \log(1-\pi) - \frac{D}{2}\log(2\pi\sigma^2), \tag{22}$$

$$\widetilde{\log} p(\vec{s}, \vec{y} \mid \Theta) = \log\left(\frac{\pi}{1-\pi}\right)\sum_{h=1}^{H} s_h - \frac{1}{2\sigma^2}(\vec{y} - W\vec{s})^{\mathrm{T}}(\vec{y} - W\vec{s}). \tag{23}$$

*SSSC.*

$$\mathrm{C}(\Theta) = \sum_{h=1}^{H}\log(1-\pi_h) - \frac{D}{2}\log(2\pi), \tag{24}$$

$$\widetilde{\log} p(\vec{s}, \vec{y} \mid \Theta) = \sum_{h=1}^{H} s_h \log\left(\frac{\pi_h}{1-\pi_h}\right) - \frac{1}{2}\log|C_{\vec{s}}| - \frac{1}{2}(\vec{y} - \tilde{W}_{\vec{s}}\vec{\mu})^{\mathrm{T}} C_{\vec{s}}^{-1}(\vec{y} - \tilde{W}_{\vec{s}}\vec{\mu}). \tag{25}$$

## Appendix B. Sparsity-Driven Bitflips

When performing sparsity-driven bitflips, we flip each bit of a particular child $\vec{s}$ with probability $p_0$ if it is 0, with probability $p_1$ otherwise. We call $p_{\mathrm{bf}}$ the average probability of flipping any bit in $\vec{s}$. We impose as constraints on $p_0$ and $p_1$ that $p_1 = \alpha p_0$ for some constant $\alpha$ and that the average number of "on" bits after mutation is set to $\widetilde{s}$. This yields the following expressions for $p_0$ and $p_1$:

$$p_0 = \frac{H p_{\mathrm{bf}}}{H + (\alpha - 1)|\vec{s}\,|}, \qquad \alpha = \frac{(H - |\vec{s}\,|)\cdot(H p_{\mathrm{bf}} - \widetilde{s} + |\vec{s}\,|)}{(\widetilde{s} - |\vec{s}\,| + H p_{\mathrm{bf}})|\vec{s}\,|}. \tag{26}$$

Trivially, random uniform bitflips correspond to the case $p_0 = p_1 = p_{\mathrm{bf}}$. For our numerical experiments (compare Section 3.2), we chose $\widetilde{s}$ based on the sparsity learned by the model (we set $\widetilde{s} = \sum_{h=1}^{H}\pi_h$ for NOR and SSSC and $\widetilde{s} = H\pi$ for BSC). We furthermore used an average bitflip probability of $p_{\mathrm{bf}} = \frac{1}{H}$.

## Appendix C. Data Estimator

*Models with binary latents and continuous observables.* Consider the posterior predictive distribution

$$p(\vec{y}^{\,\mathrm{est}} \mid \vec{y}^{\,\mathrm{obs}}, \Theta) = \sum_{\vec{s}} p(\vec{y}^{\,\mathrm{est}}, \vec{s} \mid \vec{y}^{\,\mathrm{obs}}, \Theta) = \sum_{\vec{s}} p(\vec{y}^{\,\mathrm{est}} \mid \vec{s}, \Theta)\, p(\vec{s} \mid \vec{y}^{\,\mathrm{obs}}, \Theta). \tag{27}$$

The second step in (27) exploits the fact that $\vec{y}^{\,\mathrm{est}}$ and $\vec{y}^{\,\mathrm{obs}}$ are conditionally independent given the latents. In order to infer the value of $y_d^{\mathrm{est}}$ we will take expectations w.r.t. $p(\vec{y}^{\,\mathrm{est}} \mid \vec{y}^{\,\mathrm{obs}})$:

$$\langle y_d^{\mathrm{est}}\rangle_{p(\vec{y}^{\,\mathrm{est}} \mid \vec{y}^{\,\mathrm{obs}}, \Theta)} = \int\limits_{\vec{y}^{\,\mathrm{est}}} y_d^{\mathrm{est}}\, p(\vec{y}^{\,\mathrm{est}} \mid \vec{y}^{\,\mathrm{obs}}, \Theta)\,\mathrm{d}\vec{y}^{\,\mathrm{est}} = \int\limits_{y_d^{\mathrm{est}}} y_d^{\mathrm{est}}\, p(y_d^{\mathrm{est}} \mid \vec{y}^{\,\mathrm{obs}}, \Theta)\,\mathrm{d}y_d^{\mathrm{est}}. \tag{28}$$

The second step in (28) follows from marginalizing over $\vec{y}^{\,\mathrm{est}} \backslash y_d^{\mathrm{est}}$. Equations 27 and 28 can be combined:

$$\langle y_d^{\mathrm{est}} \rangle_{p(\vec{y}^{\,\mathrm{est}} \,|\, \vec{y}^{\,\mathrm{obs}}, \Theta)} = \int\limits_{y_d^{\mathrm{est}}} y_d^{\mathrm{est}} \sum_{\vec{s}} p(y_d^{\mathrm{est}} \mid \vec{s}, \Theta)\, p(\vec{s} \mid \vec{y}^{\,\mathrm{obs}}, \Theta)\, \mathrm{d}y_d^{\mathrm{est}} \tag{29}$$

$$= \sum_{\vec{s}} \int\limits_{y_d^{\mathrm{est}}} y_d^{\mathrm{est}}\, p(y_d^{\mathrm{est}} \mid \vec{s}, \Theta)\, \mathrm{d}y_d^{\mathrm{est}}\, p(\vec{s} \mid \vec{y}^{\,\mathrm{obs}}, \Theta) \tag{30}$$

$$= \sum_{\vec{s}} \langle y_d^{\mathrm{est}} \rangle_{p(y_d^{\mathrm{est}} \,|\, \vec{s}, \Theta)}\, p(\vec{s} \mid \vec{y}^{\,\mathrm{obs}}, \Theta) \tag{31}$$

$$= \Big\langle \langle y_d^{\mathrm{est}} \rangle_{p(y_d^{\mathrm{est}} \,|\, \vec{s}, \Theta)} \Big\rangle_{p(\vec{s} \,|\, \vec{y}^{\,\mathrm{obs}}, \Theta)}. \tag{32}$$

The inner expectation in (32) is for the example of the BSC model (11) given by $\langle y_d \rangle_{p(y_d \,|\, \vec{s}, \Theta)} = \vec{W}_d \vec{s}$ with $\vec{W}_d$ denoting the $d$-th row of the matrix $W$. With this, (32) takes for the BSC model the following form

$$\langle y_d^{\mathrm{est}} \rangle_{p(\vec{y}^{\,\mathrm{est}} \,|\, \vec{y}^{\,\mathrm{obs}}, \Theta)} = \vec{W}_d \langle \vec{s} \rangle_{p(\vec{s} \,|\, \vec{y}^{\,\mathrm{obs}}, \Theta)}. \tag{33}$$

The estimator (33) can be efficiently computed by (i) approximating the full posterior distribution of the latents by using truncated posteriors (2) and by (ii) approximating the expectation w.r.t. the full posterior by applying truncated expectations (3).

*Models with binary-continuous latents and continuous observables.* The upper derivation can be extended to be applicable to models with binary-continuous latents. Following the same line of reasoning, we again start with the posterior predictive distribution $p(\vec{y}^{\,\mathrm{est}} \mid \vec{y}^{\,\mathrm{obs}})$:

$$p(\vec{y}^{\,\mathrm{est}} \mid \vec{y}^{\,\mathrm{obs}}, \Theta) = \sum_{\vec{s}} \int\limits_{\vec{z}} p(\vec{y}^{\,\mathrm{est}}, \vec{s}, \vec{z} \mid \vec{y}^{\,\mathrm{obs}}, \Theta)\, \mathrm{d}\vec{z} \tag{34}$$

$$= \sum_{\vec{s}} \int\limits_{\vec{z}} p(\vec{y}^{\,\mathrm{est}} \mid \vec{s}, \vec{z}, \Theta)\, p(\vec{s} \mid \vec{y}^{\,\mathrm{obs}}, \Theta)\, p(\vec{z} \mid \vec{s}, \vec{y}^{\,\mathrm{obs}}, \Theta)\, \mathrm{d}\vec{z}. \tag{35}$$

We then follow the steps from equations 29 to 32, i.e. we take expectations w.r.t. the posterior predictive distribution (35):

$$\langle y_d^{\mathrm{est}} \rangle_{p(\vec{y}^{\,\mathrm{est}} \,|\, \vec{y}^{\,\mathrm{obs}}, \Theta)} = \int\limits_{y_d^{\mathrm{est}}} y_d^{\mathrm{est}} \sum_{\vec{s}} \int\limits_{\vec{z}} p(y_d^{\mathrm{est}} \mid \vec{s}, \vec{z}, \Theta)\, p(\vec{s}, \vec{z} \mid \vec{y}^{\,\mathrm{obs}}, \Theta)\, \mathrm{d}\vec{z}\, \mathrm{d}y_d^{\mathrm{est}} \tag{36}$$

$$= \sum_{\vec{s}} \int\limits_{\vec{z}} \int\limits_{y_d^{\mathrm{est}}} y_d^{\mathrm{est}}\, p(y_d^{\mathrm{est}} \mid \vec{s}, \vec{z}, \Theta)\, \mathrm{d}y_d^{\mathrm{est}}\, p(\vec{s}, \vec{z} \mid \vec{y}^{\,\mathrm{obs}}, \Theta)\, \mathrm{d}\vec{z} \tag{37}$$

$$= \sum_{\vec{s}} \int\limits_{\vec{z}} \langle y_d^{\mathrm{est}} \rangle_{p(y_d^{\mathrm{est}} \,|\, \vec{s}, \vec{z}, \Theta)}\, p(\vec{s}, \vec{z} \mid \vec{y}^{\,\mathrm{obs}}, \Theta)\, \mathrm{d}\vec{z} \tag{38}$$

$$= \Big\langle \langle y_d^{\mathrm{est}} \rangle_{p(y_d^{\mathrm{est}} \,|\, \vec{s}, \vec{z}, \Theta)} \Big\rangle_{p(\vec{s}, \vec{z} \,|\, \vec{y}^{\,\mathrm{obs}}, \Theta)}. \tag{39}$$

For the example of the SSSC model (12)-(13), the inner expectation in (39) is given by $\langle y_d \rangle_{p(y_d \,|\, \vec{s},\vec{z},\Theta)} = \vec{W}_d\,(\vec{s} \odot \vec{z}\,)$ s.t. equation 39 takes for the SSSC model the following form:

$$\langle y_d^{\mathrm{est}} \rangle_{p(\vec{y}^{\,\mathrm{est}} \,|\, \vec{y}^{\,\mathrm{obs}},\Theta)} = \vec{W}_d \langle \vec{s} \odot \vec{z}\,\rangle_{p(\vec{s},\vec{z} \,|\, \vec{y}^{\,\mathrm{obs}},\Theta)}. \tag{40}$$

The estimator (40) can be efficiently computed based on (47) by approximating the expectation w.r.t. the binary posterior by using truncated expectations (3).

## Appendix D. M-step Update Equations

*NOR.* For completeness, we report the NOR update rules here:

$$\pi_h = \frac{1}{N}\sum_{n=1}^{N} \langle s_h \rangle_{q^{(n)}}\,, \qquad W_{dh} = 1 + \frac{\sum_{n=1}^{N}(y_d^{(n)} - 1)\,\langle D_{dh}(\vec{s}\,)\rangle_{q^{(n)}}}{\sum_{n=1}^{N}\langle C_{dh}(\vec{s}\,)\rangle_{q^{(n)}}} \tag{41}$$

where

$$D_{dh}(\vec{s}\,) := \frac{\widetilde{W}_{dh}(\vec{s}\,)s_h}{N_d(\vec{s}\,)(1 - N_d(\vec{s}\,))},\;\; C_{dh}(\vec{s}\,) := \widetilde{W}_{dh}(\vec{s}\,)D_{dh}(\vec{s}\,),\;\; \widetilde{W}_{dh}(\vec{s}\,) := \prod_{h' \neq h}(1 - W_{dh'}s_{h'}). \tag{42}$$

The update equations for the weights $W_{dh}$ do not allow for a closed-form solution. We instead employ a fixed-point equation whose fixed point is the exact solution of the maximization step. We exploit the fact that in practice one single evaluation of (42) is enough to (noisily, not optimally) move towards convergence to efficiently improve on the parameters $W_{dh}$.

*BSC.* As for NOR, we report the explicit forms of the M-step update rules for the BSC model here for completeness (compare, e.g., Henniges et al., 2010):

$$\begin{aligned}
\pi &= \frac{1}{NH}\sum_{n=1}^{N}\sum_{h=1}^{H}\langle s_h\rangle_{q^{(n)}}\,, \qquad \sigma^2 = \frac{1}{ND}\sum_{n=1}^{N}\Big\langle ||\,\vec{y}^{\,(n)} - W\vec{s}\,||^2\Big\rangle_{q^{(n)}}\,,\\
W &= \left(\sum_{n=1}^{N}\vec{y}^{\,(n)}\,\langle\vec{s}\,\rangle_{q^{(n)}}^{T}\right)\left(\sum_{n'=1}^{N}\big\langle\vec{s}\vec{s}^{\,T}\big\rangle_{q^{(n')}}\right)^{-1}.
\end{aligned} \tag{43}$$

*SSSC.* We report the final expressions of the SSSC M-step update equations below. The derivations can be found in Sheikh et al. (2014).

$$\begin{aligned}
W &= \frac{\sum_{n=1}^{N}\vec{y}^{\,(n)}\big\langle\vec{s}\odot\vec{z}\,\big\rangle_{q^{(n)}}^{\mathrm{T}}}{\sum_{n=1}^{N}\big\langle(\vec{s}\odot\vec{z}\,)\,(\vec{s}\odot\vec{z}\,)^{\mathrm{T}}\big\rangle_{q^{(n)}}}, \qquad \vec{\pi} = \frac{1}{N}\sum_{n=1}^{N}\big\langle\vec{s}\,\big\rangle_{q^{(n)}}, \qquad \vec{\mu} = \frac{\sum_{n=1}^{N}\big\langle\vec{s}\odot\vec{z}\,\big\rangle_{q^{(n)}}}{\sum_{n=1}^{N}\big\langle\vec{s}\,\big\rangle_{q^{(n)}}},\\
\Psi &= \sum_{n=1}^{N}\Big[\big\langle(\vec{s}\odot\vec{z}\,)\,(\vec{s}\odot\vec{z}\,)^{\mathrm{T}}\big\rangle_{q^{(n)}} - \big\langle\vec{s}\vec{s}^{\,\mathrm{T}}\big\rangle_{q^{(n)}}\odot\vec{\mu}\vec{\mu}^{\,\mathrm{T}}\Big]\odot\left(\sum_{n=1}^{N}\Big[\big\langle\vec{s}\vec{s}^{\,\mathrm{T}}\big\rangle_{q^{(n)}}\Big]\right)^{-1},\\
\sigma^2 &= \frac{1}{ND}\,\mathrm{Tr}\left(\sum_{n=1}^{N}\Big[\vec{y}^{\,(n)}(\vec{y}^{\,(n)})^{\mathrm{T}} - W\Big[\big\langle\vec{s}\odot\vec{z}\,\big\rangle_{q^{(n)}}\big\langle\vec{s}\odot\vec{z}\,\big\rangle_{q^{(n)}}^{\mathrm{T}}\Big]W^{\mathrm{T}}\Big]\right).
\end{aligned} \tag{44}$$

As defined in (15), the expectations in (44) are taken w.r.t. the full binary-continuous latent space. Importantly for applying EVO, all these expectation values can be reformulated as expectations w.r.t. the posterior over the binary latent space:

$$\big\langle \vec{s}\,\big\rangle_{q^{(n)}} = \sum_{\vec{s}} q^{(n)}(\vec{s};\Theta)\vec{s}, \tag{45}$$

$$\big\langle \vec{s}\vec{s}^{\,\mathrm{T}}\big\rangle_{q^{(n)}} = \sum_{\vec{s}} q^{(n)}(\vec{s};\Theta)\vec{s}\vec{s}^{\,\mathrm{T}}, \tag{46}$$

$$\big\langle \vec{s}\odot\vec{z}\,\big\rangle_{q^{(n)}} = \sum_{\vec{s}} q^{(n)}(\vec{s};\Theta)\vec{\kappa}^{(n)}_{\vec{s}}, \tag{47}$$

$$\big\langle (\vec{s}\odot\vec{z}\,)\,(\vec{s}\odot\vec{z}\,)^{\mathrm{T}}\big\rangle_{q^{(n)}} = \sum_{\vec{s}} q^{(n)}(\vec{s};\Theta)(\Lambda_{\vec{s}} + \vec{\kappa}^{(n)}_{\vec{s}}(\vec{\kappa}^{(n)}_{\vec{s}})^{\mathrm{T}}), \tag{48}$$

where $q^{(n)}(\vec{s};\Theta)$ is (for the exact EM solution) the binary posterior given by

$$p\,(\vec{s}\mid\vec{y},\Theta) = \frac{\mathcal{B}(\vec{s};\vec{\pi})\;\mathcal{N}(\vec{y};\,\tilde{W}_{\vec{s}}\,\vec{\mu},C_{\vec{s}})}{\sum_{\vec{s}\,'}\mathcal{B}(\vec{s}\,';\vec{\pi})\;\mathcal{N}(\vec{y};\,\tilde{W}_{\vec{s}\,'}\,\vec{\mu},C_{\vec{s}\,'})}. \tag{49}$$

In the above equations $\Lambda_{\vec{s}} = (\sigma^{-2}\tilde{W}_{\vec{s}}^{\mathrm{T}}\tilde{W}_{\vec{s}}+\Psi_{\vec{s}}^{-1})^{-1}$ and $\vec{\kappa}^{(n)}_{\vec{s}} = (\vec{s}\odot\vec{\mu})+\sigma^{-2}\Lambda_{\vec{s}}\tilde{W}_{\vec{s}}^{\mathrm{T}}(\vec{y}^{\,(n)}-\tilde{W}_{\vec{s}}\vec{\mu})$. $\sum_{\vec{s}}$ denotes a summation over all binary vectors $\vec{s}$. Based on the reformulations, the expectations in (45)-(48) can be approximated using truncated expectations (3).

## Appendix E. Evaluation Criteria for Image Restoration

A standard metric for the evaluation of image restoration (IR) methods is the peak-signal-to-noise ratio (PSNR; compare Section 4). In the context of IR benchmarks (e.g., denoising, inpainting, image super resolution), PSNR values are informative about the root-mean-square error between the restored image calculated by a specific IR algorithm and the respective clean target image. In addition to the PSNR measure, there are other criteria that can be used to compare IR approaches. Some criteria will be discussed in the following.

*Clean data.* Supervised learning-based IR methods such as the denoising and inpainting methods of category DE6 and respectively IN6 in Table 2 require external data sets with clean images for training. In contrast, sparse coding and dictionary learning approaches such as the methods from category DE1 and IN1 (including the generative model algorithms EBSC and ES3C) are trained exclusively on the corrupted data that they aim to restore. The ability of learning solely from corrupted data is very valuable for scenarios in which clean data is not available or difficult to generate. Besides, the internal statistics of a test image were observed to be often more predictive than statistics learned from external data sets (compare Shocher et al., 2018). For deep neural networks, which often do require external clean training data, recent work seeks to provide methods to also allow them to be trained on noisy data alone (e.g. Lehtinen et al., 2018; Ulyanov et al., 2018; Krull et al., 2019). Further related work proposes methods for "zero-shot" super-resolution which can be trained exclusively on the corrupted test data (Shocher et al., 2018).

*A-priori information and robustness.* For denoising, it is frequently assumed that a-priori knowledge of the ground-truth noise level is available. For instace KSVD, BM3D, WNNM, cKSVD or LSSC treat the noise level as input parameter of the algorithm. Other approaches that are trained using pairs of noisy/clean examples are typically optimized either for a single noise level (e.g. MLP, IRCNN, DnCNN-S, TNRD) or for several noise levels (e.g., MemNet, DnCNN-B, FFDNet, DPDNN, BDGAN). The denoising performance of noise-level-optimized approaches may deteriorate significantly if the algorithm is not provided with the appropriate (ground-truth) noise level of the test image (compare, e.g., Burger et al., 2012; Chaudhury and Roy, 2017; Zhang et al., 2018).

Similarly, for inpainting, it can be observed that certain methods exploit significant amounts of a-priori information, for example considering the text removal benchmark (Figure 7 F): For NLRMRF and IRCNN, the training data is generated by overlaying specific text patterns on clean images. For IRCNN, several distinct font styles and font sizes are used (Chaudhury and Roy, 2017); for NLRMRF, training examples are generated using the identical text pattern of the test image (Sun and Tappen, 2011). In contrast, the generative model algorithms EBSC and ES3C require neither task-specific external training data sets (such as IRCNN or NLRMRF) nor do they require a-priori knowledge of the noise level (in contrast to e.g. BM3D, MLP, IRCNN, FFDNET, DnCNN; compare Table 2). EBSC and ES3C learn the noise level parameter (which is part of the generative model) from the data in an unsupervised way.

*Context information.* Another criterion for the evaluation of IR methods is the amount of context information that a particular algorithm exploits. The amount of context information is primarily determined by the patch size of the image segmentation. Increasing the effective patch size was reported to be beneficial for the performance of IR algorithms (compare Burger et al., 2012; Zhang et al., 2017). Zhang et al. reported that the effective patch size used by denoising methods can be found to vary greatly between different approaches (e.g. $36 \times 36$ used by EPLL, $50 \times 50$ by DnCNN-B, $70 \times 70$ by FFDNet, $361 \times 361$ by WNNM; numbers taken from Zhang et al., 2017, 2018). The denoising experiments with EBSC and ES3C were conducted using patches that did not exceed an effective size of $23 \times 23$ (compare Table 4 in Appendix F.2), which is considerably smaller than the numbers reported by Zhang et al..

*Stochasticity in the acquisition of test data.* For the denoising benchmarks considered in Section 4.2, there is stochastic variation in the test data due to the fact that for a given AWG noise level $\sigma$ and for a given image, different realizations of the noise result in different noisy images. PSNR values can consequently vary even if the applied algorithm is fully deterministic. As all images investigated here are at least of size $256 \times 256$, variations due to different noise realizations are commonly taken as negligible for comparison (see, e.g., Mairal et al., 2009). The denoising results of EBSC and ES3C reported in Figures 5 and 6 were obtained by executing the algorithm three times on each image using different noise realizations in each run. Observed PSNR standard deviations were smaller or equal 0.06 dB (compare Table 3). For the inpainting experiments with randomly missing values (Figure 7 A-E), we also performed three runs for each image using a different realization of missing values in each run. In these experiments, we observed slightly higher PSNR variations (standard deviations ranged from 0.02 to 0.18 dB). The different realizations of the noise (or of the missing values) employed for each execution of the algorithm might be relevant to explain the PSNR variations that we observed. Additionally, stochasticity in the EBSC and ES3C algorithms themselves cannot be excluded as a further factor.

*Stochasticity in the algorithm.* The output of stochastic algorithms can differ from run to run even if the input remains constant. For learning algorithms, different PSNR values usually correspond to different optima of the learning objective which are obtained due to different initializations and/or due to stochasticity during learning. For stochastic algorithms, it is therefore the question which PSNR value shall be used for comparison. Two of the algorithms that we used for comparison in Figures 5 - 7 report averages over different runs (KSVD and LSSC). All other algorithms do report a single PSNR value per denoising/inpainting setting without values for standard deviations. A single PSNR value is (for one realization of the corrupted image) naturally obtained for deterministic algorithms (e.g. BM3D, WNNM, NL). For the stochastic algorithms, a single PSNR may mean that either (A) just one run of the learning algorithm was performed, that (B) the best PSNR of all runs was reported, or that (C) one run of the learning algorithm was selected via another criterion. For DNN algorithms, for instance, the DNN with the lowest training or validation error could be selected for denoising or inpainting. Or for sparse coding algorithms, the model parameters with the highest (approximate) likelihood could be selected for denoising or inpainting. As the contributions in Figures 5 - 7 which state just one PSNR value do not give details on how it was obtained from potentially several runs, it may be assumed that the best performing runs were reported (or sometimes the only run, e.g., in cases of a DNN requiring several days or weeks for training, compare Jain and Seung, 2009; Burger et al., 2012; Chaudhury and Roy, 2017; Zhang et al., 2018). Stating the best run is instructive as it shows how well an algorithm can perform under best conditions. For comparability, it should be detailed how the single PSNR value was selected, however (or if average and best values are essentially identical).

For practical applications, it is desirable to be able to select the best of several runs based on a criterion that can be evaluated without ground-truth knowledge. Our experimental data shows that for EBSC and ES3C the best denoising and inpainting performance in terms of PSNR cannot reliably be determined based on the learning objective, namely the variational bound (which is computable without ground-truth knowledge): The PSNR values of the runs with the highest bound may be smaller than the highest PSNR values observed in all runs. In a scenario with fixed noise realization, we observed the variational bound to be instructive about the PSNR value though (Figure 4).

## Appendix F. Details on Numerical Experiments

We provide more details on used soft- and hardware, more details on the conducted experiments, and we provide control experiments for different algorithms.

### F.1 Soft- and Hardware

We implemented EBSC and ES3C in Python using MPI-based parallelization for execution on multiple processors. Model parameter updates were efficiently computed by distributing

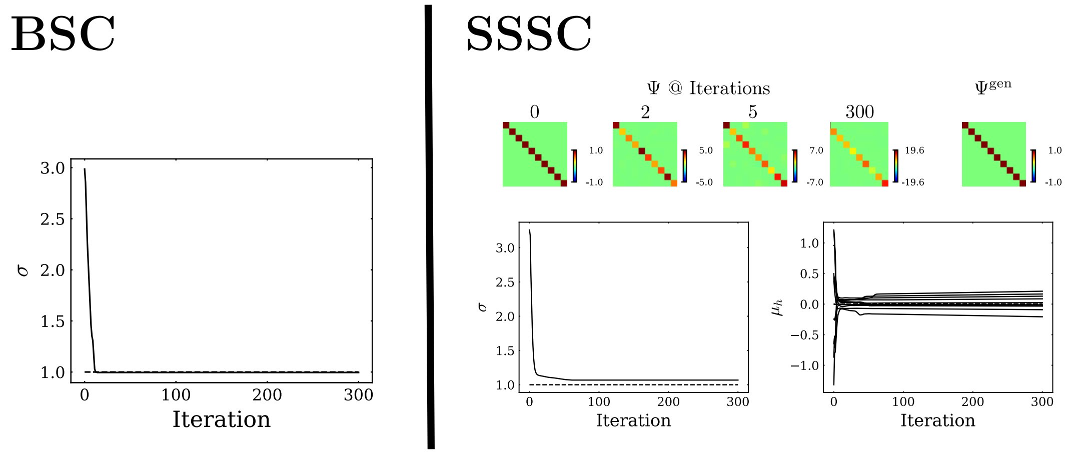


Figure 8: BSC and SSSC model parameters learned from artificial data using EVO (see the description of the experiment in Section 3.2 for more details).

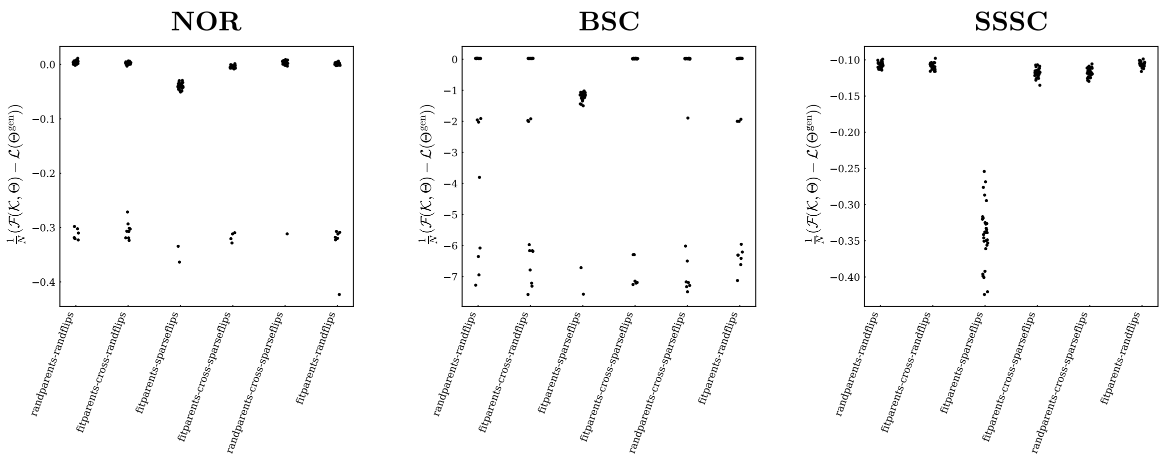


Figure 9: Final free energies obtained by different EA configurations over 30 runs of EVO for NOR, BSC and SSSC models on 5×5 bars images (see the description of the experiment in Section 3.2 for more details).

data points to multiple processors and performing calculations for batches of data points in parallel (compare Appendix D). Small scale experiments such as the bars tests described in Section 3.2 were performed on standalone workstations with four-core CPUs; runtimes were in the range of minutes. For most of the large-scale experiments described in Sections 3.3 and 4, we used the HPC cluster CARL of the University of Oldenburg to execute our algorithms. The cluster is equipped with several hundred compute nodes with in total several thousands of CPU cores (Intel Xeon E5-2650 v4 12C, E5-2667 v4 8C and E7-8891 v4 10c). For each of the simulations, potentially different compute ressources were employed (different numbers of CPU cores, different CPU types, different numbers of compute nodes, different numbers of employed cores per node), and hence runtimes were found to vary greatly between the

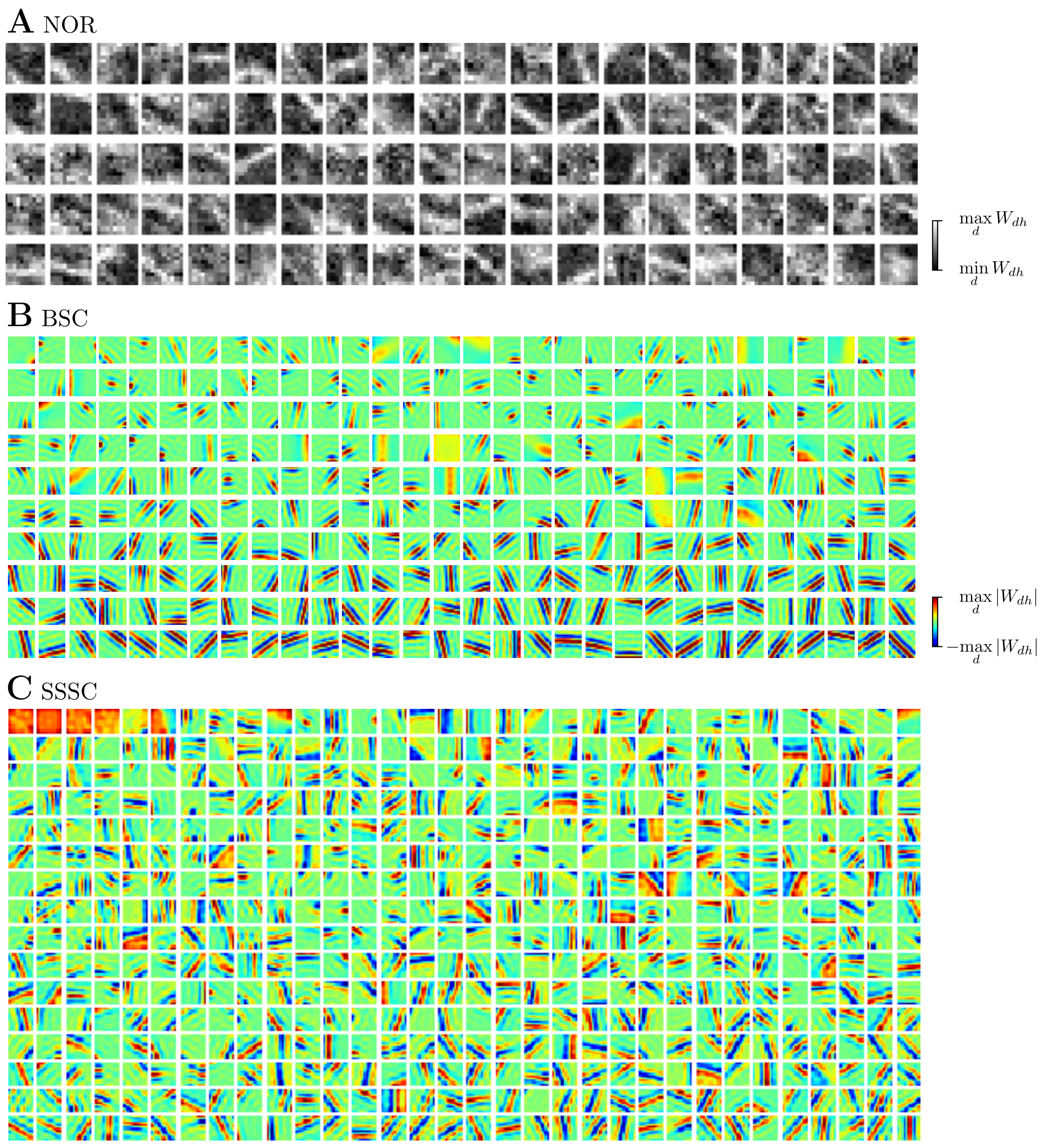


Figure 10: Dictionaries learned from natural image patches using NOR, BSC and SSSC models (see Section 3.3). For NOR, we considered raw image patches and trained a model with $H = 100$ components. For BSC and SSSC, image patches were preprocessed using a whitening procedure. For BSC and SSSC, $H = 300$ and $H = 512$ generative fields were learned, respectively. In **B** and **C**, the fields are ordered according to their activation, starting with the fields corresponding to the most active hidden units.

**A** Denoising (Comparison of generative models and approximate inference under controlled conditions; Section 4.2.2)

| Run | House $\sigma = 50$ ($H = 64$, $D = 8 \times 8$) | | | House $\sigma = 50$ ($H = 256$, $D = 8 \times 8$) | | | House $\sigma = 50$ ($H = 512$, $D = 12 \times 12$) | | |
|---|---|---|---|---|---|---|---|---|---|
| | GSVAE$^{\text{lin}}$ | EBSC | ES3C | GSVAE$^{\text{lin}}$ | EBSC | ES3C | GSVAE$^{\text{lin}}$ | EBSC | ES3C |
| 1 | 19.14 | 28.30 | 28.87 | 18.93 | 28.29 | 28.89 | 18.98 | 28.35 | 29.88 |
| 2 | 19.16 | 28.04 | 28.74 | 18.92 | 28.27 | 28.86 | 18.98 | 28.29 | 29.85 |
| 3 | 19.14 | 27.99 | 28.72 | 18.89 | 28.22 | 28.79 | 18.91 | 28.16 | 29.74 |
| ∅ | 19.15 | 28.11 | 28.78 | 18.92 | 28.26 | 28.85 | 18.95 | 28.27 | 29.83 |

**B** Denoising (General comparison of denoising approaches; Section 4.2.3)

| Run | House $\sigma = 15$ | | | House $\sigma = 25$ | | | House $\sigma = 50$ | | | Barbara $\sigma = 25$ | | | Lena $\sigma = 25$ | | | Peppers $\sigma = 25$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | GSVAE$^{\text{lin}}$ | EBSC | ES3C | GSVAE$^{\text{lin}}$ | EBSC | ES3C | GSVAE$^{\text{lin}}$ | EBSC | ES3C | GSVAE$^{\text{lin}}$ | EBSC | ES3C | GSVAE$^{\text{lin}}$ | EBSC | ES3C | GSVAE$^{\text{lin}}$ | EBSC | ES3C |
| 1 | 29.38 | 33.70 | 34.86 | 28.62 | 32.39 | 33.16 | 25.88 | 29.03 | 29.88 | 24.97 | 28.33 | 30.17 | 27.93 | 30.91 | 31.90 | 25.52 | 29.01 | 30.35 |
| 2 | 29.37 | 33.65 | 34.93 | 28.06 | 32.47 | 33.20 | 25.68 | 28.91 | 29.85 | 25.21 | 28.39 | 30.14 | 28.10 | 30.90 | 31.94 | 25.54 | 28.87 | 30.22 |
| 3 | 29.41 | 33.63 | 34.93 | 28.29 | 32.34 | 33.09 | 25.62 | 29.01 | 29.74 | 25.19 | 28.42 | 30.24 | 28.00 | 30.84 | 32.01 | 25.52 | 28.90 | 30.26 |
| ∅ | 29.39 | 33.66 | 34.90 | 28.32 | 32.40 | 33.15 | 25.73 | 28.98 | 29.83 | 25.12 | 28.38 | 30.19 | 28.01 | 30.88 | 31.95 | 25.53 | 28.93 | 30.27 |

**C** Inpainting (Section 4.3)

| Run | Barbara | Cameraman | Lena | House | | Castle | |
|---|---|---|---|---|---|---|---|
| | 50% missing | 50% missing | 50% missing | 50% missing | 80% missing | 50% missing | 80% missing |
| 1 | 35.39 | 30.91 | 37.57 | 39.55 | 32.99 | 38.14 | 29.65 |
| 2 | 35.51 | 30.46 | 37.47 | 39.70 | 33.21 | 38.39 | 29.68 |
| 3 | 35.42 | 30.70 | 37.58 | 39.54 | 32.99 | 38.14 | 29.64 |
| ∅ | 35.44 | 30.69 | 37.54 | 39.59 | 33.06 | 38.23 | 29.66 |

Table 3: PSNR values (in dB) measured for GSVAE$^{\text{lin}}$, EBSC and ES3C in the denoising and inpainting experiments described in Sections 4.2.2, 4.2.3 and 4.3.

simulations. For example for the denoising experiments with EBSC and ES3C on the "House" image (Figure 5), the algorithms were executed using between 100 and 640 CPU cores, and runtimes ranged approximately between six and one hundred hours for one run. Images larger than "House" required still longer runtimes or more CPU cores (we went up to a few thousand CPU cores on HPC clusters).

For the experiments with GSVAE, the code provided by the original publication (Jang, 2016) can execute optimization on GPU cores as is customary for deep models. When executing, for example, the CIFAR-10 experiment (Figure 11) on a single NVIDIA Tesla V100 16GB GPU, we observed runtimes of GSVAE$^{\text{lin}}$ on the order of a few seconds per iteration (on average approximately 10s); for the CIFAR-10 experiment, we observed GSVAE$^{\text{lin}}$ to converge within approximately 150 iterations. In comparison, to train EBSC on CIFAR-10, we executed our implementation in parallel on 768 CPU cores (Intel Xeon Platinum 9242) and performed 750 iterations of the algorithm. At the final iteration, the value of the lower bound was still slightly increasing, however not significantly anymore; the runtime per iteration was on the order of a few seconds (on average approximately 9s). In summary, GSVAE execution is more efficient. Comparison is difficult, however, because of the different setting. The more standard GSVAE approach uses conventional deep learning tools that can be expected to be well-optimized, while the distribution of EVO optimization across many CPUs (and cores) generates communication overhead, and the efficiency of implementation components can presumably be further enhanced.

### F.2 Hyperparameters

Table 4 lists the hyperparameters employed in the numerical experiments on verification (Section 3.2), scalability (Section 3.3), denoising (Section 4.2) and inpainting (Section 4.3).

EVO hyperparameters ($S$, $N_p$, $N_m$, $N_g$) were chosen s.t. they provided a reasonable trade-off between the accuracy of the approximate inference scheme and the runtime of the algorithm. For GSVAE$^{\text{lin}}$, we used the same neural network architecture for the encoding model and the same hyperparameters as in the source code provided by the original publication (Jang, 2016): The encoder network contained two hidden layers with 512 and 256 hidden units, respectively. The activation function of all network units was the identity (i.e. no non-linearity). The initial annealing temperature, the annealing rate and the minimal annealing temperature of the Gumbel-Softmax distribution were set to 1.0, $3 \cdot 10^{-5}$ and 0.5, respectively. The initial learning rate of the Adam optimizer was $10^{-3}$. The patch sizes and number of latents used for GSVAE$^{\text{lin}}$ in the denoising experiments are, together with the corresponding values used for EBSC and ES3C, listed in Table 4C–D.

**A** Verification and scalability (Sections 3.2 and 3.3)

| Experiment | $N$ | $D$ | $H$ | EA | $S$ | $N_p$ | $N_m$ | $N_g$ | Iterations |
|---|---|---|---|---|---|---|---|---|---|
| Bars Test NOR/BSC/SSSC | 5,000 | $5 \times 5$ | 10 | randparents-randflips | 20 | 5 | 4 | 2 | 300 |
| Bars Test NOR/BSC/SSSC | 5,000 | $5 \times 5$ | 10 | fitparents-cross-randflips | 20 | 5 | - | 2 | 300 |
| Bars Test NOR/BSC/SSSC | 5,000 | $5 \times 5$ | 10 | fitparents-sparseflips | 20 | 5 | 4 | 2 | 300 |
| Bars Test NOR/BSC/SSSC | 5,000 | $5 \times 5$ | 10 | fitparents-cross-sparseflips | 20 | 5 | - | 2 | 300 |
| Bars Test NOR/BSC/SSSC | 5,000 | $5 \times 5$ | 10 | randparents-cross-sparseflips | 20 | 5 | - | 2 | 300 |
| Bars Test NOR/BSC/SSSC | 5,000 | $5 \times 5$ | 10 | fitparents-randflips | 20 | 5 | 4 | 2 | 300 |
| van Hateren Images NOR | 30,000 | $10 \times 10$ | 100 | fitparents-cross-sparseflips | 120 | 8 | - | 2 | 200 |
| van Hateren Images BSC | 100,000 | $16 \times 16$ | 300 | fitparents-cross-sparseflips | 200 | 10 | - | 4 | 4,000 |
| van Hateren Images SSSC | 100,000 | $12 \times 12$ | 512 | fitparents-cross-sparseflips | 60 | 6 | - | 2 | 2,000 |

**B** Denoising (Relation between PSNR measure and variational lower bound; Section 4.2.1)

| Image | Size | Noise Level $\sigma$ | $D$ | $H$ | EA | $S$ | $N_p$ | $N_m$ | $N_g$ | Iterations |
|---|---|---|---|---|---|---|---|---|---|---|
| House | $256 \times 256$ | 50 | $8 \times 8$ | 256 | fitparents-randflips | 60 | 6 | 5 | 2 | 2,000 |

**C** Denoising (Comparison of generative models and approximate inference under controlled conditions; Section 4.2.2)

| Image | Size | Noise Level $\sigma$ | $D$ | $H$ | EBSC- and ES3C-specific | | | | | | GSVAE$^{\text{lin}}$-specific |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | EA | $S$ | $N_p$ | $N_m$ | $N_g$ | Iterations | Iterations |
| House | $256 \times 256$ | 50 | $8 \times 8$ | 64 | fitparents-randflips | 60 | 60 | 1 | 1 | 4,000 | 100 |
| House | $256 \times 256$ | 50 | $8 \times 8$ | 256 | fitparents-randflips | 60 | 60 | 1 | 1 | 4,000 | 100 |
| House | $256 \times 256$ | 50 | $12 \times 12$ | 512 | fitparents-randflips | 60 | 60 | 1 | 1 | 4,000 | 100 |

**D** Denoising (General comparison of denoising approaches; Section 4.2.3)

| Image | Size | Noise Level $\sigma$ | EBSC-specific | | | | | | ES3C-specific | | | | | | EBSC- and ES3C-specific | | GSVAE$^{\text{lin}}$-specific | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | $D$ | $H$ | $S$ | $N_p$ | $N_m$ | $N_g$ | $D$ | $H$ | $S$ | $N_p$ | $N_m$ | $N_g$ | EA | Iterations | $D$ | $H$ | Iterations |
| House | $256 \times 256$ | 15 | $8 \times 8$ | 256 | 200 | 10 | 9 | 4 | $12 \times 12$ | 512 | 60 | 60 | 1 | 1 | fitparents-randflips | 4,000 | $16 \times 16$ | 512 | 100 |
| House | $256 \times 256$ | 25 | $8 \times 8$ | 256 | 200 | 10 | 9 | 4 | $12 \times 12$ | 512 | 60 | 60 | 1 | 1 | fitparents-randflips | 4,000 | $32 \times 32$ | 512 | 100 |
| House | $256 \times 256$ | 50 | $8 \times 8$ | 256 | 200 | 10 | 9 | 4 | $12 \times 12$ | 512 | 60 | 60 | 1 | 1 | fitparents-randflips | 4,000 | $32 \times 32$ | 512 | 100 |
| Barbara | $512 \times 512$ | 25 | $8 \times 8$ | 256 | 200 | 10 | 9 | 4 | $11 \times 11$ | 512 | 60 | 60 | 1 | 1 | fitparents-randflips | 3,000 | $16 \times 16$ | 512 | 100 |
| Lena | $512 \times 512$ | 25 | $8 \times 8$ | 256 | 200 | 10 | 9 | 4 | $11 \times 11$ | 512 | 60 | 60 | 1 | 1 | fitparents-randflips | 4,000 | $16 \times 16$ | 512 | 100 |
| Peppers | $256 \times 256$ | 25 | $8 \times 8$ | 256 | 200 | 10 | 9 | 4 | $10 \times 10$ | 800 | 40 | 30 | 1 | 1 | fitparents-randflips | 6,000 | $32 \times 32$ | 512 | 100 |

**E** Inpainting (Section 4.3)

| Image | Size | Missing Data Ratio | $D$ | $H$ | EA | $S$ | $N_p$ | $N_m$ | $N_g$ | Iterations |
|---|---|---|---|---|---|---|---|---|---|---|
| Barbara | $512 \times 512$ | 50% | $12 \times 12$ | 512 | fitparents-randflips | 30 | 20 | 1 | 1 | 4,000 |
| Cameraman | $256 \times 256$ | 50% | $12 \times 12$ | 512 | fitparents-randflips | 30 | 20 | 1 | 1 | 4,000 |
| Lena | $512 \times 512$ | 50% | $12 \times 12$ | 512 | fitparents-randflips | 30 | 20 | 1 | 1 | 4,000 |
| House | $256 \times 256$ | 50% | $12 \times 12$ | 512 | fitparents-randflips | 30 | 20 | 1 | 1 | 4,000 |
| House | $256 \times 256$ | 80% | $15 \times 15$ | 512 | fitparents-randflips | 30 | 20 | 1 | 1 | 500 |
| Castle | $481 \times 321$ | 50% | $7 \times 7$ | 900 | fitparents-randflips | 30 | 20 | 1 | 1 | 2,000 |
| Castle | $481 \times 321$ | 80% | $7 \times 7$ | 900 | fitparents-randflips | 60 | 60 | 1 | 1 | 200 |
| New Orleans | $297 \times 438$ | Text Mask | $14 \times 14$ | 900 | fitparents-randflips | 60 | 60 | 1 | 1 | 3,000 |

Table 4: Hyperparameters employed in the numerical experiments.

## F.3 Comparison to Other Generative Models – Details

For our experiments with the Gumbel-Softmax Variational Autoencoder (GSVAE), we leveraged the source code provided by the original publication (Jang, 2016). For GSVAE$^{\text{lin}}$, we chose a categorical distribution with just two categories ("0" or "1") which matches the Bernoulli prior of BSC. Also to match the BSC generative model, we used a shallow, linear decoder (with weights $W$ and without bias terms) connected only to the category "1" units. The scalar variance $\sigma^2$ of the Gaussian was treated as a trainable parameter (optimized alongside the other parameters), and we used the same prior parameter $\pi$ for all $h = 1, \ldots, H$ as for BSC. The decoding model of GSVAE thus becomes equivalent to the generative model of BSC in Equation 11. As controls, we used different versions of GSVAE including versions with (i) deep decoders connected only to category "1" units (below and in Table 5, we refer to this GSVAE version "Binary-Deep"), (ii) shallow decoders connected to both categories (referred to as "Categorical-Shallow"), and (iii) deep decoders connected to both categories (referred to as "Categorical-Deep"; below and in Table 5, we refer to GSVAE$^{\text{lin}}$, i.e., the version used for the denoising benchmarks in Sections 4.2.2 and 4.2.3, as the "Binary-Shallow" version). The deep decoders had two hidden layers with 256 and 512 units. For GSVAE$^{\text{lin}}$ and all controls, we used the same encoding model as in the original implementation (architecture and hyperparameters listed in Appendix F.2). Also, for consistency with the original implementation, we used identity activation functions for all GSVAE versions.

To confirm the functionality of the implementation used, we first applied GSVAE$^{\text{lin}}$ and the different control versions to two standard data sets: image patches of the "Barbara" image with AWG noise and images of whole objects from the CIFAR-10 data set. For "Barbara", we used a noise level of $\sigma = 25$, $D = 8 \times 8$ patches and $H = 256$; for CIFAR-10, the patch size was $D = 32 \times 32 \times 3$, and we used $H = 1{,}024$. The CIFAR-10 images had amplitudes in the range $[0, 1]$; accordingly, we also scaled the clean "Barbara" image (that we used to generate noisy image patches) and the noise level to the range $[0, 1]$. Table 5 lists the values of the lower bound obtained in different runs of the algorithms for the two data sets. As can be observed, there are significant differences between different runs of the GSVAE algorithms, especially for CIFAR-10. On this data set, the best runs resulted in values of the lower bound in the range of 2,300 (obtained with the "Binary-Shallow" version of GSVAE) to 2,620 (obtained with the "Categorical-Shallow" version; see Table 5, bottom four rows). Already the GSVAE versions with shallow decoder performed relatively well on this benchmark. For comparison, the highest log-likelihood values reported in Park et al. (2019) for their VLAE approaches on CIFAR-10 are 2,392 for a shallow version and 2,687 for a deep version. The "Categorical-Shallow" GSVAE control improved on the "Binary-Shallow" version (GSVAE$^{\text{lin}}$); at the same time, the former version has twice as many weight parameters. Deep decoders also tended to improve performance on CIFAR-10 but not very significantly. The original GSVAE implementation used linear activation units. When we substituted linear activation by ReLU units, performance even tended to decrease (which is presumably the reason for the linear activations in the original publication (Jang et al., 2017)).

While performance of GSVAE in terms of lower bounds is relatively high on CIFAR-10, performance on image patches of the noisy "Barbara" image is relatively low. GSVAE$^{\text{lin}}$ and also all controls resulted in significantly lower values of the bound than values obtained by EBSC. On the other hand, EBSC values are significantly lower on average on CIFAR-10. On image patches of the noisy "Barbara" image, EBSC (and also other approaches such as MTMKL) are able to learn representations with generative fields (GFs) being much more aligned with actual image structures (Figure 11B); and EBSC uses a sparse code to combine

| | Run | EBSC | GSVAE | | | |
|---|---|---|---|---|---|---|
| | | | Binary-Shallow | Binary-Deep | Categorical-Shallow | Categorical-Deep |
| Barbara | 1 | 43.72 | 36.78 | 37.43 | 37.79 | 36.78 |
| | 2 | **43.85** | 37.49 | 37.71 | 37.55 | 36.88 |
| | 3 | 43.61 | 38.33 | 37.49 | 37.96 | 37.75 |
| | ∅ | 43.73 | 37.53 | 37.54 | 37.77 | 37.14 |
| CIFAR-10 | 1 | 2,027.55 | 2,325.95 | 1,919.39 | 170.74 | 2,376.67 |
| | 2 | 2,021.42 | 2,307.94 | 513.90 | 2,545.02 | 2,559.23 |
| | 3 | 2,019.11 | 2,301.24 | 2,380.29 | **2,618.58** | 2,544.10 |
| | ∅ | 2,022.70 | 2,311.71 | 1,604.53 | 1,778.11 | 2,493.33 |

Table 5: Free energies (ELBOs) per datapoint obtained with EBSC and GSVAE on image patches of the noisy "Barbara" image ($\sigma = 25$) and on CIFAR-10 (see text for details). For "Barbara", the values listed denote free energies on the noisy image patches which make up the training data set; for CIFAR-10, the values listed denote free energies on the test data set. The highest free energy obtained for each benchmark is marked bold.

only few of the fields for reconstruction (on "Barbara", EBSC used approximately 1.5 out of 256 fields, on CIFAR-10 approximately 19 out of 1,024). GSVAE$^{\text{lin}}$ uses the same generative model as EBSC, so we can directly compare the GFs (i.e., the columns of $W$). In contrast to EBSC, the GFs of GSVAE$^{\text{lin}}$ are less interpretable and contain higher spatial frequencies (Figure 11E). On average, GSVAE$^{\text{lin}}$ used about half of these fields for reconstruction (i.e. 128 out of 256 fields on "Barbara"). GSVAE$^{\text{lin}}$ also learned such fields and dense codes for CIFAR-10 (Figure 11F; approximately 512 out of 1,024 fields were used on average). On CIFAR-10, the dense codes of GSVAE$^{\text{lin}}$ and of the controls seem comparably effective, while it seems more difficult to encode whole object images using sparse codes.

For GSVAE we can, of course, not exclude that settings can be found for which sparse codes and high PSNR values are achieved on "Barbara" and other denoising benchmarks. However, no such setting and competitive denoising performance has been reported in the literature for either GSVAE or other VAEs (to the knowledge of the authors; but compare Prakash et al., 2021). For other data, sparse codes may not necessarily result in better performance (Table 5). For standard denoising and especially for the "zero-shot" setting which we focused on, good image patch models are required, however, and sparse codes are observed to be advantageous in this case.

Regarding the above discussion, note that other deep generative models *have* been applied to denoising and inpainting. Creswell and Bharath report reconstruction performance on AWG noise removal tasks; however in Creswell and Bharath (2018), (i) the considered noise levels are significantly smaller than the ones in Figures 5 and 6, and (ii) data sets of whole objects such as Omniglot or CelebA are employed rather than the test images of Figures 5 and 6. Related contributions on GANs frequently consider the task of semantic inpainting, i.e. the reconstruction of large areas in images in a semantically plausible way (see, e.g., Cao et al., 2018, for an overview). In these contributions, typically data sets such as CelebA, Street

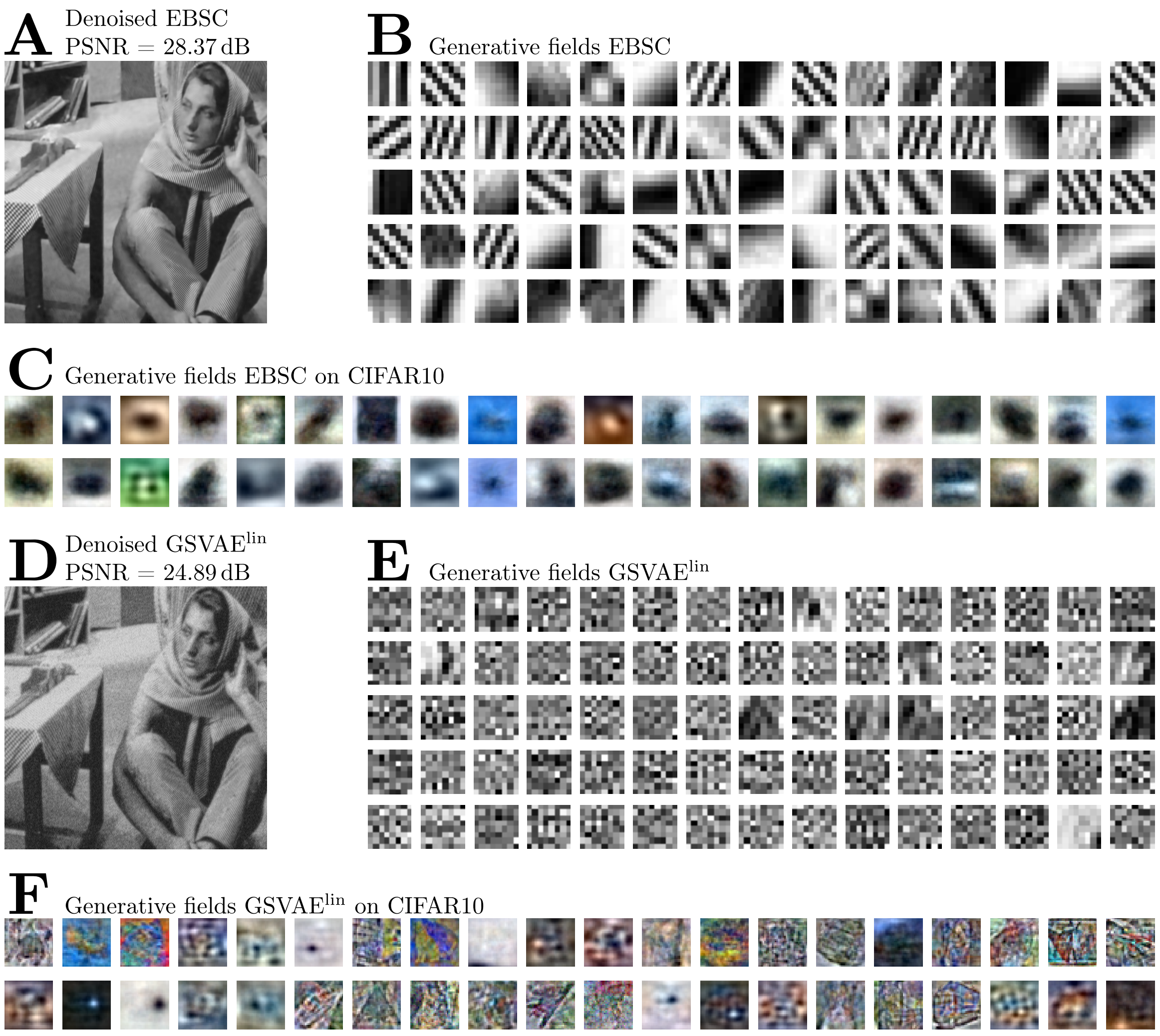

A
Denoised EBSC
PSNR = 28.37 dB
B
Generative fields EBSC
C
Generative fields EBSC on CIFAR10
D
Denoised GSVAE$^{\text{lin}}$
PSNR = 24.89 dB
E
Generative fields GSVAE$^{\text{lin}}$
F
Generative fields GSVAE$^{\text{lin}}$ on CIFAR10

Figure 11: Denoising and generative fields of EBSC and GSVAE (compare text and Table 5). **A, D** Denoising results after EBSC and GSVAE$^{\text{lin}}$ were used for denoising (input image was "Barbara" with $\sigma = 25$; depicted are the results of the run with highest lower bound). **B, C** Selection of GFs of EBSC for the run with highest lower bound on "Barbara" and CIFAR-10 data, respectively. **E, F** Selection of GFs of GSVAE$^{\text{lin}}$ for the run with highest lower bound on "Barbara" and CIFAR-10 data, respectively.

View House Numbers images or ImageNet are employed, and frequently the reconstruction of a single or a few (e.g. squared or rectangular) holes is considered as a benchmark (compare, e.g., Pathak et al., 2016; Yang et al., 2017; Yeh et al., 2017; Iizuka et al., 2017; Li et al., 2017). Yeh et al. also consider the task of restoring randomly missing values, however neither Yeh et al. (2017) nor the aforementioned publications report performance on the test images of Figure 7. The benchmarks of denoising and inpainting we use (Figures 5 to

7) are (A) very natural for EVO because of available benchmark results for mean field and sampling approaches, and (B) the benchmarks allow for comparison with state-of-the-art feed-forward DNNs (Figures 5 and 6).

## References

K. Ahmadi and E. Salari. Single-Image Super Resolution using Evolutionary Sparse Coding Technique. *IET Image Processing*, 11(1):13–21, 2016.

E. Angelino, M. J. Johnson, and R. P. Adams. Patterns of Scalable Bayesian Inference. *Foundations and Trends in Machine Learning*, 9(2-3):119–247, 2016.

A. J. Bell and T. J. Sejnowski. The "independent components" of natural scenes are edge filters. *Vision Research*, 37(23):3327–38, 1997.

S. A. Bigdeli, G. Lin, T. Portenier, L. A. Dunbar, and M. Zwicker. Learning Generative Models using Denoising Density Estimators. *arXiv preprint arXiv:2001.02728*, 2020.

D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

A. Bouchard-Côté and M. I. Jordan. Optimization of Structured Mean Field Objectives. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 67–74, 2009.

H. C. Burger, C. J. Schuler, and S. Harmeling. Image denoising: Can plain Neural Networks compete with BM3D? In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2392–2399. IEEE, 2012.

Y.-J. Cao, L.-L. Jia, Y.-X. Chen, N. Lin, C. Yang, B. Zhang, Z. Liu, X.-X. Li, and H.-H. Dai. Recent Advances of Generative Adversarial Networks in Computer Vision. *IEEE Access*, 2018.

S. Chaudhury and H. Roy. Can fully convolutional networks perform well for general image restoration problems? In *International Conference on Machine Vision Applications (MVA)*, pages 254–257, 2017.

Y. Chen and T. Pock. Trainable Nonlinear Reaction Diffusion: A Flexible Framework for Fast and Effective Image Restoration. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 39(6):1256–1272, 2017.

A. Creswell and A. A. Bharath. Denoising Adversarial Autoencoders. *IEEE Transactions on Neural Networks and Learning Systems*, 30(4):968–984, 2018.

K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. Image Denoising by Sparse 3D Transform-Domain Collaborative Filtering. *IEEE Transactions on Image Processing*, 16(8):2080–2095, 2007.

Z. Dai and J. Lücke. Autonomous Document Cleaning – A Generative Approach to Reconstruct Strongly Corrupted Scanned Texts. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 36(10):1950–1962, 2014.

P. Dayan, G. E. Hinton, R. Neal, and R. S. Zemel. The Helmholtz Machine. *Neural Computation*, 7: 889 – 904, 1995.

L. Dinh, D. Krueger, and Y. Bengio. NICE: Non-linear Independent Components Estimation. *arXiv preprint arXiv:1410.8516*, 2014.

L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density Estimation using Real NVP. In *International Conference on Learning Representations (ICLR)*, 2017.

W. Dong, P. Wang, W. Yin, G. Shi, F. Wu, and X. Lu. Denoising Prior Driven Deep Neural Network for Image Restoration. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2019.

M. Elad and M. Aharon. Image Denoising Via Sparse and Redundant Representations Over Learned Dictionaries. *IEEE Transactions on Image Processing*, 15(12):3736–3745, 2006.

G. Exarchakis and J. Lücke. Discrete Sparse Coding. *Neural Computation*, 29:2979–3013, 2017.

J. M. Fadili and J.-L. Starck. Sparse representations and Bayesian image inpainting. In *International Conferences SPARS'05*, 2005.

P. Földiák. Forming sparse representations by local anti-Hebbian learning. *Biological Cybernetics*, 64: 165–170, 1990.

D. Forster and J. Lücke. Can clustering scale sublinearly with its clusters? A variational EM acceleration of GMMs and k-means. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 124–132. PMLR, 2018.

D. Forster, A.-S. Sheikh, and J. Lücke. Neural Simpletrons: Learning in the Limit of Few Labels with Directed Generative Networks. *Neural Computation*, 30(8):2113–2174, 2018.

Z. Ghahramani and M. I. Jordan. Learning from incomplete data. 1995. A.I. Memo No. 1509.

D. E. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning.* Addison-Wesley, 1989.

I. Goodfellow, A. C. Courville, and Y. Bengio. Large-Scale Feature Learning With Spike-and-Slab Sparse Coding. In *International Conference on Machine Learning (ICML)*, 2012.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*. 2014.

S. Gu, L. Zhang, W. Zuo, and X. Feng. Weighted Nuclear Norm Minimization with Application to Image Denoising. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2862–2869. IEEE, 2014.

E. Guiraud, J. Drefs, and J. Lücke. Evolutionary Expectation Maximization. In *Genetic and Evolutionary Computation Conference (GECCO)*, 2018.

M. R. Gupta and Y. Chen. Theory and Use of the EM Algorithm. *Foundations and Trends® in Signal Processing*, 4(3):223–296, 2011.

M. Haft, R. Hofman, and V. Tresp. Generative binary codes. *Formal Pattern Analysis & Applications*, 6:269–84, 2004.

M. Henniges, G. Puertas, J. Bornschein, J. Eggert, and J. Lücke. Binary Sparse Coding. In *International Conference on Latent Variable Analysis and Signal Separation (LVA/ICA)*, pages 450–57. Springer, 2010.

F. Hirschberger, D. Forster, and J. Lücke. A Variational EM Acceleration for Efficient Clustering at Very Large Scales. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2022. doi: 10.1109/TPAMI.2021.3133763. In press.

P. O. Hoyer. Modeling receptive fields with non-negative sparse coding. *Neurocomputing*, 52-54: 547–52, 2003.

E. R. Hruschka, R. J. Campello, A. A. Freitas, et al. A Survey of Evolutionary Algorithms for Clustering. *IEEE Transactions on Systems, Man, and Cybernetics*, 39(2):133–155, 2009.

C.-W. Huang, D. Krueger, A. Lacoste, and A. Courville. Neural Autoregressive Flows. In *International Conference on Machine Learning (ICML)*, 2018.

M. C. Hughes and E. B. Sudderth. Fast Learning of Clusters and Topics via Sparse Posteriors. *arXiv preprint arXiv:1609.07521*, 2016.

A. Hyvärinen and P. Pajunen. Nonlinear Independent Component Analysis: Existence and uniqueness results. *Neural Networks*, 12(3):429–439, 1999.

S. Iizuka, E. Simo-Serra, and H. Ishikawa. Globally and Locally Consistent Image Completion. *ACM Transactions on Graphics (ToG)*, 36(4):1–14, 2017.

A. Ilin and H. Valpola. On the Effect of the Form of the Posterior Approximation in Variational Learning of ICA Models. *Neural Processing Letters*, 22(2):183–204, 2005.

R. Imamura, T. Itasaka, and M. Okuda. Zero-Shot Hyperspectral Image Denoising With Separable Image Prior. In *IEEE/CVF International Conference on Computer Vision Workshops*, 2019.

V. Jain and S. Seung. Natural Image Denoising with Convolutional Networks. In *Advances in Neural Information Processing Systems*, 2009.

E. Jang. Gumbel-Softmax. *GitHub repository*, 2016. URL `https://github.com/ericjang/gumbel-softmax`. Accessed: 2021-08-03.

E. Jang, S. Gu, and B. Poole. Categorical Reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations (ICLR)*, 2017.

Y. Jernite, Y. Halpern, and D. Sontag. Discovering Hidden Variables in Noisy-Or Networks using Quartet Tests. In *Advances in Neural Information Processing Systems*, 2013.

M. Jordan, Z. Ghahramani, T. Jaakkola, and L. Saul. An Introduction to Variational Methods for Graphical Models. *Machine Learning*, 37:183–233, 1999.

D. P. Kingma and P. Dhariwal. Glow: Generative Flow with Invertible 1x1 Convolutions. In *Advances in Neural Information Processing Systems*, 2018.

D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations (ICLR)*, 2014.

D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling. Improved Variational Inference with Inverse Autoregressive Flow. In *Advances in Neural Information Processing Systems*, 2016.

A. Krull, T.-O. Buchholz, and F. Jug. Noise2Void - Learning Denoising from Single Noisy Images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2129–2137, 2019.

H. Lee, A. Battle, R. Raina, and A. Ng. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems*, volume 20, pages 801–08, 2007.

J. Lehtinen, J. Munkberg, J. Hasselgren, S. Laine, T. Karras, M. Aittala, and T. Aila. Noise2Noise: Learning Image Restoration without Clean Data. In *International Conference on Machine Learning (ICML)*, volume 80, pages 2965–2974, 2018.

X. Li. Patch-based Image Interpolation: Algorithms and Applications. In *International Workshop on Local and Non-Local Approximation in Image Processing*, pages 1–6, 2008.

Y. Li, S. Liu, J. Yang, and M.-H. Yang. Generative Face Completion. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3911–3919. IEEE, 2017.

R. J. Little and D. B. Rubin. *Statistical Analysis with Missing Data.* Wiley Series in Probability and Statistics. John Wiley & Sons, 1st edition, 1987.

I. Loshchilov and F. Hutter. CMA-ES for Hyperparameter Optimization of Deep Neural Networks. In *International Conference on Learning Representations (ICLR) Workshop*, pages 513–520, 2016.

J. Lücke. Truncated Variational Expectation Maximization. *arXiv preprint, arXiv:1610.03113*, 2019.

J. Lücke and J. Eggert. Expectation Truncation And the Benefits of Preselection in Training Generative Models. *Journal of Machine Learning Research*, 11:2855–900, 2010.

J. Lücke and D. Forster. k-means as a variational EM approximation of Gaussian mixture models. *Pattern Recognition Letters*, 125:349–356, 2019.

J. Lücke and M. Sahani. Maximal Causes for Non-linear Component Extraction. *Journal of Machine Learning Research*, 9:1227–67, 2008.

J. Lücke and A.-S. Sheikh. Closed-Form EM for Sparse Coding and Its Application to Source Separation. In *International Conference on Latent Variable Analysis and Signal Separation (LVA/ICA)*, pages 213–221, 2012.

J. Lücke, Z. Dai, and G. Exarchakis. Truncated Variational Sampling for "Black Box" Optimization of Generative Models. In *International Conference on Latent Variable Analysis and Signal Separation (LVA/ICA)*, pages 467–478, 2018.

D. J. C. Mackay. Local Minima, Symmetry-breaking, and Model Pruning in Variational Free Energy Minimization. *Inference Group, Cavendish Laboratory, Cambridge, UK*, 2001.

D. J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

J. Mairal, M. Elad, and G. Sapiro. Sparse Representation for Color Image Restoration. *IEEE Transactions on Image Processing*, 17(1):53–69, 2008.

J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman. Non-Local Sparse Models for Image Restoration. volume 25, 2009.

S. Mohamed, K. Heller, and Z. Ghahramani. Evaluating Bayesian and L1 Approaches for Sparse Unsupervised Learning. In *International Conference on Machine Learning (ICML)*, 2012.

K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT press, 2012.

J. W. Myers, K. B. Laskey, and K. A. DeJong. Learning Bayesian Networks from Incomplete Data using Evolutionary Algorithms. In *Genetic and Evolutionary Computation Conference (GECCO)*, 1999.

R. Neal and G. Hinton. A View of the EM Algorithm that Justifies Incremental, Sparse, and other Variants. In M. I. Jordan, editor, *Learning in Graphical Models*. Kluwer, 1998.

S. Oehmcke and O. Kramer. Knowledge Sharing for Population Based Neural Network Training. In *Joint German/Austrian Conference on Artificial Intelligence*, pages 258–269, 2018.

B. Olshausen and D. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–9, 1996.

B. A. Olshausen and D. J. Field. Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision Research*, 37(23):3311–3325, 1997.

M. Opper and C. Archambeau. The Variational Gaussian Approximation Revisited. *Neural Computation*, 21(3):786–792, 2009.

M. Opper and O. Winther. Expectation Consistent Approximate Inference. *Journal of Machine Learning Research*, 6(Dec):2177–2204, 2005.

Y. Park, C. Kim, and G. Kim. Variational Laplace Autoencoders. In *International Conference on Machine Learning (ICML)*, pages 5032–5041, 2019.

D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. Context Encoders: Feature Learning by Inpainting. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2536–2544. IEEE, 2016.

F. Pernkopf and D. Bouchaffra. Genetic-based EM algorithm for learning Gaussian mixture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 27(8):1344–1348, 2005.

M. Prakash, K. Alexander, and F. Jug. Fully Unsupervised Diversity Denoising with Convolutional Variational Autoencoders. In *International Conference on Learning Representations (ICLR)*, 2021.

J. Prellberg and O. Kramer. Limited Evaluation Evolutionary Optimization of Large Neural Networks. In *Joint German/Austrian Conference on Artificial Intelligence*, pages 270–283, 2018.

R. Ranganath, S. Gerrish, and D. Blei. Black Box Variational Inference. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 814–822, 2014.

E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-Scale Evolution of Image Classifiers. In *International Conference on Machine Learning (ICML)*, pages 2902–2911, 2017.

D. J. Rezende and S. Mohamed. Variational Inference with Normalizing Flows. *International Conference on Machine Learning (ICML)*, 2015.

D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In *International Conference on Machine Learning (ICML)*, 2014.

S. Roth and M. J. Black. Fields of Experts. *International Journal of Computer Vision (IJCV)*, 82 (2):205, 2009.

M. Rotmensch, Y. Halpern, A. Tlimat, S. Horng, and D. Sontag. Learning a Health Knowledge Graph from Electronic Medical Records. *Scientific Reports*, 7(1):5994, 2017.

T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *arXiv preprint arXiv:1703.03864*, 2017.

L. K. Saul and M. I. Jordan. Exploiting Tractable Substructures in Intractable Networks. *Advances in Neural Information Processing Systems*, 1995.

L. K. Saul, T. Jaakkola, and M. I. Jordan. Mean Field Theory for Sigmoid Belief Networks. *Journal of Artificial Intelligence Research*, 4(1):61–76, 1996.

U. Schmidt, Q. Gao, and S. Roth. A Generative Perspective on MRFs in Low-Level Vision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1751–1758, 2010.

M. Seeger. Bayesian Inference and Optimal Design for the Sparse Linear Model. *Journal of Machine Learning Research*, 9:759–813, 2008.

J. G. Serra, M. Testa, R. Molina, and A. K. Katsaggelos. Bayesian K-SVD Using Fast Variational Inference. *IEEE Transactions on Image Processing*, 26(7):3344–3359, 2017.

A.-S. Sheikh and J. Lücke. Select-and-Sample for Spike-and-Slab Sparse Coding. In *Advances in Neural Information Processing Systems*, 2016.

A.-S. Sheikh, J. A. Shelton, and J. Lücke. A Truncated EM Approach for Spike-and-Slab Sparse Coding. *Journal of Machine Learning Research*, 15:2653–2687, 2014.

A.-S. Sheikh, N. S. Harper, J. Drefs, Y. Singer, Z. Dai, R. E. Turner, and J. Lücke. STRFs in primary auditory cortex emerge from masking-based statistics of natural sounds. *PLOS Computational Biology*, 15(1):e1006595, 2019.

J. A. Shelton, J. Bornschein, A.-S. Sheikh, P. Berkes, and J. Lücke. Select and Sample – A Model of Efficient Neural Inference and Learning. *Advances in Neural Information Processing Systems*, 24: 2618–2626, 2011.

J. A. Shelton, J. Gasthaus, Z. Dai, J. Lücke, and A. Gretton. GP-Select: Accelerating EM Using Adaptive Subspace Preselection. *Neural Computation*, 29(8):2177–2202, 2017.

A. Shocher, N. Cohen, and M. Irani. “Zero-Shot” Super-Resolution using Deep Internal Learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3118–3126. IEEE, 2018.

T. Singliar and M. Hauskrecht. Noisy-OR Component Analysis and its Application to Link Analysis. *Journal of Machine Learning Research*, 7:2189–2213, 2006.

K. O. Stanley and R. Miikkulainen. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

C. Steinruecken, E. Smith, D. Janz, J. Lloyd, and Z. Ghahramani. The Automatic Statistician. In *Automated Machine Learning*. Springer, 2019.

M. Suganuma, S. Shirakawa, and T. Nagao. A Genetic Programming Approach to Designing Convolutional Neural Network Architectures. In *Genetic and Evolutionary Computation Conference (GECCO)*, 2017.

J. Sun and M. F. Tappen. Learning Non-Local Range Markov Random field for Image Restoration. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2745–2752, 2011.

Y. Tai, J. Yang, X. Liu, and C. Xu. MemNet: A Persistent Memory Network for Image Restoration. In *IEEE International Conference on Computer Vision (ICCV)*, pages 4539–4547. IEEE, 2017.

C. Tian, Y. Xu, and W. Zuo. Image denoising using deep CNN with batch renormalization. *Neural Networks*, 121:461–473, 2020.

G. Tian, Y. Xia, Y. Zhang, and D. Feng. Hybrid Genetic and Variational Expectation-Maximization Algorithm for Gaussian-Mixture-Model-Based Brain MR Image Segmentation. *IEEE Transactions on Information Technology in Biomedicine*, 15(3):373–380, 2011.

M. K. Titsias and M. Lázaro-Gredilla. Spike and Slab Variational Inference for Multi-Task and Multiple Kernel Learning. In *Advances in Neural Information Processing Systems*, 2011.

J. Tohka, E. Krestyannikov, I. D. Dinov, A. M. Graham, D. W. Shattuck, U. Ruotsalainen, and A. W. Toga. Genetic algorithms for finite mixture model based voxel classification in neuroimaging. *IEEE Transactions on Medical Imaging*, 26(5):696–711, 2007.

R. E. Turner and M. Sahani. *Two problems with variational expectation maximisation for time series models*, page 104–124. Cambridge University Press, 2011.

D. Ulyanov, A. Vedaldi, and V. Lempitsky. Deep Image Prior. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9446–9454, 2018.

J. H. van Hateren and A. van der Schaaf. Independent component filters of natural images compared with simple cells in primary visual cortex. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 265:359–66, 1998.

E. Vértes and M. Sahani. Flexible and accurate inference and learning for deep generative models. In *Advances in Neural Information Processing Systems*, 2018.

C. Yang, X. Lu, Z. Lin, E. Shechtman, O. Wang, and H. Li. High-Resolution Image Inpainting using Multi-Scale Neural Patch Synthesis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6721–6729. IEEE, 2017.

R. A. Yeh, C. Chen, T. Yian Lim, A. G. Schwing, M. Hasegawa-Johnson, and M. N. Do. Semantic Image Inpainting with Deep Generative Models. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5485–5493. IEEE, 2017.

G. Yu, G. Sapiro, and S. Mallat. Solving Inverse Problems With Piecewise Linear Estimators: From Gaussian Mixture Models to Structured Sparsity. *IEEE Transactions on Image Processing*, 21(5): 2481–2499, 2012.

K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang. Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155, 2017.

K. Zhang, W. Zuo, and L. Zhang. FFDNet: Toward a Fast and Flexible Solution for CNN based Image Denoising. *IEEE Transactions on Image Processing*, 2018.

M. Zhou, H. Chen, J. Paisley, L. Ren, G. Sapiro, and L. Carin. Non-Parametric Bayesian Dictionary Learning for Sparse Image Representations. In *Advances in Neural Information Processing Systems*, 2009.

M. Zhou, H. Chen, J. Paisley, L. Ren, L. Li, Z. Xing, D. Dunson, G. Sapiro, and L. Carin. Nonparametric Bayesian Dictionary Learning for Analysis of Noisy and Incomplete Images. *IEEE Transactions on Image Processing*, 21(1):130–144, 2012.

S. Zhu, G. Xu, Y. Cheng, X. Han, and Z. Wang. BDGAN: Image Blind Denoising Using Generative Adversarial Networks. In *Chinese Conference on Pattern Recognition and Computer Vision*, pages 241–252, 2019.

D. Zoran and Y. Weiss. From Learning Models of Natural Image Patches to Whole Image Restoration. In *IEEE International Conference on Computer Vision (ICCV)*, pages 479–486, 2011.