SN Computer Science (2021) 2:301
https://doi.org/10.1007/s42979-021-00702-9

SURVEY ARTICLE

A Survey on Variational Autoencoders from a Green AI Perspective

Andrea Asperti[1] · **Davide Evangelista**[2] · **Elena Loli Piccolomini**[1]

Received: 5 March 2021 / Accepted: 13 May 2021 / Published online: 27 May 2021

Abstract
Variational Autoencoders (VAEs) are powerful generative models that merge elements from statistics and information theory with the flexibility offered by deep neural networks to efficiently solve the generation problem for high-dimensional data. The key insight of VAEs is to learn the latent distribution of data in such a way that new meaningful samples can be generated from it. This approach led to tremendous research and variations in the architectural design of VAEs, nourishing the recent field of research known as unsupervised representation learning. In this article, we provide a comparative evaluation of some of the most successful, recent variations of VAEs. We particularly focus the analysis on the energetic efficiency of the different models, in the spirit of the so-called Green AI, aiming both to reduce the carbon footprint and the financial cost of generative techniques. For each architecture, we provide its mathematical formulation, the ideas underlying its design, a detailed model description, a running implementation and quantitative results.

Keywords Generative modeling · Variational Autoencoders · Green AI

Introduction

Data generation, which is the task of generating new realistic samples given a set of training data, is a fascinating problem of AI, with many relevant applications in different areas, spanning from computer vision, to natural language processing and medicine. Due to the curse of dimensionality, the problem was practically hopeless to solve, until Deep Neural Networks enabled the scalability of the required techniques via learned approximators. In recent years, deep generative models have gained a lot of attention in the deep learning community, not just for their amazing applications, but also for the fundamental insight they provide on the encoding mechanisms of Neural Networks, the extraction of deep features, and the latent representation of data.

✉ Andrea Asperti
andrea.asperti@unibo.it

Davide Evangelista
davide.evangelista5@unibo.it

Elena Loli Piccolomini
elena.loli@unibo.it

[1] Department of Informatics: Science and Engineering (DISI), University of Bologna, Bologna, Italy

[2] Department of Mathematics, University of Bologna, Bologna, Italy

In spite of the successful results, deep generative modeling remains one of the most complex and expensive tasks in AI. Training a complex generative model typically requires a lot of time and computational resources. To make a couple of examples, the hyper-realistic Generative Adversarial Network for face generation in [36] required training on 8 Tesla V100 GPUs for 4 days; the training of BERT [18], a well-known generative model for NLP, takes about 96 h on 64 TPU2 chips.

As remarked in [51], this computational cost has huge implications, both from the ecological point of view, and for the increasing difficulties for academics, students, and researchers, in particular those from emerging economies, to do competitive, state of the art research. As a good practice in Deep Learning, one should give detailed reports about the financial cost of training and running models, in such a way to promote the investigation of increasingly efficient methods.

In this article, we offer a comparative evaluation of some recent generative models. To make the investigation more focused and exhaustive, we restricted the analysis to a single class of models: the so called Variational Autoencoders [38, 48] (VAEs).

Variational Autoencoders are becoming increasingly popular inside the scientific community [53, 60, 61], both due to their strong probabilistic foundation, that will be recalled

in "Theoretical Background", and the precious insight on the latent representation of data. However, in spite of the remarkable achievements, the behaviour of Variational Autoencoders is still far from satisfactory; there is a number of well-known theoretical and practical challenges that still hinder this generative paradigm (see "The Vanilla VAE and Its Problems"), and whose solution drove the recent research on this topic. We try to give an exhaustive presentation of most of the VAE variants in the literature, relating them to the implementation and theoretical issues they were meant to address.

Hence, we focus on a restricted subset of recent architectures that, in our opinion, deserve a deeper investigation, for their paradigmatic nature, the elegance of the underlying theory, or some key architectural insight. The three categories of models that we shall compare are the Two-stage model [16], the Regularized Autoencoder[1] [39], and some versions of Hierarchical Autoencoders. In the latter class, we provide a detailed analysis of the recent Nouveau VAE [58]; however, its complexity trespasses our computing facilities, so we investigate a much simpler model, and an interesting variant exploiting Feature-wise Linear Modulation [44] at high scales.

One of the metrics used to compare these models is their energetic efficiency, in the spirit of the emerging paradigm known as Green AI [51], aiming to assess performance/efficiency trade-offs. Specifically, for each architecture, we provide a precise mathematical formulation, a discussion of the main ideas underlying their design, a detailed model description, a running implementation in TensorFlow 2 freely available on our GitHub repository https://github.com/devangelista2/GreenVAE, and quantitative results.

Structure of the Article

The article is meant to offer a self-contained introduction to the topic of Variational Autoencoders, just assuming a basic knowledge of neural networks. In the next section, we start with the theoretical background, discussing the strong and appealing probabilistic foundation of this class of generative models. In the following section, we address the way theory is translated into a vanilla neural net implementation, and introduce the many issues arising from this operation: balancing problems in the loss function, posterior collapse, aggregate posterior vs. prior mismatch, blurriness and disentanglement.

In the next three sections, we give a detailed mathematical introduction to the three classes of models for which we provide a deeper investigation, namely the Two-Stage approach, the regularized VAE and hierarchical models. After these sections, our experimental setting is described: we discuss the metrics used for the comparison, and provide a detailed description of the neural network architectures. In the penultimate section, we provide the results of our experimentation, making a critical discussion. In the conclusive section, we summarize the content of the article and draw a few considerations on the future of this field, and the challenges ahead.

Theoretical Background

In this section, we give a formal, theoretical introduction to Variational Autoencoders (VAEs), deriving the so called Evidence Lower Bound (ELBO) adopted as a learning objective for this class of models.

To deal with the problem of generating realistic data points $x \in \mathbb{R}^d$ given a dataset $\mathbb{D} = \{x^{(1)}, \dots, x^{(N)}\}$, generative models usually make the assumption that there exists a ground-truth distribution μ_{GT} supported on a low-dimensional manifold $\chi \subseteq \mathbb{R}^d$ with dimension $k < d$, absolutely continuous with respect to the Hausdorff measure on χ and with density $p_{gt}(x)$. With this assumption, one can rewrite

$$p_{gt}(x) = \int_{\mathbb{R}^k} p_{gt}(x,z)dz = \int_{\mathbb{R}^k} p_{gt}(x|z)p(z)dz = \mathbb{E}_{p(z)}[p_{gt}(x|z)], \tag{1}$$

where $z \in \mathbb{R}^k$ is the latent variable associated with x, distributed with a simple distribution $p(z)$ named *prior distribution*.

The idea behind generative models is that if we can learn a good approximation of $p_{gt}(x|z)$ from the data, then we can use that approximation to generate new samples with ancestral sampling, that is,

- Sample $z \sim p(z)$.
- Generate $x \sim p_{gt}(x|z)$.

For this reason, it is common to define a parametric family of probability distributions $\mathcal{P}_\theta = \{p_\theta(x|z)|\theta \in \mathbb{R}^s\}$ with a neural network, and to find θ^* such that

$$\theta^* = \arg\max_\theta \mathbb{E}_{\mathbb{D}}[\log p_\theta(x)] = \arg\max_\theta \mathbb{E}_{\mathbb{D}}\left[\log \int_{\mathbb{R}^k} p_\theta(x|z)p(z)dz\right], \tag{2}$$

i.e. the Maximum Likelihood Estimation (MLE).

Unfortunately, (2) is usually computationally infeasible. For this reason, VAEs define another probability distribution $q_\phi(z|x)$ named *encoder distribution* which describes the relationship between a data point $x \in \chi$ and its latent variable $z \in \mathbb{R}^k$ and optimizes ϕ and θ such that:

[1] Strictly speaking, this is not a Variational model, but it helps in understanding them.

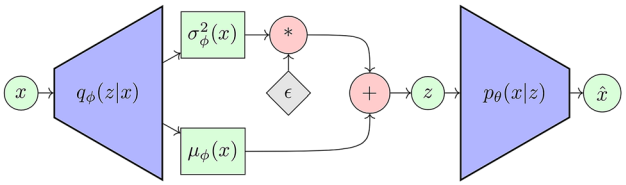

Fig. 1 A diagram representing the VAE architecture. The stochastic component ϵ in the gray diamond is sampled from G(0,I)

$$\theta^*, \phi^* = \arg\min_{\theta,\phi} \mathbb{E}_{\mathbb{D}}[D_{KL}(q_\phi(z|x)||p_\theta(z|x))], \tag{3}$$

where $D_{KL}(q_\phi(z|x)||p_\theta(z|x)) = \mathbb{E}_{q_\phi(z|x)}[\log q_\phi(z|x) - \log p_\theta(z|x)]$ is the Kullback–Leibler divergence between $q_\phi(z|x)$ and $p_\theta(z|x)$.

But

$$\begin{aligned} &D_{KL}(q_\phi(z|x)||p_\theta(z|x)) \\ &\quad = \mathbb{E}_{q_\phi(z|x)}[\log q_\phi(z|x) - \log p_\theta(z|x)] \\ &\quad = \mathbb{E}_{q_\phi(z|x)}[\log q_\phi(z|x) - \log p_\theta(x|z) - \log p_\theta(z) + \log p_\theta(x)] \\ &\quad = D_{KL}(q_\phi(z|x)||p(z)) - \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] + \log p_\theta(x). \end{aligned} \tag{4}$$

Thus,

$$\begin{aligned} &\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)||p(z)) \\ &= \log p_\theta(x) - D_{KL}(q_\phi(z|x)||p_\theta(z|x)) \\ &\leq \log p_\theta(x), \end{aligned} \tag{5}$$

since $D_{KL}(q_\phi(z|x)||p_\theta(z|x)) \geq 0$, which implies that the Left Hand Side of the equation above is a lower bound for the loglikelihood of $p_\theta(x)$. For this reason, it is usually called Evidence Lower BOund (*ELBO*).

Since ELBO is more tractable than MLE, it is used as the cost function for the training of neural network to optimize both θ and ϕ:

$$\mathcal{L}_{\theta,\phi}(x) := \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)||p(z)) \tag{6}$$

$$\mathcal{L}_{\theta,\phi} := \mathbb{E}_{\mathbb{D}}[\mathcal{L}_{\theta,\phi}(x)]. \tag{7}$$

It is worth to remark that ELBO has a form resembling an autoencoder, where the term $q_\phi(z|x)$ maps the input x to its latent representation z, and $p_\theta(x|z)$ decodes z back to x. Figure 1 shows a diagram representing the basic VAE structure.

For generative sampling, we forget the encoder and just exploit the decoder, sampling the latent variables according to the prior distribution $p(z)$ (that must be known).

The Vanilla VAE and Its Problems

In this section, we explain how the theoretical form of the ELBO (Eq. 6) can be translated into a numerical loss function exploitable for training of neural networks. This will allow us to point out some of the typical problems that affect this architecture and whose solution drove the design of the variants discussed in the sequel.

In the vanilla VAE, we assume $q_\phi(z|x)$ to be a Gaussian (spherical) distribution $G(\mu_\phi(x), \sigma^2_\phi(x))$, so that learning $q_\phi(z|x)$ amounts to learning its two first moments.

Similarly, we assume $p_\theta(x|z)$ has a Gaussian distribution around a decoder function $\mu_\theta(z)$. The functions $\mu_\phi(x)$, $\sigma^2_\phi(x)$ and $\mu_\theta(z)$ are modelled by deep neural networks. We remark that knowing the variance of latent variables allows sampling during training.

If the model approximating the decoder function $\mu_\theta(z)$ is sufficiently expressive (that is case, for deep neural networks), the shape of the prior distribution $p(z)$ does not really matter, and for simplicity it is assumed to be a normal distribution $p(z) = G(0, I)$. The term $D_{KL}(q_\phi(z|x)||p(z))$ is hence the KL-divergence between two Gaussian distributions $G(\mu_\phi(x), \sigma^2_\phi(x))$ and $G(0, I)$ and it can be computed in closed form as

$$\begin{aligned} &D_{KL}(G(\mu_\phi(x), \sigma_\phi(x)), G(0, I)) = \\ &\quad \frac{1}{2} \sum_{i=1}^{k} \mu_\phi(x)_i^2 + \sigma^2_\phi(x)_i - log(\sigma^2_\phi(x)_i) - 1, \end{aligned} \tag{8}$$

where k is the dimension of the latent space. The previous equation has an intuitive explanation, as a cost function. By minimizing $\mu_\phi(x)$, when x is varying on the whole dataset, we are centering the latent space around the origin (i.e. the mean of the prior). The other component is preventing the variance $\sigma^2_\phi(x)$ to drop to zero, implicitly forcing a better coverage of the latent space.

Coming to the reconstruction loss $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]$, under the Gaussian assumption, the logarithm of $p_\theta(x|z)$ is the quadratic distance between x and its reconstruction $\mu_\theta(z)$; the variance of this Gaussian distribution can be understood as a parameter balancing the relative importance between reconstruction error and KL-divergence [20].

The problem of integrating sampling with backpropagation during training is solved by the well-known reparametrization trick proposed in [38, 48], where the sample is performed using a standard distribution (outside of the backpropagation flow) and this value is rescaled with $\mu_\phi(x)$ and $\sigma_\phi(x)$.

The basic model of the Vanilla VAE that we just outlined is unfortunately hindered by several known theoretical and practical challenges. In the next Sections, we give a short list of important topics which have been investigated in the

literature, along with a short discussion of the main works addressing them.

The Balancing Issue

The VAE loss function is the sum of two distinct components, with somehow contrasting effects

$$\mathcal{L}_{\theta,\phi}(x) := \underbrace{\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]}_{\text{log-likelihood}} - \gamma \underbrace{D_{KL}(q_\phi(z|x)||p(z))}_{\text{KL-divergence}}. \quad (9)$$

The log-likelihood loss is just meant to improve the quality of reconstruction, while the Kullback–Leibler component is acting as a regularizer, pushing the aggregate inference distribution $q_\phi(z) = \mathbb{E}_{\mathbb{D}}[q_\phi(z|x)]$ towards the desired prior $p(z)$.

Log-likelihood and KL-divergence are frequently balanced by a suitable parameter, allowing to tune their mutual relevance. The parameter is called γ, in this context, and it is considered as a normalizing factor for the reconstruction loss.

Privileging log-likelihood will improve the quality of reconstruction, neglecting the shape of the latent space (with ominous effects on generation). Privileging KL-divergence typically results in a smoother and normalized latent space, and more disentangled features [11, 29]; this usually comes at the cost of a more noisy encoding, finally resulting in more blurriness in generated images. [1].

Discovering a good balance between these components is a crucial aspect for an effective training of VAEs.

Several techniques for the calibration of γ have been investigated in the literature, comprising an annealed optimization schedule [8] or a policy enforcing minimum KL contribution from subsets of latent units [37]. These schemes typically require hand-tuning and, as observed in [63], they easily risk to interfere with the principled regularization scheme that is at the core of VAEs.

An alternative possibility, investigated in [16], consists in *learning* the correct value for the balancing parameter during training, that also allows its automatic calibration along the training process.

In [2] it is observed that considering the objective function used in [16] to learn γ, the optimal γ parameter is in fact proportional to the current reconstruction error; so learning can be replaced by a mere computation, using, e.g. a running average. This has a simple and intuitive explanation: what matters is to try to maintain a *fixed* balance between the two components during training: if the reconstruction error decreases, we must proportionally decrease the KL component that could otherwise prevail, preventing further improvements. The technique in [2] is simple and effective: we shall implicitly adopt it in all our VAE models, unless explicitly stated differently.

A similar technique has been recently investigated in [52], where the KL-divergence is used as a feedback during model training for dynamically tuning the balance of the two components.

Variable Collapse Phenomenon

The KL-divergence component of the VAE loss function typically induces a parsimonious use of latent variables, some of which may be altogether neglected by the decoder, possibly resulting in an under-exploitation of the network capacity; if this is a beneficial side effect or regularization [5, 16] or an issue to be solved ([10, 46, 57, 63]), it is still debated.

The variable collapse phenomenon has a quite intuitive explanation. If, during training, a latent variable gives a modest contribution for the reconstruction of the input (in comparison with other variables), then the Kullback–Leibler divergence may prevail, pushing the mean towards 0 and the standard deviation towards 1. This will make the latent variable even more noisy, in a vicious cycle that will eventually induce the network to completely ignore the latent variable (see Fig. 2, Left).

As described in [3], one can easily get an empirical evidence of the phenomenon by adding some artificial noise to a variable and monitoring its evolution during training (Fig. 2, Right). The contribution of a latent variable to reconstruction is computed as the difference between the reconstruction loss when the variable is masked with respect to the case when it is normally taken into account; we call this information *reconstruction gain*.

When the reconstruction gain of the variable is becoming less than the KL-divergence, the variable gets ignored by the network: its correspondent mean value will collapse to 0 (independently from x) and its sampling variance is pushed to 1. Sampling has no impact on the network, precisely because the variable is ignored by the decoder.

The variable collapse phenomenon is, at some extent, reversible. However, reactivating a collapsed variable is not a completely trivial operation for a network, probably due to saturation effects and vanishing gradients.

Aggregate Posterior vs. Expected Prior Mismatch

The crucial point of VAEs is to learn an encoder producing an *aggregate posterior distribution* $q_\phi(z) = \mathbb{E}_{\mathbb{D}}[q_\phi(z|x)]$ close to the prior $p(z)$. If this objective is not achieved, generation is doomed to fail.

Before investigating ways to check the intended behavior, let us discuss how the Kullback–Leibler divergence term in (9) acts on the distance $q_\phi(z)$ and $p(z)$. So, let us average over all x (we omit the ϕ subscript):

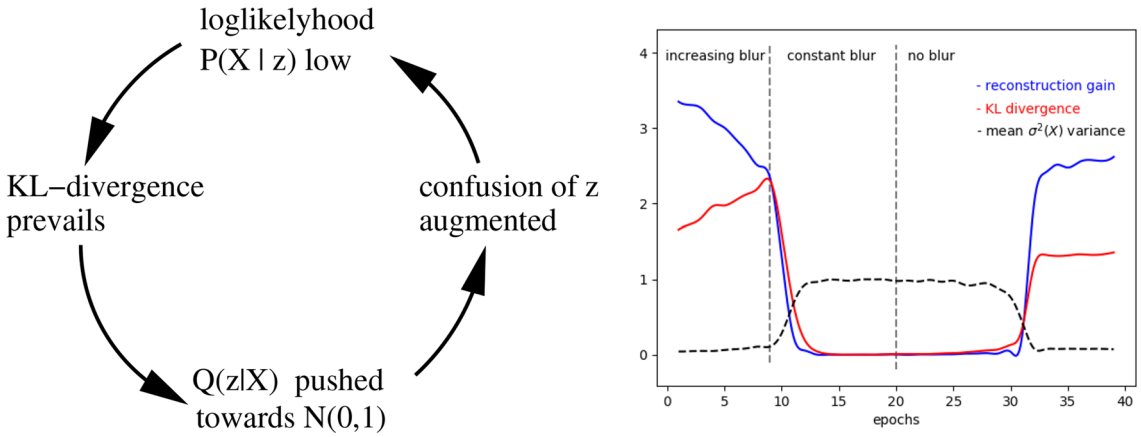

Fig. 2 (Left) The vicious cycle leading to the variable collapse. (Right) An empirical demonstration of the phenomenon: we apply a progressive noise to a latent variable, reducing its contribution to reconstruction; at some point, KL-divergence prevails, enlarging the sampling variance of the variable and making it even more noisy; the phenomenon has catastrophic nature, leading to a complete collapse of the variable. If we remove the artificial noise, the variable gets reactivated. Pictures borrowed from [3]

$$
\begin{aligned}
&\mathbb{E}_{p_{gt}(x)}[D_{KL}(q(z|x)|p(z))] \\
&= -\mathbb{E}_{p_{gt}(x)}[\mathcal{H}(q(z|x))] + \mathbb{E}_{p_{gt}(x)}[\mathcal{H}(q(z|x), p(z))] && \text{by def. of KL} \\
&= -\mathbb{E}_{p_{gt}(x)}[\mathcal{H}(q(z|x))] + \mathbb{E}_{p_{gt}(x)}[\,\mathbb{E}_{q(z|x)}[\log p(z)]] && \text{by def. of entropy} \\
&= -\mathbb{E}_{p_{gt}(x)}[\mathcal{H}(q(z|x))] + \mathbb{E}_{q(z)}[\log p(z)] && \text{by marginalization} \\
&= \underbrace{-\mathbb{E}_{p_{gt}(x)}[\mathcal{H}(q(z|x))]}_{\text{Avg. Entropy of } q(z|x)} + \underbrace{\mathcal{H}(q(z), p(z))}_{\text{Cross-entropy of } q(x) \text{ vs } p(z)} && \text{by def. of entropy}
\end{aligned}
\tag{10}
$$

By minimizing the cross-entropy between $q(z)$ and $p(z)$ we are pushing one towards the other. Jointly, we try to augment the entropy of $q(z|x)$; under the assumption that $q(z|x)$ is Gaussian, its entropy is $\frac{1}{2}log(e\pi\sigma^2)$: we are thus enlarging the (mean) variance, further improving the coverage of the latent space, essential for generative sampling.

As a simple sanity check, one should always monitor the moments of the aggregate posterior distribution $q(z)$ during training: the mean should be 0, and the variance 1. Since collapsed variables could invalidate this computation (both mean and variance are close to 0), it is better to use an alternative rule [4] : if we look at $q(z) = \mathbb{E}_{p_{gt}(x)}[q(z|x)]$ as a Gaussian Mixture Model (GMM), its variance σ^2_{GMM} is given by the sum of the variances of the means $\mathbb{E}_{p_{gt}(x)}[\mu_\phi(x)^2]$ and the mean of the variances $\mathbb{E}_{p_{gt}(x)}[\sigma^2_\phi(x)]$ of the components (supposing that $\mathbb{E}_{p_{gt}(x)}[\mu_\phi(x)]$=0):

$$
\sigma^2_{GMM} = \mathbb{E}_{p_{gt}(x)}[\mu_\phi(x)^2] + \mathbb{E}_{p_{gt}(x)}[\sigma^2_\phi(x)] = 1, \tag{11}
$$

where in this case $\mu_\phi(x)$ and $\sigma^2_\phi(x)$ are the values computed by the encoder.

This is called *variance law* in [4], and can be used to verify that the regularization effect of the KL-divergence is properly working.

The big problem is that, even if the two first moments of $q(z)$ are 0 and 1, this does not imply that it should look like a Normal (meaning that the KL-divergence got lost in some local minimum, contenting itself with adjusting the first moments of the distributions).

The potential mismatch between $q(z)$ and the expected prior $p(z)$ is a problematic aspect of VAEs that, as observed by many authors [4, 30, 49], could seriously compromise the whole generative framework. Attempts to solve this issue have been made both by acting on the loss function [55] or by exploiting more complex priors [7, 37, 56].

An interesting possibility, that has been recently deployed in the Hyperspherical VAE [17], consists in replacing the Gaussian Distribution with the von Mises-Fisher (vMF) distribution [24], that is a continuous distribution on the N-dimensional sphere in use in *directional statistics*.

An orthogonal, drastic alternative consists in renouncing to work in the comfortable setting of continuous latent variables, passing instead in the discrete domain. This approach is at the core of the Vector Quantized VAE [59] (VQ-VAE):

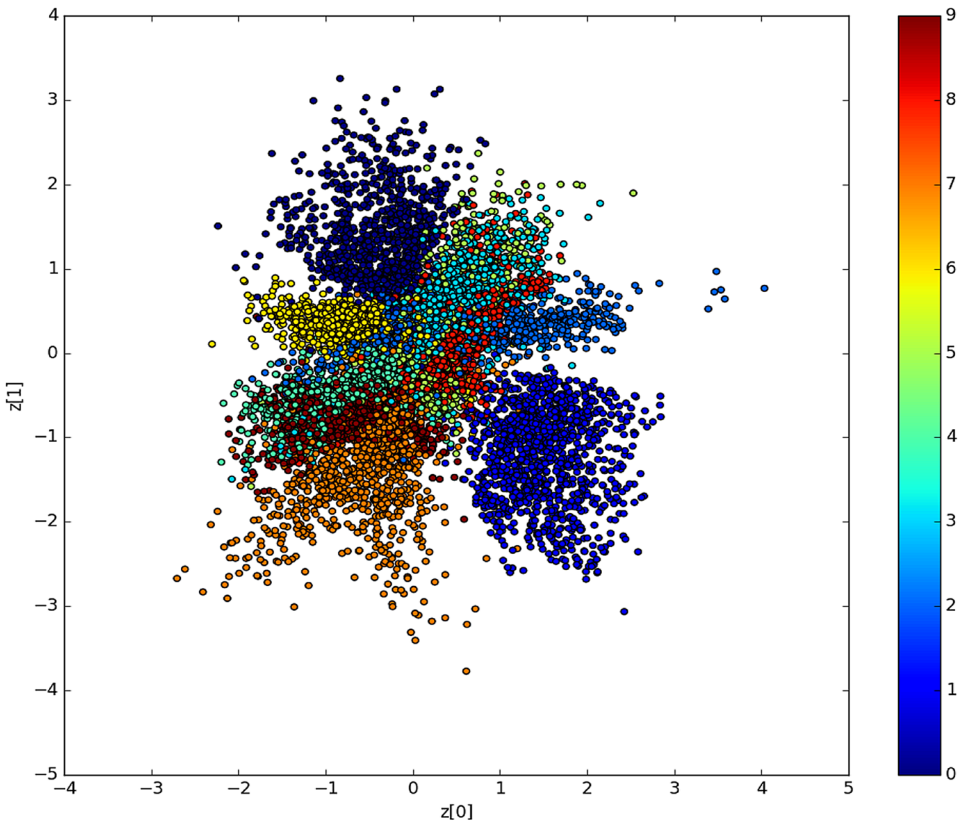

Fig. 3 Latent encoding of MNIST digits in a latent space of dimension 2. Digits in different categories are represented with a different color. Observe (1) the overall (rough) Gaussian-like disposition of all digits and (2) the typical organization in clusters, in contrast with the uni-modal objective of KL-regularization

each latent variable is forced to occupy a position in a finitely sampled space, so that we can treat each latent variable as a k-dimensional vector in a space of dimension d. This discrete encoding is exploited during sampling, where the prior is learnt via a suitable autoregressive technique.

Clustering, GMM and Two-Stage

In case input data are divided into subcategories (as in the case of MNIST and Cifar10), or have macroscopic attributes like, say, a different color for hairs in the case of CelebA, we could naturally expect to observe this information in the latent encoding of data [62]. In other words, we could imagine the latent space to be organized in *clusters*, (possibly) reflecting macroscopic features of data.

To make an example, in Fig. 3 it is described the latent encoding of MNIST digits, with a different color for each class in the range 0–9.

We can clearly observe that different digits naturally organize themselves in separate clusters. While the overall distribution still has a Gaussian-like shape, the presence of clusters may obviously contrast with the required smoothness of the internal encoding, introducing regions with higher/lower probability densities. Observe, e.g. the gaps between some of the clusters: sampling in such a region will eventually result in a poor generative output. In other words, clustering could be one of the main source for the mismatch between the prior and the aggregate posterior.

While the phenomenon is evident in a low-dimensional setting, it is more difficult to observe and testify it in higher dimensions. Remember that one of the VAE assumptions is that, as far as you have a sufficiently expressive decoder, the prior does not really matter since the decoder will be able to turn each distribution into the desired one [20].

Still, it makes sense to try to exploit clustering, and a natural approach consists in using a GMM model. Several works have been done in this direction. The simplest approach, followed in [39], is to superimpose a GMM of fixed dimension on the latent space via ex-post estimation using standard machine learning techniques (this is also the approach we shall follow in some of our tests). Alternatively, the GMM model can be *learned*. In the Variational Deep Embedding approach [62] (VaDE), that essentially provides an unsupervised clustering model, the relevant statistics of the GMM are estimated via Maximum Likelihood Estimation, in a way similar to the Vanilla case (see also [19] for a similar, slightly more sophisticated approach).

In the so-called Two-Stage model [16] a second VAE is trained to learn an accurate approximation of $q(z)$; samples from a Normal distribution are first used to generate samples of $q(z)$, passed to the actual generator of data points.

We shall give an extensive discussion of to the Two-Stage approach in "Two-Stage VAE".

In [26], it is proposed to give an ex-post estimation of $q(z)$, e.g. imposing a distribution with a sufficient complexity (they consider a combination of 10 Gaussians, reflecting the ten categories of MNIST and Cifar10). A suitable regularization technique alternative to KL is used to induce the desirable smoothness of the latent space. A deeper analysis of this approach is done in "Regularized VAE (RAE)".

An additional and interesting issue of the Two-Stage model concerns the similarity measure to use as a loss function in the second stage. In [16], the traditional mean squared error and categorical cross entropy are considered. However, we discovered that *cosine distance* works amazingly better. We did not get to cosine distance by trial and error, but by a long and deep investigation on latent representations. These results will be the object of a forthcoming article.

Blurriness

Variational Autoencoders (VAEs), in comparison with alternative generative techniques, usually produce images with a characteristic and annoying blurriness. The phenomenon can also be observed in terms of the mean variance of pixels in generated images, which is significantly lower than that for data in the training set [6].

The source of the problem is not easy to identify, but it is likely due to *averaging*, implicitly underlying the VAE frameworks (and, more generally, the whole autoencoder approach). In presence of multimodal output, a loglikelihood objective typically results in averaging and hence blurriness [27].

Variational Autoencoders are intrinsically multimodal, both due to dimensionality reduction, and to the sampling process during training.

Several attempts to solve the issue acting on the reconstruction metrics have been made. Structural similarity (frequently used for deblurring purposes) does not seem to be effective [21]. Better results can be obtained by considering deep hidden features extracted from a pretrained image classification model, like e.g. VGG19 [31]. In models of the VAE-GAN family [41, 50, 64], the reconstruction loss is altogether replaced by a discriminator trying to distinguish real images from generated ones. The use of a discriminator, assessing the quality of generated data and acting on the density of the prior, is also a basic component of the recent VAEPP model (VAEs with a pullback prior) [14].

The most promising approaches are however based on iterative/hierarchical approaches [22, 28, 58]. In these architectures, following the idea of latent Gaussian models [35], the vector of latent variables z is split into L groups of latent variables $z_l, l = 1, ..., L$ and the density over the variable of interest is constructed sequentially, in terms of latent variables of lower indices. For instance, the prior $p(z)$ would be written as an autoregressive density of the following kind:

$$p(z) = \prod_{l=1}^{L} p_l(z_l|z_{<l}). \tag{12}$$

Similarly, the inference probability would be decomposed as

$$q_\phi(z|x) = \prod_{l=1}^{L} q_\phi^{(l)}(z_l|x, z_{<l}), \tag{13}$$

where $q_\phi^{(l)}(z_l|x, z_{<l})$ is the encoder density of the lth group. Suitable (iterative) neural networks modules are used to sequentially compute the relevant statistics of these distributions, in terms of previous outputs.

As an example of these architectures, the structure of NVAE will be detailed in "NVAE".

The advantage of this approach is that it usually allows to work with a larger number of latent variables, responsible for small and progressive adjustments of generated samples.

Disentanglement

Besides the task of generating new images, [11, 29] noticed that VAEs can also be used to learn an efficient way to represent the data, with important applications in transfer learning and classification.

To understand this phenomenon, suppose that there exists a set of *true* generative factors $v = (v_1, \dots, v_S) \in \mathbb{R}^S$ such that $p_{gt}(v|x) = \prod_{i=1}^{S} p_{gt}(v_i|x)$ (i.e. v are conditionally independent given x) and that each v_i encodes a meaningful feature of the data point x generated by it. Under the assumption that $k \geq S$, the latent variables $z = (z_1, \dots, z_k)$ learnt during the training are a redundant representation of v in a basis where the features are not disentangled. To learn an optimal latent representation of the input image x, it is necessary to train the network in such a way that S coordinates of z are related to v, while the other $k - S$ coordinates can be used to improve the reconstruction of x, recovering the high frequency components that are missing in v.

In β-VAE [11, 29], this constraint is imposed by noting that in the ELBO function the prior distribution $p(z) = G(0, I)$ forces the decoder $q_\phi(z|x)$ to learn a vector z where each variable is independent of each other. To improve disentanglement, we should hence induce the D_{KL} term to be as small as possible, that can be achieved by augmenting the decoder variance γ to be greater than 1. Unfortunately, since

$$\mathbb{E}_{p_{gt}(x)}[D_{KL}(q_\phi(z|x)||p(z))] = D_{KL}(q_\phi(z)||p(z)) + I_{q_\phi}(X;Z),$$

where $I_{q_\phi}(X;Z)$ is the mutual information between X and Z with respect to the joint distribution $q_\phi(x,z) = q_\phi(z|x)p_{gt}(x)$, by pushing $D_{KL}(q_\phi(z|x)||p(z))$ to zero, the mutual information between X and Z is also minimized, reducing the reconstruction efficiency of the network. This problem is addressed in [23, 43] where the ELBO is modified by adding more parameters with the intent to improve disentanglement without losing too much the performance.

Two-Stage VAE

To address the mismatch of aggregate posterior versus the expected prior, Bin Dai and David Wipf in [16] introduced the Two-Stage VAEs.

The idea behind this model is to train two different VAEs sequentially. The first VAE is used to learn a good representation $q_\phi(z|x)$ of the data in the latent space without guaranteeing exactly $q(z) = p(z)$, whereas the second VAE should learn to sample from the true $q(z)$ without using the prior distribution $p(z)$. A scheme of the implementation follows (a detailed architectural description is given in "Architecture Overview"):

- Given a data set $\mathbb{D} = \{x^{(1)}, \dots, x^{(N)}\}$, train a VAE with a fixed latent dimension k, possibly small.
- Generate latent samples $\mathcal{Z} = \{z^{(1)}, \dots, z^{(N)}\}$ via $z^{(i)} \sim q_\phi(z|x^{(i)})$, $i = 1, \dots N$. By design, these samples are distributed as $q_\phi(z) = \mathbb{E}_{p_{gt}(x)}[q_\phi(z|x)]$, but likely not as $p(z) = G(0, I)$.
- Train a second VAE with parameters (θ', ϕ') and latent variable $u \sim p(u) = G(0, I)$ of dimension k to learn the distribution $q_\phi(z)$ with $\mathcal{Z}$ as the dataset.
- Sample new images by ancestral sampling, i.e. by first sampling $u \sim p(u)$, then generate a z value by $p_{\theta'}(z|u)$ and finally $x \sim p_\theta(x|z)$.

The theoretical foundation of the Two-Stage VAE algorithm is well presented in [16]. We summarize here the main results. The two VAEs aim at separating the components of the ELBO loss function (9), by suitably using the decoder variance γ. Remarking that $p_{gt}(x)$ is the unknown data distribution which we desire to learn and that $p_\theta(x) = \mathbb{E}_{q_\phi(z)}[p_\theta(x|z)]$ is the learnt distribution, we hope that $p_\theta(x) \approx p_{gt}(x)\ \forall x$.

Unfortunately, this is not always possible. In fact, there is a critical distinction between the cases where the dimension of the data d and the latent space dimension k are equal, and the case where $d > k$.

As a matter of facts, in the first case, it is possible to prove that, under suitable assumptions, for the optimal choice of the parameters (θ^*, ϕ^*) it holds that $p_{\theta^*}(x) = p_{gt}(x)$ almost everywhere (i.e. VAEs strongly converges to the true distribution $p_{gt}(x)$). In the second case, only weak convergence, in the sense that $\int_A p_{\theta^*}(x)dx = \int_A p_{gt}(x)dx$ where A is an open subset of $\mathbb{R}^d$, can be proved (see Theorems 1 and 2 in [16]).

In the first stage, since the ambient dimension is obviously greater than the latent space dimension (i.e. $d > k$), for the previous results only a weak convergence is guaranteed; the parameter γ is chosen in this case to get a good reconstruction (Theorem 4 in [16]). In the second stage by construction the data variable z and its correspondent latent variable u have the same dimension, hence the unknown distribution $q_\phi(z)$ is exactly identified by the VAE. As a consequence it is possible to sample directly from $q_\phi(z)$, without using the prior $p(z)$, thus bypassing the problem of mismatch between the aggregate posterior and the prior distributions.

Regularized VAE (RAE)

One of the most interesting variations of vanilla VAE is the work of Partha Ghosh and Mehdi S. M. Sajjadi [26], where the authors tried to solve all the problems related to the classical VAE by completely changing the the way of approaching the problem. They pointed out that, in their typical implementation, VAEs can be seen as a regularized Autoencoder with Additive Gaussian Noise on the decoder input.

In their work, the authors argued that noise injection in decoders input can be seen as a form of regularization, since it implicitly helps to smooth the function learnt by the network.

To get a new insight into this problem, they took in consideration the distinct components of ELBO already introduced in (9):

$$\mathcal{L}_{\theta,\phi}(x) := \underbrace{\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]}_{:=\mathcal{L}_{REC}(\theta,\phi)} - \gamma \underbrace{D_{KL}(q_\phi(z|x)||p(z))}_{:=\mathcal{L}_{KL}(\phi)}, \tag{14}$$

where $\mathcal{L}_{REC}$ is a term that measures the distance between the input and the reconstruction, whereas $\mathcal{L}_{KL}$ is a regularization term that enforces the aggregate posterior to follow the prior distribution.

To show how $\mathcal{L}_{KL}(\phi)$ regularizes the loss, in [26] the Constant-Variance VAEs (CV-VAEs) [26] have been investigated, where the encoder variance $\sigma^2_\phi(x)$ is fixed for every $x \in \mathbb{D}$ and thus treated as an hyperparameter σ^2. In this situation,

$$\mathcal{L}_{REC}(\theta,\phi) = -\mathbb{E}_{q_\phi(z|x)}\left[\frac{1}{2}||x - \mu_\theta(z)||_2^2\right] \tag{15}$$

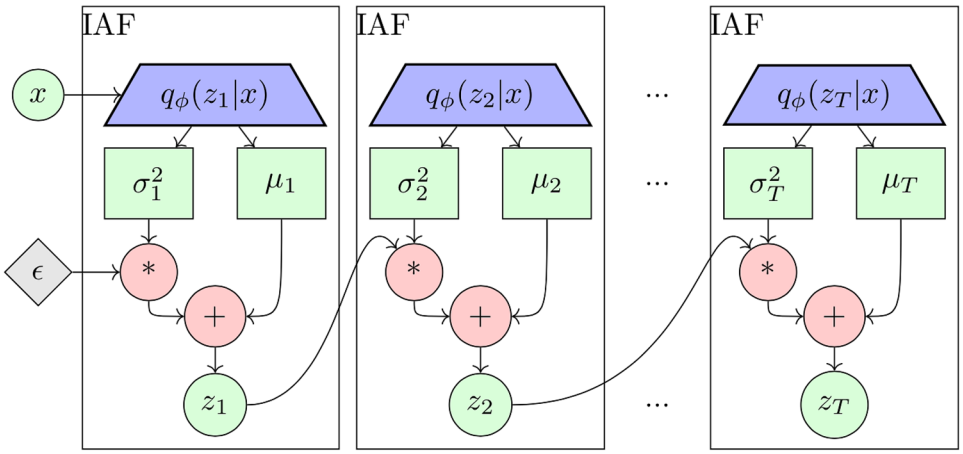

Fig. 4 A scheme of Inverse Autoregressive Flow. Each white box represents one iteration of Eq. (21), where μ_t, σ_t^2 are generated by the encoder $q_\phi(z_t|x)$

$$\mathcal{L}_{KL}(\phi) = D_{KL}(q_\phi(z|x)||p(z)) = ||\mu_\phi(x)||_2^2 + C \tag{16}$$

$$\mathcal{L}_{\theta,\phi}(x) = -\mathbb{E}_{p_{gt}}\left[\mathbb{E}_{q_\phi(z|x)}\left[\frac{1}{2}||x - \mu_\theta(z)||_2^2\right] - \gamma||\mu_\phi(x)||_2^2\right]. \tag{17}$$

We observe that the expression in (17) is a Mean Squared Error (MSE) with L_2 regularization on $\mu_\phi(x)$.

The authors proposed to substitute noise injection in the decoder input with an explicit regularization scheme in a classical CV-VAE. This is done by modifying the cost function $\mathcal{L}_{\theta,\phi} = \mathbb{E}_{p_{gt}(x)}[\mathcal{L}_{REC}(\theta,\phi) - \gamma\mathcal{L}_{KL}(\phi) - \lambda\mathcal{L}_{REG}(\theta)]$, where $\mathcal{L}_{REG}(\theta)$ is a regularizer for the decoder weights, while $\gamma, \lambda \geq 0$ are regularization parameters.

Whereas $\mathcal{L}_{REC}(\theta,\phi) = -\mathbb{E}_{q_\phi(z|x)}[\frac{1}{2}||x - \mu_\theta(z)||_2^2]$ and $\mathcal{L}_{KL}(\phi) = \frac{1}{2}||z||_2^2$ are fixed a priori by the CV-VAE architecture, $\mathcal{L}_{REG}(\theta)$ needs to be defined. The choice for $\mathcal{L}_{REG}(\theta)$ identifies the specific kind of network. Ghosh and Sajjadi proposed three possible choices for $\mathcal{L}_{REG}(\theta)$:

- *L_2-Regularization*, where $\mathcal{L}_{REG}(\theta) = ||\theta||_2^2$ is simply the weight decay on the decoder parameters.
- *Gradient penalty*, where $\mathcal{L}_{REG}(\theta) = ||\nabla\mu_\theta(\mu_\phi(x))||_2^2$ bounds the gradient norm of the decoder with respect to its input, enforcing smoothness.
- *Spectral normalization*, where each weight matrix θ_l in the decoder is normalized by an estimate of its largest singular value: $\theta_l^{SN} = \frac{\theta_l}{s(\theta_l)}$ (the estimate $s(\theta_l)$ can be easily obtained with one iteration of the power method).

Moreover, they argued that removing noise injection from the decoder input prevents from knowing the distribution of latent variables, thus losing the generative ability of the network. They solved this problem by proposing an *ex-post density estimation*, where the distribution of the latent variables is learned a posteriori, by fitting $\mathcal{Z} = \{z^{(i)}; z^{(i)} = \mu_\phi(x^{(i)})\}$ with a GMM model $q_\delta(z)$ with a fixed number of components and then sampling z from $q_\delta(z)$ to generate new samples from $p_\theta(x|z)$. The generative model defined in this way is called *Regularized Autoencoder (RAE)*.

Hierarchical Variational Autoencoder

To improve the quality of the generation in Variational Autoencoders, Kingma et al. [37] strengthened the inference network $q_\phi(z|x)$ with the powerful Normalizing Flows introduced by Rezende and Mohamed [47]. The idea of Normalizing Flows is to begin with a latent variable z_0 sampled by a simple distribution $q_\phi(z_0|x)$, and to iteratively construct more complex variables by applying transformations $z_t = f_t(z_{t-1})$ for $t = 1, \ldots, T$. By observing that the D_{KL} expression is

$$D_{KL}(q_\phi(z_T|z_{<T}, x)||p(z_T)) = \mathbb{E}_{q_\phi(z_T|z_{<T},x)}\big[\log q_\phi(z_T|z_{<T}, x) - \log p(z_T)\big], \tag{18}$$

its implementation requires the computation of the logarithm of $q_\phi(z_T|, z_{<T}, x)$. If the functions $f_t(\cdot)$ are simple enough, it is possible to efficiently use them to compute $\log q_\phi(z_T|z_{<T}, x)$ as:

$$\log q_\phi(z_T|z_{<T}, x) = \log q_\phi(z_0|x) - \sum_{t=1}^{T} \log\det\left|\frac{\partial f_t}{\partial z_{t-1}}\right|, \tag{19}$$

where $\frac{\partial f_t}{\partial z_{t-1}}$ is the Jacobian matrix of $f_t(z_{t-1})$ computed by repeatedly applying the well-known change of variable theorem to the multi-variate random variable z_T defined as

$$z_T = f_T(f_{T-1}(\ldots(f_1(z_0))\ldots)). \tag{20}$$

An interesting aspect concerning Normalizing Flows is that, under suitable assumptions, they are provably universal, in the sense defined in [32]. As already mentioned, the first successfully integration of Normalizing Flows in VAEs was

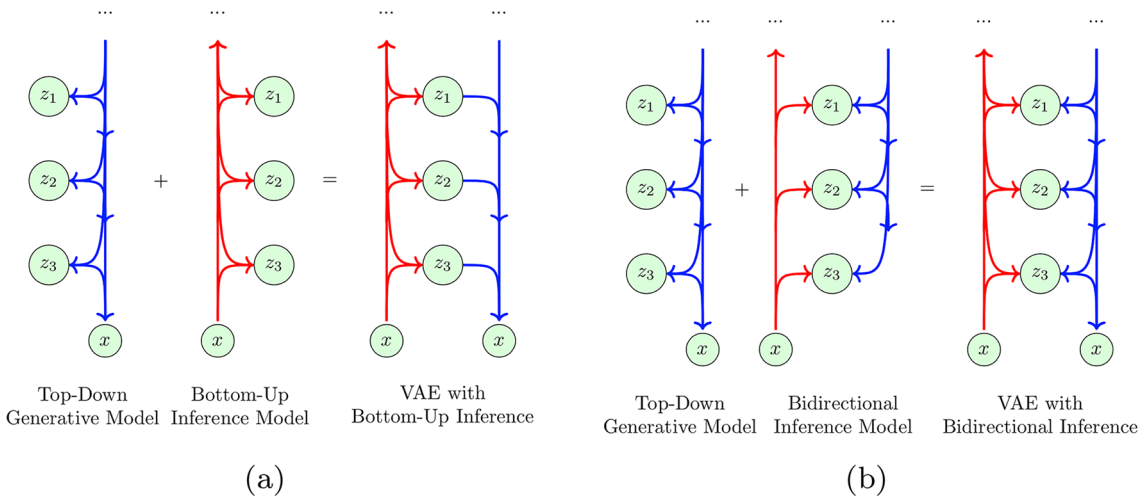

Fig. 5 Diagrams that schematically represents Hierarchical VAE in two different configurations: bottom-up inference (**a**) and bidirectional inference (**b**)

by Kingma et al. [37], where they introduced Inverse Autoregressive Flows (IAF). The idea was to define $f_t(z_{t-1})$ as a simple affine function of the form:

$$z_t = f_t(z_{t-1}) = \mu_t + \sigma_t \odot z_{t-1} \quad \forall t = 1, \dots, T, \tag{21}$$

where $z_0 \sim q_\phi(z_0|x) = G(\mu_\phi(x), \sigma^2_\phi(x))$.

Figure 4 schematically represents the unrolling of Eq. (21).

We highlight that the IAF introduces a natural order in the latent variables. For this reason, we will refer to this kind of models as Hierarchical Variational Autoencoder (HVAE). In this paradigm, we will refer to each z_t as a group of latent variables, and we will collect the set of all groups in a vector $z = (z_0, \dots, z_T)$ where the variables are written in the order defined above.

If we distinguish between the encoder (inference) network $q_\phi(z|x)$ and the decoder (generative) network, we need to choose if the ordering of latent variables is the same in the two parts of the network (*bottom-up inference*), or if it is reversed (*bidirectional inference*) as shown in Fig. 5.

As it is clear from Fig. 5, in bottom-up inference the image $x \in \mathbb{R}^d$ is encoded to $z = (z_1, \dots, z_T)$ independently from the prior $p(z) = \prod_{t=1}^{T} p(z_t|z_{<t})$; in the generative phase the image is reconstructed by taking z_T as the final output of the encoder, and then sampling each z_t, $t = T-1, \dots, 0$ from the prior distribution independently from $q_\phi(z_t|x)$ (i.e. the encoder and decoder are independent from each other). We underline that this fact makes the bottom-up inference training unstable.

Conversely, in bidirectional inference, the process of generating latent variables is shared between the two parts of the network, which makes the training easier, although the design of the network is a bit more difficult.

Since the results of vanilla IAF are not competitive with the state of the art, we will not use them in our future analysis (see the original paper for more information), whereas we will focus our experimental results on two powerful variants of IAF, making use of bidirectional inference and residual blocks to generate high-quality images.

Experimental Setting

For each variant of Variational Autoencoder discussed in the previous sections, we provide an original implementation in TensorFlow 2, and a set of detailed benchmarks on traditional datasets, such as MNIST, Cifar10 and CelebA. The specific architectures which have been tested are described in the following. All models have been compared using standard metrics, assessing both their energy consumption through the number of floating point operations (see "Green AI and FLOPS"), and their performance via the so-called Frechèt Inception Distance [42], briefly discussed in "Frechèt Inception Distance". Numerical results are given in "Numerical Results", along with examples of reconstructed and generated images.

Green AI and FLOPS

The paradigm of Green AI [51] is meant to raise the attention on the computational efficiency of neural models, encouraging a reduction in the amount of resources required for their

training and deployment. This concept is not so trivial as it seems; in fact, most of traditional AI research (referred to as Red AI, in this con) targets accuracy rather than efficiency, exploiting massive computational power, and resulting in rapidly escalating costs; this trend is not sustainable for various reasons, it is environmentally unfriendly [40], socially not inclusive and inefficient.

The computation of floating point operations (FLOPS) was advocated in [51] as a measure of the efficiency of models; the main advantages of this measure are that it is hardware independent and has a direct (even if not precise) correlation with the running time of the model [13]. There are also known problems related to FLOPs, mostly related to the fact that memory access time can be a more dominant factor in real implementations (see the "Trap of FLOPs" discussion in [33]).

So, while we shall adopt FLOPS for our comparison, we shall also investigate performance through more traditional tools, like Tensorboard, also to gain confidence on the reliability of FLOPs-based assessments.

Frechèt Inception Distance

To test the quality of the generator, we should compare the probability distribution of generated vs. real images. Unfortunately, the dimension of the feature space is typically too large to allow a direct, significant comparison; moreover, in the case of images, adjacent pixels are highly correlated, reducing their statistical relevance. The main idea behind the so-called Frechèt Inception Distance (FID) [42] is to use, instead of raw data, their internal representations generated by some third party, agnostic network. In the case of FID, the Inception v3 network [54] trained on Imagenet is used to this purpose; Inception is usually preferred over other models due to the limited amount of preprocessing performed on input images (images are rescaled in the interval [– 1,1], sample wise). The activations that are traditionally used are those relative to the last pooling layer, with a dimension of 2048 features.

Given the activations a_1 and a_2, relative to real and generated images, and called $\mu_i, i = 1, 2$ and $C_i, i = 1, 2$ their empirical mean and covariance matrix, respectively, the Frèchet Distance between a_1 and a_2 is defined as

$$FID(a_1, a_2) = ||\mu_1 - \mu_2||^2 + Tr(C_1 + C_2 - 2(C_1 * C_2)^{\frac{1}{2}}), \tag{22}$$

where we indicate with Tr the trace of a matrix.

A problem of FID is that it is extremely sensible to a number of different factors listed below.

1. the weights of the Inception network. The checkpoint that is traditionally used is the `inception_v3_2016_08_28/inception_v3.ckpt` downloaded from TF-Slim's pre-trained models, also available through Tensorflow-HUB. These weights were originally obtained by training on the ILSVRC-2012-CLS dataset for image classification ("Imagenet").
2. The dimension of the datasets of real/generated images to be compared. Traditionally, sets of 10 K images are compared; typically, the FID score is inversely proportional to this dimension.
3. The dimension of input images fed to Inception. Inception may work with images of arbitrary size (larger than 75×75); however, the "canonical" input dimension is 299×299. Again, varying the dimension may result in very different scores.
4. The resizing algorithm. Images must be resized to bring them to the expected input dimension of 299×299; as observed in [2], the FID score is quite sensible to the algorithm used to this aim, and in particular to the employed modality: nearest neighbour, bilinear interpolation, cubic interpolation, ... The default, is usually bilinear interpolation, being a good compromise between efficiency and quality.

Unfortunately, articles in the literature are not always fully transparent on the previous points, that may explain some discrepancies and the difficulty one frequently faces in replicating results.

All our experiments have been conducted with "defaults" values: the standard Inception checkpoint `inception_v3_2016_08_28/inception_v3.ckpt`, 10000 images of dimension 299×299, rescaled by means of bilinear interpolation.

Let us finally observe that, in the case of VAE, it is customary to measure both the FID score for reconstructed images (FID_{rec}) and the FID score for generated images (FID_{gen}). The former one is usually reputed to be a lower bound for the latter, no matter what help we may provide to the generator during the sampling phase.

Architecture Overview

In this section, we provide detailed descriptions of the several different neural networks architectures we have been dealing with, each one inspired by a different article. For each of them, different possible configurations have been investigated, varying the number and dimension of layers, as well as the learning objectives. Moreover, since some of the techniques considered are not dependent from the encoder/

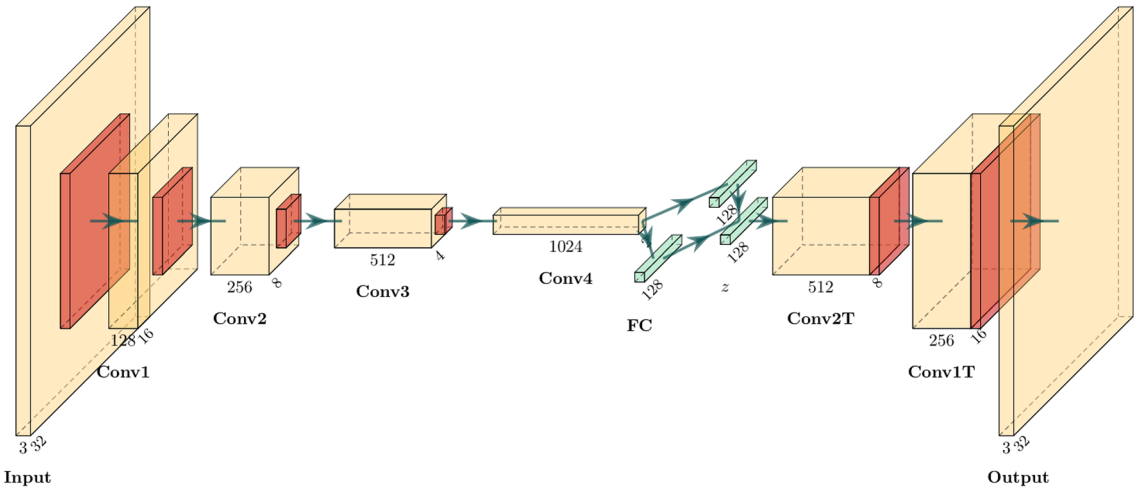

Fig. 6 Graphical representation of the Vanilla VAE architecture. The yellow, orange and green boxes represent convolutional, downsampling and dense layers, respectively

decoder structure, we also tested a mix of different architectures, hyperparameters configurations, and optimization objectives.

Vanilla Convolutional VAE

In our first experiment, we followed the same structure of [26], which is a simple CNN architecture where we doubled the number of channels for each Convolution, and we downsampled the spatial dimension by 2 (see Fig. 6).

The encoder is structured as follows. In the first layer, the input image of dimension $(d, d, 3)$ (where $d = 32$ and $d = 64$ in CIFAR10 and CelebA, respectively) was passed through a convolutional layer with 128 channels and stride equals 2, to obtain 128 images of dimension $(d/2, d/2)$. This operation is repeated for 256, 512, 1024 channels. The result is flattened and passed through two Dense layers to obtain the mean and the variance of the latent variables.

The decoder has the same structure of the encoder, with Transposed convolutions and Upsample layers.

Each convolutional filter has kernel size 4 and ReLU activation function, except for the last layer of the decoder, where we used a sigmoid activation to ensure that the output is in the range [0, 1].

Resnet-Like

The Resnet-like architecture was adopted in [16]. The main difference of this network with respect to the Vanilla VAE is that, before downsampling, the input is processed by a so called *Scale Block*, that is just a sequence of Residual Blocks. In turn, a Residual Block is an alternated sequence of BatchNormalization and Convolutional layers (with unit stride), intertwined with residual connections. The number of Scale Blocks at each scale of the image pyramid, the number of Residual Blocks inside each Scale Block, and the number of convolutions inside each Residual Block are user configurable hyperparameters.

In the encoder, at the end of the last Scale Block, a global average level extracts spatial agnostic features. These are first passed through a so called Dense Block (similar to a Residual Block but with dense layers instead of convolutions), and finally used to synthesize mean and variance for latent variables.

The decoder first maps the internal encoding z to a small map of dimension $4 \times 4 \times base\_dim$ via a dense layer suitably reshaped. This is then up-sampled to the final expected dimension, inserting Scale Blocks at each scale.

Two-Stage VAE

To check in what extent the Two-Stage VAE improve the generation ability of a Variational Autoencoder, we tried to fit a second stage to every model we tested, following the architecture described in the following and graphically represented in Fig. 7.

The encoder in the second stage model in composed of a couple of Dense layers of dimension 1536 and ReLU activation function, followed by a concatenation with the Input of the model and then by another Dense layer to obtain the latent representation u with the same dimension of z, following what is described in "Two-Stage VAE". The decoder has exactly the same structure of the encoder.

As already described, we used the *cosine similarity* as the reconstruction part of the ELBO objective function.

We observed that, to improve the quality of the generation, the second stage should be trained for a large number of epochs.

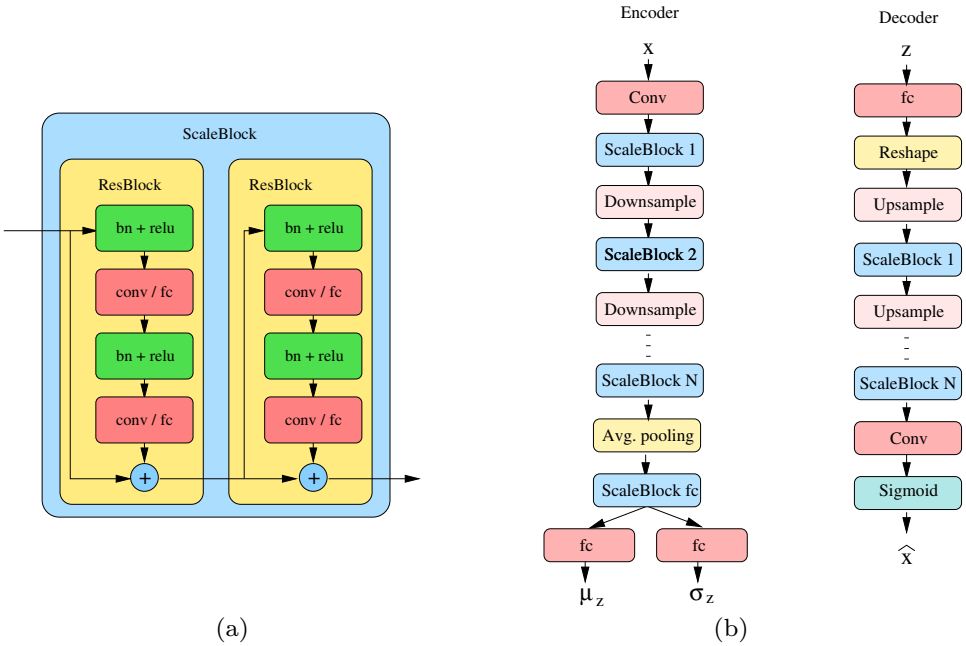

Fig. 7 **a** Scale block. The Scale Block is used to process features at a given scale; it is a sequence of Residual Blocks intertwined with residual connections. A Residual Block is an alternation of batchnormalization layers, rectified linear units and convolutions. **b** The input is progressively downsampled via convolutions with stride 2, intermixed by Scale Blocks; at a given scale, a global average pooling layer extract features that are further processed via dense layers to compute mean and variance for latent variables. The decoder is essentially symmetric

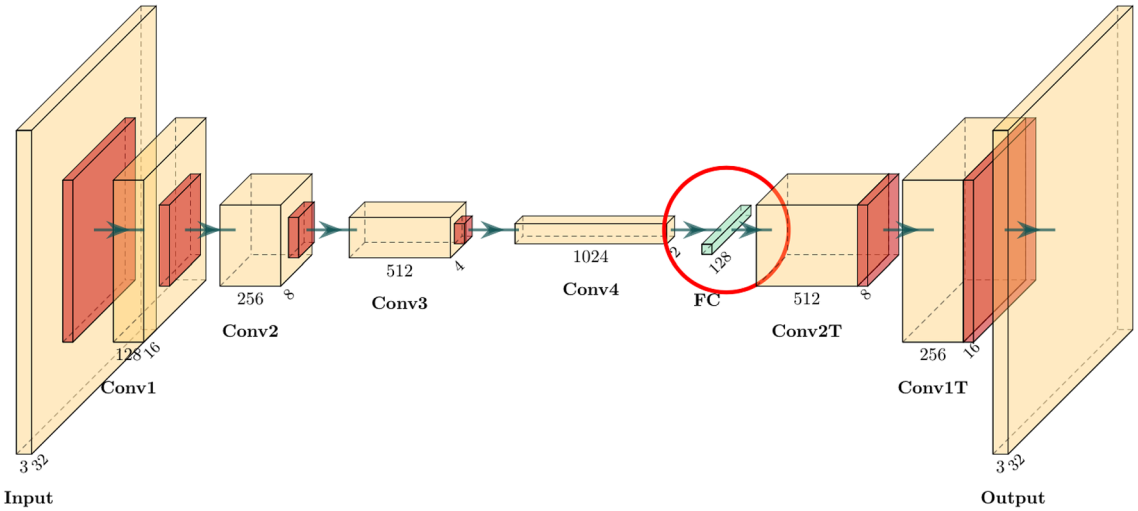

Fig. 8 Graphical representation of the RAE architecture. The yellow, orange and green boxes represent convolutional, downsampling and dense layers, respectively. The red circle underlines the sole architectural difference between our implementation of VanillaVAE and RAE, i.e. the fact that in the latter, the latent space is composed by a single Dense layer that directly encodes to z, while in VanillaVAE the encoding is performed by a couple of Dense layers that represents the mean and the variance of a Gaussian distribution

Convolutional RAE

In our implementation of RAE, we followed exactly the same structure as in Convolutional Vanilla VAE, with the sole difference that, in RAEs, the latents space is composed of just one fully connected layer representing the variable z (see Fig. 8).

In our tests, we only compared L_2 and GP regularization, with parameter λ heuristically computed to achieve the best performance.

NVAE

The model is organized in a bottom-up inference network and a top-down generative network (see Fig. 9). Each one of two networks is composed by a hierarchy of modules at different *scales*. Each scale is composed by *groups* of sequential (residual) blocks.

During generation, each module computes from the current input h_l a prior $p(z_l|h_l)$ (h_l depends from $z_{<l}$): after sampling from this prior, the result is combined in some way

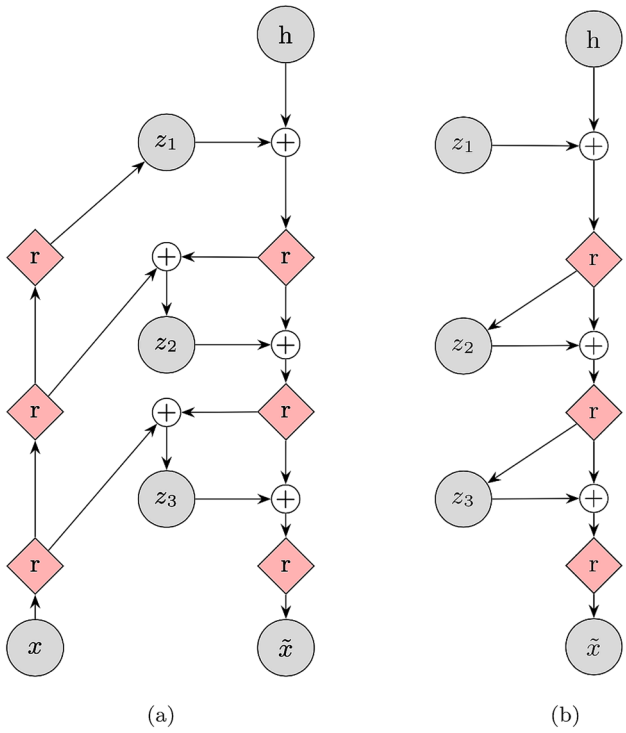

Fig. 9 The whole NVAE architecture (**a**) and a focus on its decoder (**b**)

Table 1 Summary of the hyperparameters used in the training of NVAE on the datasets used in this paper

Hyperparameter	MNIST	Cifar10	CelebA
Input size	28×28	32×32	64×64
Epochs	400	400	90
Batch size	200	32	16
Normalizing flows	0	2	2
Scales	2	1	3
Groups per scale	5,10	30	5,10,20
Spatial dims of z per scale	$4^2, 8^2$	16^2	$8^2, 16^2, 32^2$
Channel dims of z	20	20	20
Initial channels in Enc.	32	128	64
Residual cells per group	1	2	2
GPUs	2	8	8
Total train time (h)	21	55	92

with the current input h_l, the two informations are processed together and passed to the next module.

During inference, we extract the latent representation at stage l by synthesizing a mean and a standard deviation for $q(z_l|x, h_l)$: since this information depends from h_l, we expect to provide additional information, not already available by previous latent encodings. Moreover, the computation of h_l, is done by the top-down network, that is hence a sub-network of the inference network. During training, both networks are trained together.

Each network has a hierarchical organization at different *scales*. Each scale is composed by *groups* of Blocks.

Both Encoder Blocks (EB) and Decoder Blocks (DB) have similar architectures, and are essentially composed by an alternated sequence of BatchNormalization and Convolutional layers, separated by non linear activation layers, and intermixed with residual connections (so, very similar to the Scale Block discussed in the previous section). A few technical novelties are, however, introduced by the authors:

- the recent *Swish* activation function $f(u) = \frac{u}{1+e^{-u}}$ [45] is used instead of Relu, Elu, or other more traditional choices;
- a Squeeze-and-Excitation (SE) layer [34] is added at the end of each block;
- a moderate use of depthwise separable convolutions [15] is deployed to reduce the number of parameters of the network.

Table 1 gives a summary of hyperparameters used in training NVAE on the datasets addressed in this article, borrowed from [58]. D^2 indicates a latent variable with the spatial dimensions of $D \times D$. As an example, the MNIST model consists of two scales: in the first one, we have five groups of $4 \times 4 \times 20$-dimensional latent variables: in the second one, we have ten groups of $8 \times 8 \times 20$-dimensional variables.

The figures of merit in Table 1 help to understand the key novelty of NVAE, that is in the massive usage of space located latent variables. Consider for instance the case of Cifar10. The original input of dimension $32 \times 32 \times 3$ is first transformed to dimension $16 \times 16 \times 128$ and then, *without any further downscaling*, processed though a long sequence of residual cells (30×2). At each iteration, a huge number of latent variables ($16 \times 16 \times 20$) is extracted and used for the internal representation, which hence has a dimension widely larger than the input. Due to this fact, as it is also observed by the authors in the appendix, it is not surprising that most of the variables will collapse during training.

Working with such a huge number of latent variables introduces a lot of issues; in particular, it becomes crucial to balance the KL-component of variables belonging to different groups. To this aim, the authors introduce additional balancing coefficients γ_l to ensure that a similar amount of information is encoded in each group (see [58], Appendix A):

$$D_{KL}(q(z|x)||p(z)) = \sum_{l=1}^{L} \gamma_l \, \mathbb{E}_{q(z<l|x)}[D_{KL}(q(z_l|x, z_{<l})||p(z_l|z_{<l}))].$$

Table 2 Summary of the results obtained with the networks in the first column on Cifar10

Model	Params	FLOPs (M)	MSE	REC	GEN1	GEN2	GMM
CNN-b128-l128	31,034,755	2397	2.8	27.6	96.2	96.8	89.0
L2-RAE-b128-l128	30,510,339	2395	**1.2**	**9.9**	108.1	88.4	78.2
GP-RAE-b128-l128	30,510,339	2395	**1.2**	10.6	118.0	97.6	**76.4**
Resnet-s4-b48-l128	16,179,363	1431	1.5	37.2	110.0	93.9	96.3
Resnet-s4-b48-l100	16,064,619	1430	1.6	37.5	102.9	88.4	91.4
Resnet-s4-b64-l64	27,766,275	2539	1.7	36.5	94.2	**78.8**	85.1
HFVAE-s4-z4-l48	27,139,755	**1163**	1.8	45.9	93.3	90.8	90.0
HFVAE-s4-z12-l64	48,113,051	2085	1.3	33.3	**89.0**	85.7	86.4
NVAE-z10-b100-l4	**8,305,521**	4478	3.2	62.6	96.1	87.4	91.4

The balancing coefficient γ_l is kept proportional to the KL term for that group, in such a way to encourage the model to revive the latent variables in that group when KL is low, and to clip them if KL is too high. Additionally, γ_l is also proportional to the size of each group, to encourage the use of variables at lower scales.

NVAE architectures have a relatively small number of parameters, due to the extensive use of convolutions and depthwise separable convolutions; however, they require a massive amount of memory, and huge computational power: for the configuration used for Cifar10, composed by 30 groups at scale 16, we estimated a number of flops for the inference phase larger then 100 G.

Due to this reasons, we experimented a sensibly lighter architecture, just composed of five groups, with a few additional convolutions to augment the receptive fields of the spatially located latent variables. The good news is that the network, even in this severely crippled form, still seems to learn; however, results are really modest and below the performances of different networks with comparable complexity.

HFVAE

As we already remarked, the main novelty of NVAE is in the massive exploitation of a huge number of spatially located latent variables. To test the relevance of this architectural decision, we also tested a different variant of the hereditary architecture of Fig. 9, where we drop the spatial dimension for latent variables, using instead a Featurewise Linear Modulation Layer [44] to modulate channels according to the internal representation. In addition, the first approximation h_1 is directly produced from the latent variable set z_0 through a dense transformation. The general idea is that at lower scales we decide the content of the resulting image, while stylistic details at different resolutions (usually captured in channels correlations [25]) are added at higher scales. We call this variant HFVAE (Hereditary Film VAE); a similar architecture has been investigated in [9].

Numerical Results

In this section, we provide quantitative evaluations for some configurations of the models previously discussed. The precise configurations (layers, channels, blocks, etc.) are discussed below.

The datasets used for the comparison are CIFAR10 and CelebA: in a GreenAI perspective, we are reluctant to address more complex datasets, at higher resolutions, that would require additional computational resources and additional costs. On CelebA, we just evaluated a subset of particularly interesting models.

For each model, we provide the following figures:

- **Params**: the number of parameters;
- **FLOPs**: an estimation of number of FLOPS (see "Green AI and FLOPS" for more details);
- **MSE**: the mean reconstruction error $\times 10^3$;
- **REC**: the FID value computed over *reconstructed* images;
- **GEN1**: the FID value computed over images generated by a first VAE;
- **GEN2**: the FID value computed taking advantage of a second VAE;
- **GMM**: the FID value computed by superimposing a GMM of ten Gaussians[2] on the latent space. In the case of hierarchical models, the GMM is computed on the innermost set of latent variables.

The following list provides a legends for the names of models used in the following tables:

- **CNN-by-lz:** Vanilla VAE with CNN architecture, basedim of y channels and latent space of dimension z.
- **L2-RAE-by-lz:** L_2-RAE with CNN architecture, basedim of y channels and latent space of dimension z.

[2] Augmenting the number of Gaussians does not sensibly improve generation.

Table 3 Summary of the results obtained with the networks in the first column on CelebA

Model	Params	FLOPs (M)	MSE	REC	GEN1	GEN2	GMM
CNN-b128-l64	40,668,419	4104	3.2	48.4	66.9	56.2	55.2
L2-RAE-b128-l64	27,359,043	4102	3.3	39.8	230.2	61.7	45.1
GP-RAE-b128-l64	27,359,043	4102	3.2	41.2	230.6	65.3	47.0
Resnet-s4-b32-l64	**19,330,627**	**2924**	2.8	51.4	66.0	54.9	57.4
Resnet-s4-b48-l64	38,996,003	6452	**2.5**	46.8	61.7	50.8	54.5
Resnet-s3-b64-l64	21,370,179	5949	2.6	**39.2**	**59.3**	**44.9**	**45.8**

- **GP-RAE-by-lz:** GP-RAE with CNN architecture, basedim of y channels and latent space of dimension z.
- **Resnet-sx-by-lz:** Resnet-like model, with x ScaleBlocks, a basedim of y channels, and a latent space of dimension z.
- **HFVAE-sx-by-lz:** HFVAE with x scales, ScaleBlocks, a basedim of y channels, and a latent space of dimension z at hereditary scales; the base latent dimension z_0 is 64.
- **NVAE-zx-by-lz:** NVAE with x latent variables channels, a basedim of y and z latent groups of the same scale.

Quality Evaluation

Here we draw a few conclusions about the design of Variational Autoencoders deriving from the previous investigation (Tables 2 and 3) and our past experience with VAEs.

- The decoder is more important than the encoder. For instance, in the ResNet architecture latent features are extracted via a GlobalAverage layer, obtaining robust features, less prone to overfitting.
- Working with a larger number of latent variables improves reconstruction, but this does not eventually implies better generation. This is, e.g. evident comparing the two Resnet-like architectures with latent spaces of dimension 128 and 100.
- Fitting a GMM over the latent space [26] is a cheap technique (it just takes a few minutes) that invariably improves generation, both in terms of perceptual quality and FID score. This fact also confirms the mismatch between the prior and the aggregated posterior discussed in "Aggregate Posterior vs. Expected Prior Mismatch".
- The second stage technique [16] typically requires some tuning to properly works, but when it does it usually outperforms the GMM approach. Tuning may involve the loss function (we used cosine similarity in this work), the architecture of the second VAE, and the learning rate (more generally, the optimizer's parameters).
- Hierarchical architectures are complex systems, difficult to understand and to work with (monitoring/calibrating training is a really complex task). We cannot express an opinion about NVAE, since its complexity trespasses our modest computational facilities, but simpler architectures like those described in [28] or [22], in our experience, do not sensibly improve generation over a well-constructed traditional VAE.
- The loss of variance for generated images [6] (see "Blurriness") is confirmed in all models, and it almost coincides with the mean squared error for reconstruction.

A qualitative comparison between the different models in generating images can also be done by looking at the images in the Appendix.

Energetic Evaluation

Before comparing the energetic footprint of the different models, let us briefly discuss the notion of FLOPs as a measure of computational efficiency. FLOPs have been computed by a library for Tensorflow Keras under development at the University of Bologna, and inspired by similar works for PyTorch (see, e.g. https://github.com/sovrasov/flops-counter.pytorch). FLOPs only provide an abstract, machine-independent notion of complexity; typically, only the most expensive layers are taken into account (those with superlinear complexity with respect to the size of inputs). The way this quantity will result in an actual execution time and energetic consumption does, however, largely depend

Table 4 Average forward time (in seconds) over the Cifar10 Test Set (10 k images) for different networks, hardware and batchsize (bs). The two times entries in each cell refer to different machines: the first is a Laptop with an NVIDIA Quadro T2000 graphics card and a Core i7-9850H CPU; the second is a workstation with an Asus GeForce DUAL-GTX1060-O6G graphic card and a intel Core i7-7700K CPU

Network	bs100	bs10	bs1
Resnet-s4-b48-l128	3.0 ± 0.2	6.0 ± 0.2	33.3 ± 0.4
	4.9 ± 0.2	8.8 ± 0.2	49.5 ± 0.5
Resnet-s4-b48-l100	2.86 ± 0.1	5.9 ± 0.2	32.9 ± 0.4
	4.8 ± 0.2	8.7 ± 0.2	49.1 ± 0.5
Resnet-s3-b64-l64	4.4 ± 0.2	$9. \pm 0.3$	47.8 ± 0.4
	7.2 ± 0.2	13.5 ± 0.2	78.6 ± 0.5

Table 5 Average Forward Time (in seconds) over the Cifar10 Test Set (10 k images) for different architectures and different batchsize (bs); times refer to a workstation equipped with an Asus GeForceDUAL-GTX1060-O6G GPU and a Intel Core i7-7700K CPU

Model	Params	FLOPS (M)	bs100	bs10	bs1
CNN-b128-l128	31,034,755	2397	5.8 ± 0.1	9.0 ± 0.1	54.1 ± 0.4
L2-RAE-b128-l128	30,510,339	2395	11.6 ± 0.2	13.9 ± 0.2	57.3 ± 0.5
GP-RAE-b128-l128	30,510,339	2395	12.5 ± 0.2	14.1 ± 0.2	56.3 ± 0.5
Resnet-s4-b48-l128	16,179,363	1431	4.9 ± 0.2	8.8 ± 0.2	49.5 ± 0.4
Resnet-s4-b48-l100	16,064,619	1430	4.8 ± 0.2	8.7 ± 0.2	49.1 ± 0.4
Resnet-s4-b64-l64	27,766,275	2539	7.2 ± 0.2	13.5 ± 0.2	78.6 ± 0.5
HFVAE-s4-z4-l48	27,139,755	1163	13.1 ± 0.2	29.2 ± 0.3	207.0 ± 1.1
HFVAE-s4-z12-l64	48,113,051	2085	21.3 ± 0.3	48.7 ± 0.4	325.2 ± 1.6

from the underlying hardware, and the available parallelism. As an example, in Table 4 we compare the execution time for a forward step over the test set of Cifar10 (10 K) for a couple of hardware configurations. The first one is a Laptop with an NVIDIA Quadro T2000 graphics card and a CPU Core i7-9850H; the second one is a workstation with an Asus GeForce DUAL-GTX1060-O6G graphic card and a CPU intel Core i7-7700K. Observe the strong dependency from the batch size that is not surprising but worth to be recalled (see [12] for a critical analysis of the performance of Neural Networks architectures). Of course, as soon as we move the computation on a cloud, execution times are practically unpredictable.

Unfortunately, as we shall see, even for a *given* computational device, the relation between FLOPs and execution time is quite aleatory.

Following the traditional paradigm, we compare performances on the forward pass. This is already a questionable point; on one side, it is true that this reflects the final usage of the network when it is deployed in practical applications; on the other side, it is plausible to believe that training still takes a prevalent part of the lifetime of any neural network. Restricting the investigation to forward time means not taking into account some expensive techniques of the training of modern systems, such as regularization components. For example, it is possible to notice that in Table 5, L_2-RAE and GP-RAE have exactly the same number of FLOPs, since in terms of forward execution they are equal. However, we highlight that the training of GP-RAE is almost ten times slower than the training of L_2-RAE. This is a consequence of the fact that the regularization term of GP-RAE involves the computation of the decoder gradient with respect to the latent variables, which is an expensive operation not required in L_2-RAE. Consequently, even if the two models have more or less the same performance in terms of generation quality, L_2-RAE should be preferred, since its training is cheaper. Moreover, taking into account only the FLOPs of the model, the actual convergence speed of systems is neglected.

The results of the energetic evaluation on the forward pass are given in Table 5; inference times have been computed over a workstation with an Asus GeForceDUAL-GTX1060-O6G graphic card and a intel Core i7-7700K CPU. The same results have also been expressed in graphical form in Fig. 10, relatively to a batch size of dimension 1. In the plot, we omit L2-RAE and GP-RAE, since their architectures and figures are essentially analogous to the basic CNN; similarly for some Resnet architectures.

As it is clear from these results, there is no precise correlation between FLOPS and execution time.

As an example, from Table 5, we see that HFVAE requires a computation time 4–6 times higher than the others in the face of the lowest number of FLOPS. One of the possible reasons for this behaviour is, in our opinion, the fact that the total computation time also includes memory access time in addition to FLOPS. As observed by several authors (see, e.g. [33]), memory access time is a crucial factor in real implementations, as densely packed data might be read faster than a few numbers of scattered values. For instance, while depthwise convolutions greatly reduce the number of parameters and FLOPS, they require a more fragmented memory access, harder to

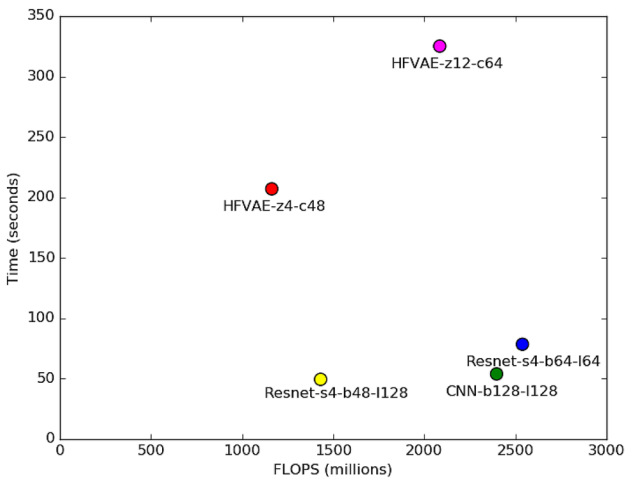

Fig. 10 FLOPs versus execution time. From the plot, we can evince the little relation between the two figures but, possibly, at a magnitude level

be implemented efficiently. In future works, we intend to deeper investigate other causes for the absence of correlation between FLOPS and time.

Conclusions

In this article, we presented a critical survey of recent variants of Variational Autoencoders, referring them to the several problems that still hinder this generative paradigm. In view of the emerging Green AI paradigm [51], we also focused the attention on the computational cost of the different architectures. The main conclusions of our investigation are given in "Quality Evaluation", and we shall not try to summarize them here; we just observe that, while we strongly support the Green AI vision, we must eventually find better metrics than FLOPs to compare the energetic performance of neural networks, or more realistic way to compute them.

The constant improvement in generative sampling during the last few years is very promising for the future of this field, suggesting that state-of-the-art generative performance can be achieved or possibly even improved by carefully designed VAE architectures.

At the same time, the quest for scaling models to higher resolution and larger images, and the introduction of additional, and usually computationally expensive, regularization techniques, is a scaring and dangerous perspective from the point of view of Green AI.

From this point of view, our experience with NVAE is explicative and quite frustrating. The architecture is interesting, and it should eventually deserve a deeper investigation; unfortunately, it seems to require computational facilities far beyond those at our disposal.

A Examples of Generated Images

A.1 Cifar10

See Figs. 11, 12, 13, 14, 15 and 16.

Fig. 11 Examples of Cifar-like images generated by Vanilla VAE

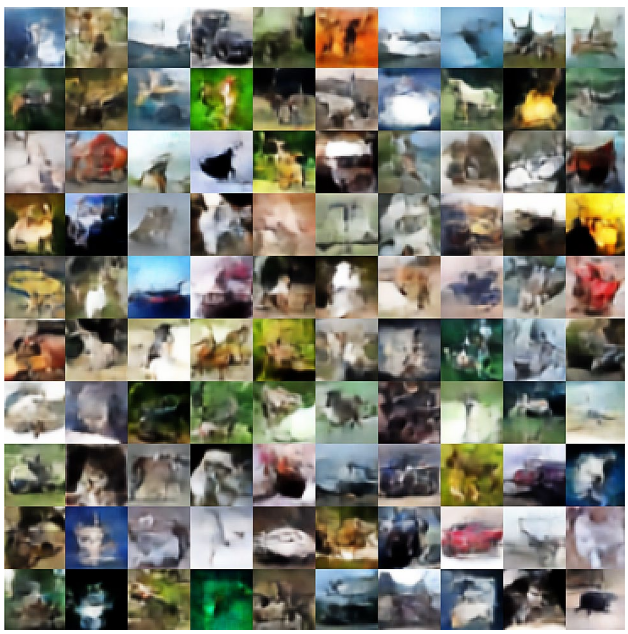

Fig. 12 Examples of Cifar-like images generated by Resnet

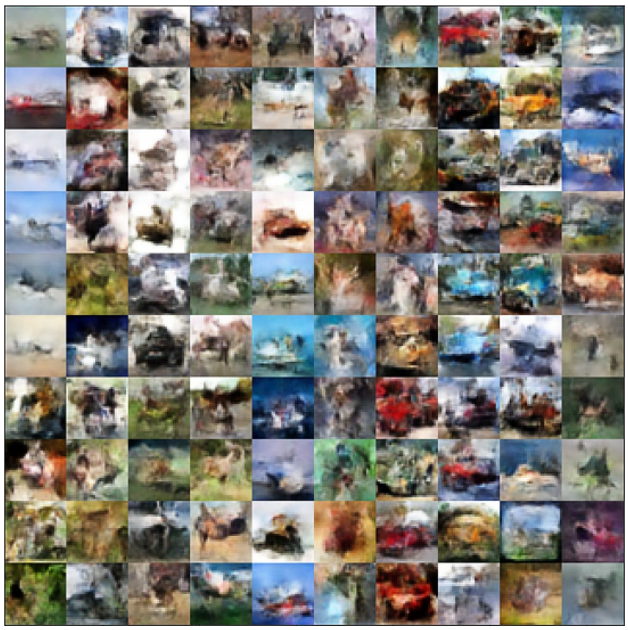

Fig. 13 Examples of Cifar-like images generated by L_2-RAE

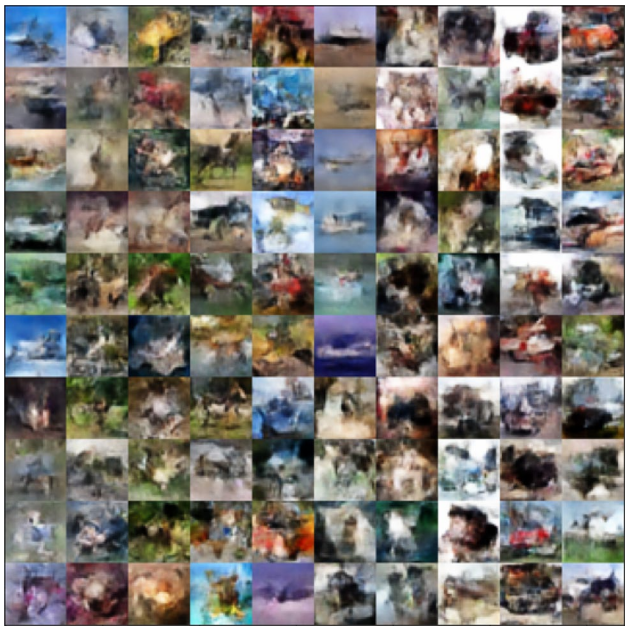

Fig. 14 Examples of Cifar-like images generated by GP-RAE

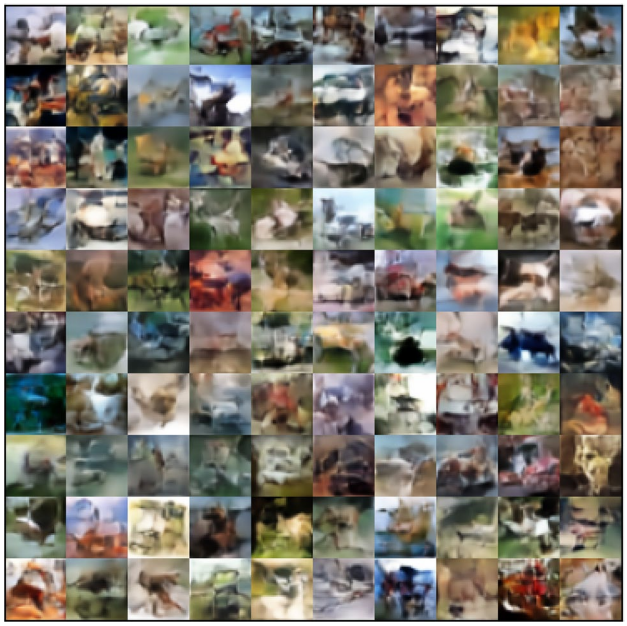

Fig. 15 Examples of Cifar-like images generated by HFVAE

Fig. 16 Examples of Cifar-like images generated by (a severely simplified version of) NVAE

A.2 CelebA

See Figs. 17, 18, 19 and 20.

Fig. 17 Examples of CelebA faces generated by Vanilla VAE

Fig. 18 Examples of CelebA faces generated by Resnet-s3-b64-l64

Fig. 19 Examples of CelebA faces generated by L_2-RAE

Fig. 20 Examples of CelebA faces generated by GP-RAE

Acknowledgements We would like to thank Federico Brunelllo who, under the supervision of Prof. Asperti, is developing the library for the computation of flops used in this article.

Funding Open access funding provided by Alma Mater Studiorum - Università di Bologna within the CRUI-CARE Agreement.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Alemi AA, Poole B, Fischer I, Dillon JV, Saurous RA, Murphy K. Fixing a broken ELBO. In: Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10–15, 2018; 2018. pages 159–68.
2. Andrea Asperti and Matteo Trentin. Balancing reconstruction error and kullback-leibler divergence in variational autoencoders. IEEE Access. 2020;8:199440–8.
3. Asperti A. Variational autoencoders and the variable collapse phenomenon. Sens Transducers. 2019;234(6):1–8.
4. Asperti A. About generative aspects of variational autoencoders. In: Machine Learning, Optimization, and Data Science—5th International Conference, LOD 2019, Siena, Italy, September 10–13, 2019, Proceedings; 2019. pages 71–82.
5. Asperti A. Sparsity in variational autoencoders. In: Proceedings of the First International Conference on Advances in Signal Processing and Artificial Intelligence, ASPAI, Barcelona, Spain, 20–22 March 2019; 2019.
6. Asperti A. Variance loss in variational autoencoders. In: Machine Learning, Optimization, and Data Science—6th International Conference, LOD 2020, Siena, Italy, September 10–13, 2020, July 19–23, 2020, Proceedings, volume To appear of Lecture Notes in Computer Science. Springer; 2020.
7. Bauer M, Mnih A. Resampled priors for variational autoencoders. CoRR. 2018. arxiv:abs/1810.11428.
8. Bowman SR, Vilnis L, Vinyals O, Dai AM, Józefowicz R, Bengio S. Generating sentences from a continuous space. CoRR. 2015. arxiv:abs/1511.06349.
9. Branca D. Generazione di attributi facciali mediante feature-wise linear modulation. Master's thesis, University of Bologna. 2020.
10. Burda Y, Grosse RB, Salakhutdinov R. Importance weighted autoencoders. CoRR. 2015. arxiv:abs/1509.00519.
11. Burgess CP, Higgins I, Pal A, Matthey L, Watters N, Desjardins G, Lerchner A. Understanding disentangling in beta-vae. 2018.
12. Canziani A, Culurciello E, Paszke A. Evaluation of neural network architectures for embedded systems. In: IEEE International Symposium on Circuits and Systems, ISCAS 2017, Baltimore, MD, USA, May 28–31, 2017, pages 1–4. IEEE; 2017.
13. Canziani A, Paszke A, Culurciello E. An analysis of deep neural network models for practical applications. 2017.
14. Chen W, Liu W, Cai Z, Xu H, Pei D. VAEPP: variational autoencoder with a pull-back prior. In: Yang H, Pasupa K, Leung AC-S, Kwok JT, Chan JH, King I, editors. Neural Information Processing—27th International Conference, ICONIP 2020, Bangkok, Thailand, November 23-27, 2020, Proceedings, Part III, volume 12534 of Lecture Notes in Computer Science, pages 366–79. Springer; 2020.
15. Chollet F. Xception: Deep learning with depthwise separable convolutions. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2017. pages 1800–7.
16. Dai B, Wipf DP. Diagnosing and enhancing vae models. In: Seventh International Conference on Learning Representations (ICLR 2019), May 6–9, New Orleans; 2019.
17. Davidson TR, Falorsi L, De Cao N, Kipf T, Tomczak JM. Hyperspherical variational auto-encoders. In: Amir G, Ricardo S, editors. Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6–10, 2018, pages 856–65. AUAI Press; 2018.
18. Devlin J, Chang M-W, Lee K, Toutanova K. BERT: pre-training of deep bidirectional transformers for language understanding. In: Jill B, Christy D, Thamar S, editors. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2–7, 2019, Volume 1 (Long and Short Papers), pages 4171–86. Association for Computational Linguistics; 2019.
19. Dilokthanakul N, Mediano PAM, Garnelo M, Lee MCH, Salimbeni H, Arulkumaran K, Shanahan M. Deep unsupervised clustering with gaussian mixture variational autoencoders. CoRR. 2016. arxiv:abs/1611.02648.
20. Doersch C. Tutorial on variational autoencoders. CoRR. 2016. arxiv:abs/1606.05908.
21. Dosovitskiy A, Brox T. Generating images with perceptual similarity metrics based on deep networks. In: Lee DD, Sugiyama M, von Luxburg U, Guyon I, and Garnett R, editors. Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5–10, 2016, Barcelona, Spain; 2016. pages 658–66.
22. Eslami SMA, Danilo JR, Frederic B, Fabio V, Morcos AS, Garnelo M, Ruderman A, Rusu AA, Danihelka I, Gregor K, Reichert DP, Buesing L, Weber T, Vinyals O, Rosenbaum D, Rabinowitz N, King H, Hillier C, Botvinick M, Wierstra D, Kavukcuoglu K, Hassabis D. Neural scene representation and rendering. 2018; 360(6394):1204–10.
23. Esmaeili B, Wu H, Jain S, Bozkurt A, Siddharth N, Paige B, Brooks DH, Dy JG, van de Meent J-W. Structured disentangled representations. In: Kamalika C and Masashi S, editors. The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16–18 April 2019, Naha, Okinawa, Japan, volume 89 of Proceedings of Machine Learning Research, pages 2525–34. PMLR; 2019.
24. Fisher NI, Lewis T, Embleton BJJ. Statistical analysis of spherical data. Cambridge: Cambridge University Press; 1987.
25. Gatys LA, Ecker AS, Bethge M. Image style transfer using convolutional neural networks. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27–30, 2016, pages 2414–23. IEEE Computer Society; 2016.
26. Ghosh P, Sajjadi MSM, Vergari A, Black MJ, Schölkopf B. From variational to deterministic autoencoders. In: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26–30, 2020. OpenReview.net; 2020.
27. Goodfellow IJ. NIPS 2016 tutorial: Generative adversarial networks. CoRR. 2017. arxiv:abs/1701.00160.

28. Gregor K, Danihelka I, Graves A, Rezende DJ, Wierstra D. DRAW: a recurrent neural network for image generation. In: Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6–11 July 2015, volume 37 of JMLR Workshop and Conference Proceedings, pages 1462–71. JMLR.org; 2015.
29. Higgins I, Matthey L, Pal A, Burgess C, Glorot X, Botvinick M, Mohamed S, Lerchner A. beta-vae: Learning basic visual concepts with a constrained variational framework. 2017.
30. Hoffman MD, Johnson MJ. Elbo surgery: yet another way to carve up the variational evidence lower bound. In: Workshop in Advances in Approximate Bayesian Inference, NIPS, volume 1; 2016.
31. Hou X, Shen L, Sun K, Qiu G. Deep feature consistent variational autoencoder. CoRR. 2016. arxiv:abs/1610.00291.
32. Jaini P, Kobyzev I, Yu Y, Brubaker M. Tails of lipschitz triangular flows. In: Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13–18 July 2020, Virtual Event, volume 119 of Proceedings of Machine Learning Research, pages 4673–81. PMLR; 2020.
33. Jeon Y, Kim J. Constructing fast network through deconstruction of convolution. In: Bengio S, Wallach HM, Larochelle H, Grauman K, Cesa-Bianchi N, and Garnett R, editors. Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada; 2018. pages 5955–65.
34. Jie H, Shen L, Albanie S, Sun G, Enhua W. Squeeze-and-excitation networks. IEEE Trans Pattern Anal Mach Intell. 2020;42(8):2011–23.
35. Junyoung C, Kyle K, Laurent D, Kratarth G, Courville Aaron C, Yoshua B. A recurrent latent variable model for sequential data. In: Cortes C, Lawrence ND, Lee DD, Sugiyama M, Garnett R, editors. Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015(December), pp. 7–12, . Montreal. Canada: Quebec; 2015. p. 2980–8.
36. Karras T, Aila T, Laine S, Lehtinen J. Progressive growing of gans for improved quality, stability, and variation. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30–May 3, 2018, Conference Track Proceedings. OpenReview.net; 2018.
37. Kingma DP, Salimans T, Józefowicz R, Chen X, Sutskever I, Welling M. Improving variational autoencoders with inverse autoregressive flow. In: Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5–10, 2016, Barcelona, Spain; 2016. pages 4736–44.
38. Kingma DP, Welling M. Auto-encoding variational bayes. In: 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14–16, 2014, Conference Track Proceedings; 2014.
39. Kumar A, Poole B, Murphy K. Regularized autoencoders via relaxed injective probability flow. In: Silvia C and Roberto C, editors. The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26–28 August 2020, Online [Palermo, Sicily, Italy], volume 108 of Proceedings of Machine Learning Research, pages 4292–4301. PMLR; 2020.
40. Lacoste A, Luccioni A, Schmidt V, Dandres T. Quantifying the carbon emissions of machine learning. 2019.
41. Larsen Anders BL, Sønderby SK, Larochelle H, Winther O. Autoencoding beyond pixels using a learned similarity metric. In: Maria-Florina Balcan and Kilian Q, edition. Weinberger, editors, Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19–24, 2016, volume 48 of JMLR Workshop and Conference Proceedings, pages 1558–66. JMLR.org; 2016.
42. Martin H, Hubert R, Thomas U, Bernhard N, Sepp H. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4–9 December 2017. CA, USA: Long Beach; 2017. p. 6629–40.
43. Mathieu E, Rainforth T, Siddharth N, Teh YW. Disentangling disentanglement in variational autoencoders. In: Kamalika C and Ruslan S, editors. Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA, volume 97 of Proceedings of Machine Learning Research, pages 4402–12. PMLR; 2019.
44. Perez E, Strub F, de Vries H, Dumoulin V, Courville AC. Film: visual reasoning with a general conditioning layer. In: McIlraith SA and Weinberger KQ, editors. Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2–7, 2018, pages 3942–51. AAAI Press; 2018.
45. Ramachandran P, Zoph B, Le QV. Searching for activation functions. 2017.
46. Razavi A, van den Oord A, Poole B, Vinyals O. Preventing posterior collapse with delta-vaes. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6–9, 2019. OpenReview.net. 2019.
47. Rezende DJ, Mohamed S. Variational inference with normalizing flows. In: Bach FR and Blei DM, editors. Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015, volume 37 of JMLR Workshop and Conference Proceedings, pages 1530–8. JMLR.org; 2015.
48. Rezende DJ, Mohamed S, Wierstra D. Stochastic backpropagation and approximate inference in deep generative models. In: Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21–26 June 2014, volume 32 of JMLR Workshop and Conference Proceedings, pages 1278–86. JMLR.org; 2014.
49. Rosca M, Lakshminarayanan B, Mohamed S. Distribution matching in variational inference. 2018.
50. Rui G, Xingsong H, Jie Q, Jiaxin C, Li L, Fan Z, Zhao Z, Ling S. Zero-vae-gan: generating unseen features for generalized and transductive zero-shot learning. IEEE Trans Image Process. 2020;29:3665–80.
51. Schwartz R, Dodge J, Smith NA, Etzioni O. Green AI. Commun ACM. 2020;63(12):54–63.
52. Shao H, Xiao Z, Yao S, Zhang A, Liu S, Abdelzaher T. Control-vae: tuning, analytical properties, and performance analysis. 2020.
53. Spindler A, Geach JE, and Smith MJ. Astrovader: astronomical variational deep embedder for unsupervised morphological classification of galaxies and synthetic image generation, 2020.
54. Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z. Rethinking the inception architecture for computer vision. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27–30, 2016, pages 2818–26. IEEE Computer Society, 2016.
55. Tolstikhin IO, Bousquet O, Gelly S, Schölkopf B. Wasserstein auto-encoders. CoRR, abs/1711.01558, 2017.
56. Tomczak JM, Welling M. VAE with a vampprior. In: International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain; 2018. pages 1214–23.
57. Trippe B and Turner R. Overpruning in variational bayesian neural networks. In: Advances in Approximate Bayesian Inference workshop at NIPS 2017, 2018.
58. Vahdat A and Kautz J. NVAE: a deep hierarchical variational autoencoder. CoRR, abs/2007.03898, 2020.

59. van den Oord A, Vinyals O, and Kavukcuoglu K. Neural discrete representation learning. In: Guyon I, von Luxburg U, Bengio S, Wallach HM, Fergus R, Vishwanathan SVN, and Garnett R, editors. Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4–9, 2017, Long Beach, CA, USA; 2017. pages 6306–15.
60. Wei R, Mahmood A. Recent advances in variational autoencoders with representation learning for biomedical informatics: a survey. IEEE Access. 2021;9:4939–56.
61. Wei R, Garcia C, El-Sayed A, Peterson V, Mahmood A. Variations in variational autoencoders—a comparative evaluation. IEEE Access. 2020;8:153651–70.
62. Xie J, Girshick RB, and Farhadi A. Unsupervised deep embedding for clustering analysis. In: Balcan MF and Weinberger KQ, editors. Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19–24, 2016, volume 48 of JMLR Workshop and Conference Proceedings, pages 478–87. JMLR.org, 2016.
63. Yeung S, Kannan A, Dauphin Y, and Fei-Fei L. Tackling overpruning in variational autoencoders. CoRR, abs/1706.03643, 2017.
64. Yongqin X, Saurabh S, Bernt S, and Zeynep A. F-VAEGAN-D2: a feature generating framework for any-shot learning. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16–20, 2019, pages 10275–84. Computer Vision Foundation / IEEE, 2019.