Machine Learning (2025) 114:151
https://doi.org/10.1007/s10994-025-06784-3

A comparison of latent space modeling techniques in a plain-vanilla autoencoder setting

Fabian Kächele[1] · **Maximilian Coblenz**[2] · **Oliver Grothe**[1]

Received: 17 September 2024 / Revised: 27 January 2025 / Accepted: 15 April 2025 /
Published online: 23 May 2025

Abstract
By sampling from the latent space of an autoencoder and decoding the latent space samples to the original data space, any autoencoder can be turned into a generative model. For this to work, it is necessary to model the latent space with a distribution from which samples can be obtained. Several simple possibilities such as kernel density estimates or a Gaussian distribution and more sophisticated ones such as Gaussian mixture models, copula models, and normalization flows can be thought of and have been tried recently. In a plain-vanilla autoencoder setting, this study aims to discuss, assess, and compare various techniques that can be used to capture the latent space so that an autoencoder can become a generative model. Furthermore, we provide insights into further aspects of these methods, such as targeted sampling or synthesizing new data with specific features.

Keywords Autoencoder · Latent space · Copula · Generative methods

1 Introduction

Generating realistic sample points of various data formats has been of growing interest in recent years. This is why models dedicated to other tasks, such as autoencoders (AEs), which try to find a low-dimensional representation of the high-dimensional input data (Baldi & Hornik, 1989; Goodfellow et al., 2016; Kramer, 1991), have been

Editor: Henry Gouk.

✉ Maximilian Coblenz
maximilian.coblenz@hwg-lu.de

Fabian Kächele
fabian.kaechele@kit.edu

Oliver Grothe
oliver.grothe@kit.edu

[1] Institute for Operations Research, Karlsruhe Institute of Technology (KIT), Kaiserstr. 12, 76131 Karlsruhe, Baden-Württemberg, Germany

[2] Department of Services and Consulting, Ludwigshafen University of Business and Society, Ernst-Boehe-Str. 4, 67059 Ludwigshafen am Rhein, Rhineland-Palatinate, Germany

repurposed to generate new data, as well. We call the low-dimensional representation of the data in the AE the latent space in the following. To turn an AE into a generative model, the latent space distribution is modeled, samples are drawn, and thereupon new data points in the original space are constructed with the decoder. This is where this paper comes in, by investigating how distributions of a latent space can best be modeled statistically. Other AE models, such as Variational Autoencoders (VAEs), which have evolved for variational inference by optimizing for a Gaussian distribution in the latent space (Kingma & Welling, 2014), can also be used to generate new data. Furthermore, adversarial autoencoders utilize elements of autoencoders and generative adversarial nets (GANs) (Goodfellow et al., 2014), where a discriminant model penalizes the distance of the encoded data from a prior (Gaussian) distribution (Makhzani et al., 2016). However, such strong (and simplifying) distributional assumptions as in the VAE or adversarial autoencoder can have a negative impact on performance, leading to a rich literature coping with the challenge of reducing the gap between approximate and true posterior distributions (e.g., Cremer et al. (2018), Gregor et al. (2015), Kingma et al. (2016), Marino et al. (2018), Rezende and Mohamed (2015), Takahashi et al. (2019), Tomczak and Welling (2018)). In this paper, we discuss more flexible approaches for modeling the latent space without imposing restrictions on the underlying distribution.

Recently, Tagasovska et al. (2019) presented the Vine Copula Autoencoder. Their approach comprises two building blocks, an AE and a vine copula which models the dependence structure in latent space. By that, they were able to create realistic, new images with samples from the fitted vine copula model in the latent space. In this work, we want to elaborate on this idea and compare various methods to model the latent space of an AE to turn it into a generative model. Here, we restrict ourselves to image data, because this opens up the possibility of visual inspection. To this end, we analyze, among others, the usage of Gaussian mixture models (GMM) as done by Ghosh et al. (2020), the vine copula approach by Tagasovska et al. (2019), and simple multivariate Kernel Density Estimates along the lines of Saha et al. (2022). Additionally, we introduce a new, non-parametric copula approach based on the empirical beta copula.

In this study, we do not aim to beat the latest state-of-the-art generative models but want to shed light on different modeling techniques in the latent space and their characteristics. Therefore, we use a rather straightforward "plain-vanilla" AE setting, however, the techniques treated may be applied in more sophisticated AE models, as well. We believe that such an analysis in a simple setting is essential for understanding the effects from different sampling methods, which may then be applied in more advanced generative models. We also check whether the methods may be a simple alternative to more complex models, such as normalization flows (Rezende & Mohamed, 2015) or diffusion models (see, e.g., Rombach et al. (2022), Vahdat et al. (2021)). More specifically, we use the well-known Real NVP (Dinh et al., 2017) as an example from these more sophisticated machine learning models in the latent space but do not elaborate on these in detail as this is not the focus of the paper. Note that in contrast to other methods (e.g., as proposed by Oring et al. (2021), Berthelot et al. (2019), van den Oord et al. (2017)), the investigated overall approach does not restrict or change the training of the AE in any form.

All models considered in this work are constructed in three steps, visualized in Fig. 1. First, an AE, consisting of an encoder f and a decoder g, is trained to find a low-dimensional representation of the data X. Second, the data in the latent space Y is used to learn the best fitting representation Y' of it. This is where the examined models differ from each other by using different methods to model the latent space. Finally, we sample from the

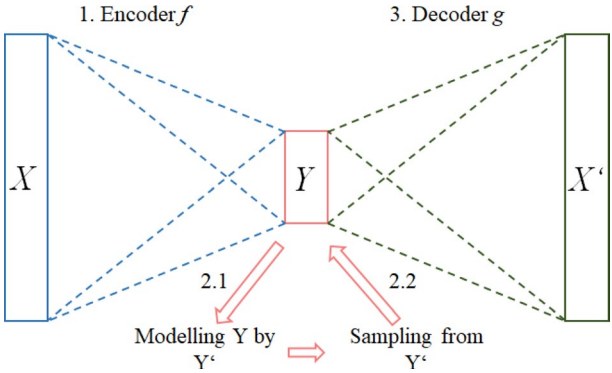

Fig. 1 Function scheme of simple generative autoencoders. 1. An encoder f encodes the data X to a low dimensional representation Y. 2.1 Y is modeled by Y'. 2.2 Generate new synthetic samples of the latent space by sampling from Y'. 3. Decode the new samples with the decoder g

learned representation of the latent space and feed the samples into the decoder part of the AE, creating new synthetic data samples.

We want to emphasize that for the models we consider, no prior is needed, nor the optimization approach is changed, i.e., the latent space is modeled after the training of the AE post-hoc. Thus, the presented approach can be transferred to other, more sophisticated, state-of-the-art AEs, as hinted in Ghosh et al. (2020). For example, it can be used to model the latent space of a VAE, because when training is stopped early the latent space might not be perfectly normally distributed, and thus, modeling it, e.g., with a Gaussian mixture model might yield better data generation results (Ghosh et al., 2020). The general approach used here is particularly applicable, if sampling from the AE model directly is costly, since some of the presented latent space modeling methods have low training and sampling times (cf. Sect. 2.3). Furthermore, modeling the latent space after training opens up the possibility for targeted sampling—even if the classes are only known after the training—or recombination of features (cf. Sect. 3.2.6).

Generative models are a vivid part of the machine learning literature. For example, new GAN developments (Hudson & Zitnick, 2021; Karras et al., 2021; Lee et al., 2021; Varshney et al., 2021), developments in the field of AEs (Larsen et al., 2016; Shen et al., 2020; Yoon et al., 2021; Zhang et al., 2020), or developments in VAEs (Havtorn et al., 2021; Masrani et al., 2019; Sohn et al., 2015; Xu et al., 2019) are emerging. The general idea of creating new data by sampling in the latent space of a generative model has already been used by, e.g., Tagasovska et al. (2019), Dai and Wipf (2019), Brehmer and Cranmer (2020), Ghosh et al. (2020), or Saha et al. (2022), but to the best of our knowledge, no analysis and comparison of such methods have been made so far. Closely related, more and more researchers specifically address the latent space of generative models in their work (Fajtl et al., 2020; Hofert et al., 2021; Mishne et al., 2019; Moor et al., 2020; Oring et al., 2021). There, especially hierarchical methods as suggested by Maaløe et al. (2019) seem to be promising. Further, AEs based on the Wasserstein Distance lately achieved excellent results by changing the regularization term of a VAE and using or learning a Gaussian mixture prior (Mondal et al., 2021; Tolstikhin et al., 2019), analogously to our use of Gaussian Mixtures fitting the latent space distribution. Also, we thank a referee for pointing us towards works in compression, see, e.g., Qin et al. (2023b, 2023c, 2023a, 2024a, 2024b).

This work does not propose a new 'black-box algorithm' for generating data but analyzes challenges and possible answers on how AEs can be turned into generative models by using well-understood tools of data modeling. One of our main findings is, that it is hard to find a

trade-off between out-of-bound sampling and creating new images. We conclude that besides a pure numerical perspective and looking at new random samples of a generative model with a latent space, the resulting image of the nearest neighbor in the latent space from the training data should be inspected. We demonstrate in our experiments that copula-based approaches may be promising alternatives to traditional modeling methods since they allow for the recombination of marginal distributions from one class with the dependence structure of another class. This leads to new possibilities in synthesizing images and targeted sampling. Note that in this work, we restrict ourselves to image data. However, we hope that researchers in other data domains can benefit from our discussion as well as receive new impulses for their research.

The remainder of the paper is structured as follows. Section 2 introduces various methods for modeling the latent space. Besides traditional approaches, copula-based methods are introduced. Section 3 describes the implementation, evaluation, and results of the experiments carried out. In Sect. 4, we discuss the results and conclude. Computer code for replication is available at the following url: https://github.com/FabianKaechele/SamplingFromAutoencoders.

2 Modeling the latent space

In this section, we introduce and reflect on different methods to model the latent space in an AE (Step 2 in Fig. 1). All methods aim to fit the low-dimensional data Y as best as possible to be able to create new sample points in the latent space, which leads to new realistic images after passing the decoder. Pseudocode of the overall sampling approach is given in Algorithm 1.

We first recap more 'traditional' statistical tools, followed by copulas as an intuitive and flexible tool for modeling high-dimensional data in Sects. 2.1 and 2.2, respectively. We briefly explain how each approach can be used to model data in the latent space and how to obtain samples thereof. Finally, in Sect. 2.3 we compare the methods theoretically in terms of what kinds of distribution can be modeled with each. Note that we do not introduce our benchmark models, namely the standard plain vanilla VAE and the Real NVP, and refer to the original papers instead (Dinh et al., 2017; Kingma & Welling, 2014).

Algorithm 1 Overall sampling approach

Algorithm 1: Overall sampling approach

Input: Autoencoder with Encoder f and Decoder g
begin
  Compute latent space Y by passing training samples through encoder f
  for *each method* **do**
    Model the latent space by fitting the respective method
    Create new samples from the latent space Y' by drawing (randomly) from the fitted method
  for *each element* Y'_i *in* Y' **do**
    Decode Y'_i by passing it through the decoder g
Output: New samples X'

2.1 Traditional modeling methods

We classify the multivariate Gaussian distribution, a Kernel Density Estimation (KDE), and a Gaussian Mixture Model (GMM) as traditional modeling methods and give a rather short treatment of each below. They are well known and can be studied in various statistics textbooks such as Hastie et al. (2001) or Bishop (2006).

2.1.1 Multivariate Gaussian

The probably simplest method is to assume the data in the latent space follow a multivariate Gaussian distribution. Thus, we estimate the covariance matrix $\hat{\Sigma}$ and mean vector $\hat{\mu}$ of the data Y. In the second step, we draw samples thereof and pass them through the decoder to generate new images. Note that this is similar to the sampling procedure in a VAE (Kingma & Welling, 2014), but without forcing the latent space to be Gaussian during training.

2.1.2 Gaussian mixture model

The Gaussian Mixture Model aims to model the density $p(y)$ of the latent space by mixing M multivariate Gaussian distributions. Thus, the GMM has the form

$$p(y) = \sum_{m=1}^{M} \alpha_m \phi(y; \mu_m, \Sigma_m) \tag{1}$$

where α_m denote the mixing parameters and ϕ denotes the density of the multivariate normal distribution with mean vector μ_m and covariance matrix Σ_m. The model is usually fit by maximum likelihood using the expectation-maximization algorithm. By combining several Gaussian distributions, it is more flexible than estimating only one Gaussian distribution as above. Note that modeling the latent space with a GMM is the approach introduced in Ghosh et al. (2020).

A GMM can be seen as some kind of kernel method (Hastie et al., 2001), having a rather wide kernel. In the extreme case, i.e., where m equals the number of points the density is estimated on, a Gaussian distribution with zero variance is centered over each point. Kernel density estimation is introduced in the following.

2.1.3 Kernel density estimation

Kernel Density Estimation is a well-known non-parametric tool for density estimation. Put simply, a KDE places a density around each data point. The total resulting estimated density is constructed by

$$p(y) = \frac{1}{Nh} \sum_{i=1}^{N} K_h(y, y_i) \tag{2}$$

with N being the total number of data points, h the bandwidth, and K the used kernel. Note that the choice of bandwidth and kernel can affect the resulting estimated density. Using KDE to model the latent space is akin to the approach of Saha et al. (2022). However, they train the AE and KDE in one pass, whereas here, we first train the AE and fit the KDE post-hoc in the latent space.

KDE can be performed in univariate data as well as in multivariate data. In this work, we rely on the most commonly used kernel, the Gaussian Kernel, and a bandwidth fitted via Silverman's rule of thumb (Silverman, 1986) for the univariate KDEs, while we use a grid search with 10-fold cross-validation in the multivariate case. We use KDE in multiple manners throughout this work. First, we use a multivariate KDE to model the density of the data in the latent space itself. In the case of a Gaussian kernel, it can be written by

$$p(y) = \frac{1}{N\sqrt{|\mathbf{H}|2\pi}} \sum_{i=1}^{N} e^{-1/2(y-y_i)^T \mathbf{H}^{-1}(y-y_i)} \tag{3}$$

where $\mathbf{H}$ represents the covariance matrix of the kernel, i.e., the matrix of bandwidths. Second, we ignore the dependence structure between variables and estimate the univariate densities of each dimension in the latent space by a KDE for each marginal distribution. In this way, we are able to find out whether explicitly modeling the dependence structure is necessary or not. We call that approach the Independent modeling approach, also denoted short by Independent in the following. Last, we use univariate KDEs for modeling the marginal distributions of each dimension in the latent space and use them in the copula models described next.

2.2 Copula based models

Besides the traditional modeling methods introduced above, we apply copula based models. In the following, we first introduce copulas as a tool for high-dimensional data, which allows us to model the latent space in our application. Then, we focus on two copula-based methods to model the latent space of the AE: the vine copula and the empirical beta copula in Sects. 2.2.1 and 2.2.2, respectively. For detailed introductions to copulas, we refer the reader to Nelsen (2006), Joe (2014), Durante and Sempi (2015).

Copulas have been subject to an increasing interest in the Machine Learning community over the last decades, see, e.g., Dimitriev and Zhou (2021), Janke et al. (2021), Messoudi et al. (2021), Ma et al. (2021), Letizia and Tonello (2020), Liu (2019), Kulkarni et al. (2018), Tran et al. (2015). In a nutshell, copula theory enables us to decompose any d-variate distribution function into d marginal univariate distributions and their joint dependence structure, given by the copula function. Thus, copulas "couple" multiple univariate distributions into one joint multivariate distribution. More formally, a d-variate copula $C : [0,1]^d \xrightarrow{[} 0,1]$ is a d-dimensional joint distribution function whose margins are uniformly distributed on the unit interval. Decomposing and coupling distributions with copulas is formalized in Theorem 1 going back to Sklar (1959).

Theorem 1 (Sklar (1959)) Consider a d-dimensional vector of random variables $\boldsymbol{Y} = (Y_1, \dots, Y_d)$ with joint distribution function $F_{\boldsymbol{Y}}(y) = P(Y_1 \le y_1, \dots, Y_d \le y_d)$. The marginal distribution functions F_j are defined by $F_j(y_j) = P(Y_j \le y_j)$ for $y_j \in \mathbb{R}$, $j = 1, \dots, d$. Then, there exists a copula C, such that

$$F_{\boldsymbol{Y}}(y_1, \dots, y_d) = C(F_1(y_1), \dots, F_d(y_d))$$

for $(y_1, \dots, y_d) \in \mathbb{R}^d$. Vice versa, using any copula $\tilde{C}$, it follows that $\tilde{F}_{\boldsymbol{Y}}(y_1, \dots, y_d) := \tilde{C}(F_1(y_1), \dots, F_d(y_d))$ is a proper multivariate distribution function.

This allows us to construct multivariate distributions with the same dependence structure but different margins or multivariate distributions with the same margins but different couplings/pairings, i.e., dependence structures. The simplest estimator of a copula is given by the empirical copula. Based on an independent sample $Y_i = (Y_{i,1}, \dots, Y_{i,d})$ for $i = 1, \dots, n$ of $\boldsymbol{Y}$ it can be calculated directly on the ranks by

$$\hat{C}(\mathbf{u}) = \frac{1}{n} \sum_{i=1}^{n} \prod_{j=1}^{d} \mathbf{1}\left\{ \frac{r_{i,j}^{(n)}}{n} \leq u_j \right\} \tag{4}$$

with $\mathbf{u} = (u_1, \dots, u_d) \in [0,1]^d$ and $r_{i,j}^{(n)}$ denoting the rank of each $y_{i,j}$ within $(y_{1,j}, \dots, y_{n,j})$, i.e.,

$$r_{i,j}^{(n)} = \sum_{k=1}^{n} \mathbf{1}\{y_{k,j} \leq y_{i,j}\}. \tag{5}$$

Note that $\mathbf{u} = (u_1, \dots, u_d)$ represents a quantile level, hence a scaled rank. Simultaneously, the univariate margins can be estimated using a KDE so that the full distribution of the latent space is governed for. Note that it is not possible to draw new samples from the empirical copula directly as no random process is involved. In our applications, the latent space is typically equipped with dimensions $\gg 2$. Although a variety of two-dimensional copula models exist, the amount of multivariate (parametric) copula models is somewhat limited. We present two solutions to this problem in the following, namely vine copulas and the empirical beta copula.

2.2.1 Vine Copula

Vine copulas decompose the multivariate density as a cascade of bivariate building blocks organized in a hierarchical structure. This decomposition is not unique, and it influences the estimation procedure of the model. Here, we follow Tagasovska et al. (2019) and use regular-vine (r-vine) models (Czado, 2019; Joe, 2014) to model the 10, 20 and 100 dimensional latent space of the AEs in our experiments later. An r-vine is built of a sequence of linked trees $T_i = (V_i, E_i)$, with nodes V_i and edges E_i for $i = 1, \dots, d-1$ and follows distinct construction rules, which we briefly introduce in the following. An in-depth introduction to vine copulas can be found, e.g., in Czado (2019).

A $d-$dimensional vine tree structure $V = (T_1, \dots, T_{d-1})$ is a sequence of $T-1$ trees if e (see Czado (2019)):

1. Each tree $T_i = (N_i, E_i)$ is connected, i.e., for all nodes $a, b \in T_i, i = 1, \dots, d-1$, there exists a path $n_1, \dots, n_k \subset N_j$ with $a = n_1, b = n_k$.
2. T_1 is a tree with node set $N_1 = \{1, \dots, d\}$ and edge set E_1.
3. For $i \geq 2$, T_i is a tree with node set $N_i = E_{i-1}$ and edge set E_i.
4. For $i = 2, \dots, d-1$ and $\{a, b\} \in E_i$ it must hold that $|a \cap b| = 1$.

An example of a five-dimensional vine tree structure is given in Fig. 2.

In an r-vine, the d-dimensional copula density can be written as the product of its bivariate building blocks:

$$c(u_1, \dots, u_d) = \prod_{i=1}^{d-1} \prod_{e \in E_i} c_{a_e b_e; D_e}(u_{a_e|D_e}, u_{b_e|D_e}) \tag{6}$$

with conditioning set D_e, conditional probabilities, e.g., $u_{a_e|D_e} = \mathbb{P}(U_{a_e} \leq u_{a_e}|D_e)$, and conditional copula density $c_{a_e b_e; D_e}$ given D_e. The conditioning set D_e includes all variables conditioned on at the respective position in the vine structure (see Fig. 2). For each resulting two-dimensional copula of conditional variables, any parametric or non-parametric copula model (as done by Tagasovska et al. (2019)) can be chosen. However, the construction and estimation of vine copulas is rather complicated. Hence, assuming independence for seemingly unimportant building blocks, so-called truncation, is regularly applied. Because of this, truncated vine copula models do not capture the complete dependence structure of the data, and their usage is not underpinned by asymptotic theory. We refer to Czado (2019), Czado and Nagler (2022), Aas (2016) for reviews of vine copula models.

2.2.2 Empirical Beta Copula

The empirical beta copula (Segers et al., 2017) avoids the problem of choosing a single, parametric multivariate copula model due to its non-parametric nature. Further, and in contrast to the presented vine copula approach, it offers an easy way to model the full, non-truncated multivariate distribution based on the univariate ranks of the joint distribution and, thus, seems to be a reasonable choice to model the latent space. The empirical beta copula is closely related to the empirical copula (see Formula 5). It is solely based on the ranks $r_{i,j}^{(n)}$ of the original data Y and can be interpreted as a continuous counterpart of the empirical copula. It is defined by

$$C^{\beta} = \frac{1}{n} \sum_{i=1}^{n} \prod_{j=1}^{d} F_{n, r_{i,j}^{(n)}}(u_j) \tag{7}$$

for $\mathbf{u} = (u_1, \dots, u_d) \in [0,1]^d$, where

$$F_{n, r_{i,j}^{(n)}}(u_j) = P(U_{(r_{i,j}^{(n)})} \leq u_j) \tag{8}$$

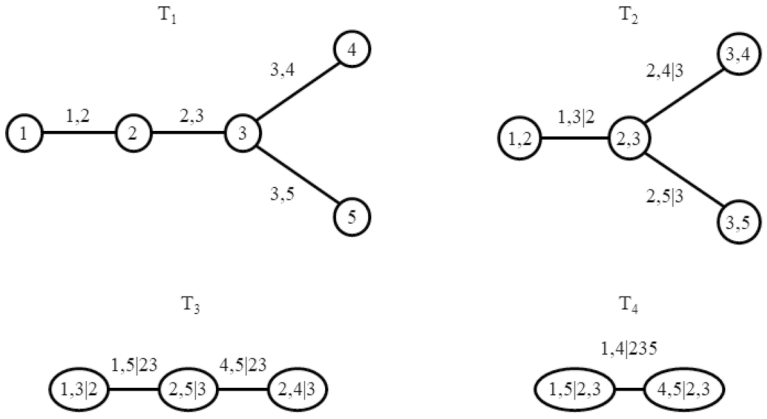

Fig. 2 Example of a vine copula tree structure $T_1 - T_4$ for five dimensions

$$= \sum_{p=r_{i,j}^{(n)}}^{n} \binom{n}{p} u_j^p (1-u_j)^{(n-p)} \tag{9}$$

is the cumulative distribution function of a beta distribution $\mathbb{B}(r_{i,j}^{(n)}, r_{i,j}^{(n)} + n - 1)$. As $r_{i,j}$ is the rank of the i^{th} element in dimension j, $U_{(r_{i,j}^{(n)})}$ represents the $r_{i,j}^{\text{th}}$ order statistic of n i.i.d. uniformly distributed random variables on [0, 1]. For example, if the rank of the i^{th} element in dimension j is 5, $U_{(r_{i,j}^{(n)})} = U_{(5^{(n)})}$ denotes the 5^{th} order statistic on n i.i.d. uniformly distributed random variables.

The intuition behind the empirical beta copula is as follows: Recall that the marginal distributions of a copula are uniformly distributed on [0, 1] and, hence, the k^{th} smallest value of scaled ranks $r_{i,j}^{(n)}/n$ corresponds to the k^{th} order statistic $U_{(k)}$. Such order statistics are known to follow a beta distribution $\mathbb{B}(k, k+n-1)$ (David & Nagaraja, 2003). Consequently, the mathematical idea of the empirical beta copula is to replace each indicator function of the empirical copula with the cumulative distribution function of the corresponding rank $r_{i,j}^{(n)}$.

Since the Beta distribution captures the expected variation of ranks and the empirical copula is based on ranks, we argue that the empirical beta copula can be seen as a naturally extended version of the empirical copula. It can be shown that the empirical beta copula has the same large-sample distribution as the empirical copula and, thus, converges to the true copula (Segers et al., 2017). However, the empirical beta copula performs better for small samples. Segers et al. (2017) demonstrate that the empirical beta copula outperforms the empirical copula both in terms of bias and variance.

Synthetic samples Y' in the latent space are created by reversing the modeling path. First, random samples from the copula model $\mathbf{u} = (u_1, \ldots, u_d)$ are drawn. Then, the copula samples are transformed back to the natural scale of the data by the inverse probability integral transform of the marginal distributions, i.e., $Y'_j = \hat{F}_j(u_j)$, where $\hat{F}_j$ is the estimated marginal distribution and u_j the jth element of the copula sample for $j \in \{1, \ldots, d\}$. Algorithm 2 summarizes the procedure.

Algorithm 2 Sampling from Empirical Beta Copula

Algorithm 2: Sampling from Empirical Beta Copula

Input: Sample $Y \subset \mathbb{R}^{n \times d}$, new sample size m
begin
  Compute rank matrix $R^{n \times d}$ out of Y
  Estimate marginals of Y with KDE, $\hat{f}_1(y_1), \ldots, \hat{f}_d(y_d)$.
  for $i \leq m$ **do**
    Draw random from $I \in [1, \ldots, n]$
    for $j \leq d$ **do**
      Draw $u_{I,j} \sim \mathbb{B}(R_{Ij}, n + 1 - R_{Ij})$
    Set $u_i = (u_{I1}, \ldots, u_{Id})$
    Rescale margins by $Y_i = \hat{F}_1^{-1}(u_{i1}), \ldots, \hat{F}_d^{-1}(u_{id})$.
Output: New sample Y' of size m

2.3 Theoretical comparison of latent space modeling techniques

In this section, we compare the modeling capabilities and features of the methods discussed so far on theoretical grounds. For this, it would be interesting to know, what latent space distribution an AE generates. Unfortunately, except for some simple edge cases, this is unknown. The latent space distribution will depend on both the input distribution and the network architecture since the architecture dictates how the inputs are transformed into the latent space. It is well-known that simple AEs with specific activation functions perform some kind of singular value decomposition (Baldi & Hornik, 1989; Kramer, 1991). In the case of a one-layer encoder (input plus one hidden layer) and a linear activation function, an AE performs principal component analysis (Oja, 1982). Hence, if the inputs are multivariate normally distributed, the latent space distribution is a multivariate normal distribution because a linear transformation of a normal distribution obtains a normal distribution. Apart from this, making claims about the latent space distribution is tough. Thus, one should expect atypical distributions that can still be high-dimensional and are represented by a limited number of points during training. Therefore, the method to model the latent space distribution should be flexible such that a wide range of distribution classes can be represented and should not suffer from overfitting in high dimensions.

The multivariate Gaussian distribution can only represent Gaussian distributions and, thus, is limited in this aspect. However, it is easy to understand and for the Gaussian inputs one-layer linear activation AE obtains the true distribution. Particularly, the multivariate Gaussian is unable to represent marginal fat-tails, asymmetries as well as tail dependence—a concept for extreme observations in multivariate distributions, cf. Nelsen (2006). As a mixture of Gaussian distributions, the GMM is unable to model fat-tails and tail dependence as well. KDE is a non-parametric method. However, often a Gaussian kernel is used. Thus, tail dependence cannot be modeled. Also, see the analysis in Coblenz et al. (2022). Additionally, KDE suffers from the curse of dimensionality.

On the other hand, a vine copula in its general form can model any distribution (Czado, 2019). Yet, using the simplifying assumption, non-parametric building blocks, and truncation has the effect that some distributions might not be represented well. However, in its non-parametric version it does not suffer from the curse of dimensionality (Nagler & Czado, 2016). In contrast to that, the empirical beta copula is a non-parametric copula method that is able to represent any copula asymptotically (Segers et al., 2017). Yet, in a finite sample regime it is restricted with respect to tail dependence. A further point in favor of copula methods is that marginal distributions can be chosen freely, which particularly enables them to incorporate fat-tailed marginal distributions. In summary, from the perspective of modeling capabilities, copula methods seem superior compared to the other methods.

A further case we want to make for copula methods stems from the fact that obtaining samples from them is done in two steps. First, pseudo-samples are generated from the respective copula. Second, the pseudo-samples are transformed to the latent space distribution via the inverse marginal CDFs (e.g., see Algorithm 2). Hence, due to the two-step sampling procedure of the copula methods, it is possible to change the marginal distributions when sampling. In particular, this feature can be used to synthesize data. For example, when sampling from a latent space of human faces, one could sample from the copula of human faces without glasses, however, transform the pseudo-samples with marginal CDFs of human faces with glasses in order to obtain images with glasses. In Sect. 3.2.6 we show that this kind of recombination is feasible. Note that for the non-copula methods, this

is impossible, since their dependence structure and marginal distributions are mathematically tightly linked and they sample from the latent space directly.

Lastly, targeted sampling, i.e., sampling from a specific class directly, is most easily done with KDE and the empirical beta copula without amending the training phase. This works by sampling from the estimated density of the corresponding sub-group in the case of KDE. For the empirical beta copula we can randomly choose among rows in the rank matrix that share the desired class we want to sample from, i.e., we sample I in the first for-loop in Algorithm 2 conditional on the sub-group. In Sect. 3.2.6, we conduct experiments for targeted sampling for these methods.

In the next section, we present the experiments and results of a comparative study including all mentioned methodologies to model the latent space.

3 Experiments

In this section, we present the setup of our experiments in 3.1 and the results in 3.2. For each dataset, we use the same dataset specific architecture for the AE in all experiments but replace the modeling technique for the latent space. We further include a standard VAE in our experiments to serve as a benchmark.

3.1 Setup

In Sect. 3.1.1, we first describe the overall methodology and the usage of the methods proposed in Sect. 2. We then introduce the datasets used, architectures for neural net models, and evaluation framework in Sects. 3.1.2, 3.1.3, and 3.1.4, respectively.

3.1.1 Methodology

We train an AE consisting of two neural nets, an encoder f, and a decoder g. The encoder f maps data X from the original space to a lower-dimensional space, while the decoder g reconstructs this low-dimensional data Y from the low-dimensional latent space to the original space (see Fig. 1). We train both neural nets in a way that the reconstruction loss is minimized, i.e., that the reconstructed data $X' = g(f(X))$ is as similar to the original data X as possible. In the second step, we model the latent space data Y with a multivariate Gaussian distribution, a GMM, a multivariate KDE, a KDE where we only model the marginal distributions (the Independent approach, cf. Sect. 2.1.3), an r-vine copula, an empirical beta copula, and a Real NVP. Thus, we fit models with different flexibility and complexity while keeping the training process of the AE untouched. Last, new samples are generated by decoding random samples from the learned model in the latent space. Note that such an approach is only reasonable when the underlying AE has learned a relevant and interesting representation of the data and the latent space is smooth. We demonstrate this in Sect. 3.2.1. As a benchmark, we also fit a standard VAE to the data.

3.1.2 Datasets

We conduct experiments on one small-scale, one medium-scale, and one large-scale dataset. The small-scale MNIST dataset (LeCun et al., 2010) includes binary images of digits, while

the medium-scale SVHN dataset (Netzer et al., 2011) contains images of house numbers in Google Street View shots. The large-scale CelebA dataset (Liu et al., 2015) consists of celebrity images covering 40 different face attributes. We split data into a train set and a test set of 2000 samples which is a commonly used size for evaluation (Tagasovska et al., 2019; Xu et al., 2018). Note that the data sets cover different dimensionalities in the latent space, allowing for a thorough assessment of the methods under investigation.

3.1.3 Architectures and implementation details

We implemented the experiments in Python 3.8 (Van Rossum & Drake Jr, 1995) using `numpy 1.22.0, scipy 1.7.1, scikit-learn 1.1.0 and pytorch 1.10.1` (Harris et al., 2020; Paszke et al., 2019; Pedregosa et al., 2011; Virtanen et al., 2020). The AEs were trained using the Adam optimizer with learning rate 0.001 for MNIST and 0.0005 for SVHN and CelebA. A weight decay of 0.001 was used in all cases. Batch sizes were fixed to 128 (MNIST), 32 (SVHN) and 100 (CelebA) samples for training, while the size of the latent space was set to 10 (MNIST), 20 (SVHN), and 100 (CelebA) according to the datasets' size and complexity. Training was executed on a separate train set and evaluated on a hold-out test set of 2000 samples, similar to Tagasovska et al. (2019). For comparison, we have resorted to the r-vine copula implementation of Tagasovska et al. (2019) and to performance metrics from Xu et al. (2018). We trained the AEs on an NVIDIA Tesla V100 GPU with 10 Intel Xeon Gold 6248 CPUs. The experiments are executed afterwards on a PC with an Intel i7-6600U CPU and 20GB RAM.

We use the same architecture for the AEs and VAEs per dataset as described below. All models were trained for an ideal number of epochs by minimizing the Binary Cross Entropy loss. *MNIST*

Encoder:

$$\begin{aligned} x \in R^{32\times 32} \rightarrow \quad & Conv_{32} \rightarrow BN \rightarrow \quad ReLu \\ \rightarrow \quad & Conv_{64} \rightarrow BN \rightarrow \quad ReLu \\ \rightarrow \quad & Conv_{128} \rightarrow BN \rightarrow \quad ReLu \\ & \rightarrow \quad FC_{10} \end{aligned}$$

Decoder:

$$\begin{aligned} y \in R^{10} \rightarrow FC_{100} \rightarrow \quad & ConvT_{128} \rightarrow BN \rightarrow \quad ReLu \\ \rightarrow \quad & ConvT_{64} \rightarrow BN \rightarrow \quad ReLu \\ \rightarrow \quad & ConvT_{32} \rightarrow BN \rightarrow \quad ReLu \\ & \rightarrow \quad FC_{1} \end{aligned}$$

For all (de)convolutional layers, we used 4×4 filters, a stride of 2, and a padding of 1. *BN* denotes batch normalization, *ReLU* rectified linear units, and *FC* fully connected layers. Last, $Conv_k$ and $ConvT_k$ denotes convolution and deconvolution with k filters, respectively.

SVHN In contrast to the MNIST dataset, images in SVHN are colored. We do not use any preprocessing in this dataset.

Encoder:

$$\begin{aligned} x \in R^{3\times32\times32} \rightarrow \quad & Conv_{64} \rightarrow BN \rightarrow \quad ReLu \\ \rightarrow \quad & Conv_{128} \rightarrow BN \rightarrow \quad ReLu \\ \rightarrow \quad & Conv_{256} \rightarrow BN \rightarrow \quad ReLu \\ & \rightarrow FC_{100} \rightarrow \quad FC_{20} \end{aligned}$$

Decoder:

$$\begin{aligned} y \in R^{20} \rightarrow FC_{100} \rightarrow \quad & ConvT_{256} \rightarrow BN \rightarrow \quad ReLu \\ \rightarrow \quad & ConvT_{128} \rightarrow BN \rightarrow \quad ReLu \\ \rightarrow \quad & ConvT_{64} \rightarrow BN \rightarrow \quad ReLu \\ \rightarrow \quad & ConvT_{32} \rightarrow BN \rightarrow \quad ReLu \\ & \rightarrow \quad FC_1 \end{aligned}$$

Notations are the same as above.

CelebA Images in CelebA are colored. Further, we first took central crops of 140×140 and resized the images to a resolution of 64×64.

Encoder:

$$\begin{aligned} x \in R^{3\times64\times64} \rightarrow \quad & Conv_{64} \rightarrow BN \rightarrow \quad LeakyReLu \\ \rightarrow \quad & Conv_{128} \rightarrow BN \rightarrow \quad LeakyReLu \\ \rightarrow \quad & Conv_{256} \rightarrow BN \rightarrow \quad LeakyReLu \\ \rightarrow \quad & Conv_{512} \rightarrow BN \rightarrow \quad LeakyReLu \\ & \rightarrow FC_{100} \rightarrow \quad FC_{100} \end{aligned}$$

Decoder:

$$\begin{aligned} y \in R^{100} \rightarrow FC_{100} \rightarrow \quad & ConvT_{512} \rightarrow BN \rightarrow \quad ReLu \\ \rightarrow \quad & ConvT_{256} \rightarrow BN \rightarrow \quad ReLu \\ \rightarrow \quad & ConvT_{128} \rightarrow BN \rightarrow \quad ReLu \\ \rightarrow \quad & ConvT_{64} \rightarrow BN \rightarrow \quad ReLu \\ \rightarrow \quad & ConvT_{32} \rightarrow BN \rightarrow \quad ReLu \\ & \rightarrow \quad FC_1 \end{aligned}$$

LeakyReLU uses a negative slope of 0.2, and padding was set to 0 for the last convolutional layer of the encoder and the first layer of the decoder. All other notations are as above.

In our experiments, we also used a Real NVP normalizing flow (Dinh et al., 2017) as a neural network benchmark competitor to model the latent space of the AE. For all data sets, we use spatial checkerboard masking, where the mask has a value of 1 if the sum of coordinates is odd, and 0 otherwise. For the MNIST data set, we use 4 coupling layers with 2 hidden layers each and 256 features per hidden layer. Similarly, for the SVHN data set, we also use four coupling layers with two hidden layers each and 256 hidden layer features.

Lastly, for the CelebA data set, we use four coupling layers with two hidden layers each and 1024 hidden layer features. For all data sets, we applied a learning rate of 0.0001 and train the Real NVP normalizing flow for 2000 epochs.

3.1.4 Evaluation

Evaluation of results is performed in several ways. First, we visually compare random images generated by the models. Second, we evaluate the results with the framework proposed by Xu et al. (2018), since a log-likelihood evaluation is known to be incapable of assessing the quality (Theis et al., 2016) and unsuitable for non-parametric models. Based on their results, we choose five metrics in our experiments: The earth mover distance (EMD), also known as Wasserstein distance (Vallender, 1974); the mean maximum discrepancy (MMD) (Gretton et al., 2007); the 1-nearest neighbor-based two-sample test (1NN), a special case of the classifier two-sample test (Lopez-Paz & Oquab, 2017); the Inception Score (Salimans et al., 2016); and the Frêchet inception distance (Heusel et al., 2017). The latter two are over ResNet-34 softmax probabilities and, thus, are only available for the SVHN and CelebA datasets. In line with Tagasovska et al. (2019) and as proposed by Xu et al. (2018), we further apply the EMD, MMD, and 1NN over feature mappings in the convolution space over ResNet-34 features. For all metrics except the Inception Score, lower values are preferred. For more details on the metrics, we refer to Xu et al. (2018).

Next, we evaluate the ability to generate new, realistic images by the different latent space modeling techniques. Therefore, we compare new samples with their nearest neighbor in the latent space stemming from the original data. This shows us whether the learned distribution covers the whole latent space, or stays too close to known examples, i.e., the model does not generalize enough. Finally, we compare other features of the tested models, such as their ability of targeted sampling and of recombining attributes.

3.2 Results

In the following, we show results for our various experiments. First, we present image interpolation capabilities of the underlying autoender model. Second, we show visual results for each of the methods investigated to gain a qualitative understanding of their differences. Third, we compare the methods in terms of performance metrics. Fourth, we evaluate the latent space and nearest neighbors in the latent space. Finally, we address computing times and discuss targeted sampling and recombination of image features.

3.2.1 Image interpolation of the autoencoder

We show that our used AE learned a relevant and smooth representation of the data by interpolation in the latent space and, thus, modeling the latent space for generating new images is reasonable. For example, consider two images A and B with latent variables $y_{A,1}, \ldots, x_{A,100}$ and $y_{B,1}, \ldots, y_{B,100}$. We now interpolate linearly in each dimension between these two values and feed the resulting interpolation to the decoder to get the interpolated images. We exemplify this on the CelebA dataset.

Each row in Fig. 3 shows a clear linear progression in ten steps from the first face on the left to the final face on the right. For example, in the last row, we see a female with blonde hair slowly transforming into a male with a beard. The transition is smooth, and no sharp changes or random images occur in-between.

3.2.2 Visual results

Figure 4 shows images generated from each method for MNIST, SVHN, and CelebA. The GMM model is composed of 10 elements, and the KDE is constructed using a Gaussian kernel with a bandwidth fitted via a grid search and 10-fold cross-validation.

For the MNIST dataset, we observe the best results for the empirical beta copula (row 6) and KDE (row 3), while the other methods seem to struggle a bit. For the CelebA, our visual observations are slightly different. All methods produce images that are clearly recognizable as faces. However, the Gaussian samples in row 1 and independent margins in row 2 create images with some unrealistic artifacts, blurry backgrounds, or odd colors. This is also the case for the GMM in row 4 and the vine copula in row 5, but less severe. We believe that this comes from samples of an empty area in the latent space, i.e., where none of the original input images were projected to. In contrast to that, the samples in the latent space of the KDE, empirical beta copula, and Real NVP stay within these natural bounds, producing good results after passing the decoder (rows 3, 6, 8). Recall that all methods use the same AE and only differ by means of sampling in the latent space. From our observations, we also conclude that the AE for the CelebA dataset is less sensitive toward modeling errors in the latent space since all images are clearly recognizable as faces. In contrast, for the MNIST dataset, not all images clearly show numbers. Similar results can be observed for SVHN.

3.2.3 Numerical results

We report results over the number of samples in the latent space in Figs. 5–7. This, at first sight, unusual perspective visualizes the capability to reach good performance even for small sample sizes in the latent space. In a small-sample regime, it is crucial to assess how fast a method adapts to data in the latent space and models it correctly. We see that all methods perform well for sample sizes as small as $n = 200$. Note that table versions of these figures are available in Appendix A.

The numerical results prove that dependence truly matters within the latent space, since the independent model is outclassed across metrics and data sets by the other models that

Fig. 3 Interpolation in the latent space of samples of the autoencoder

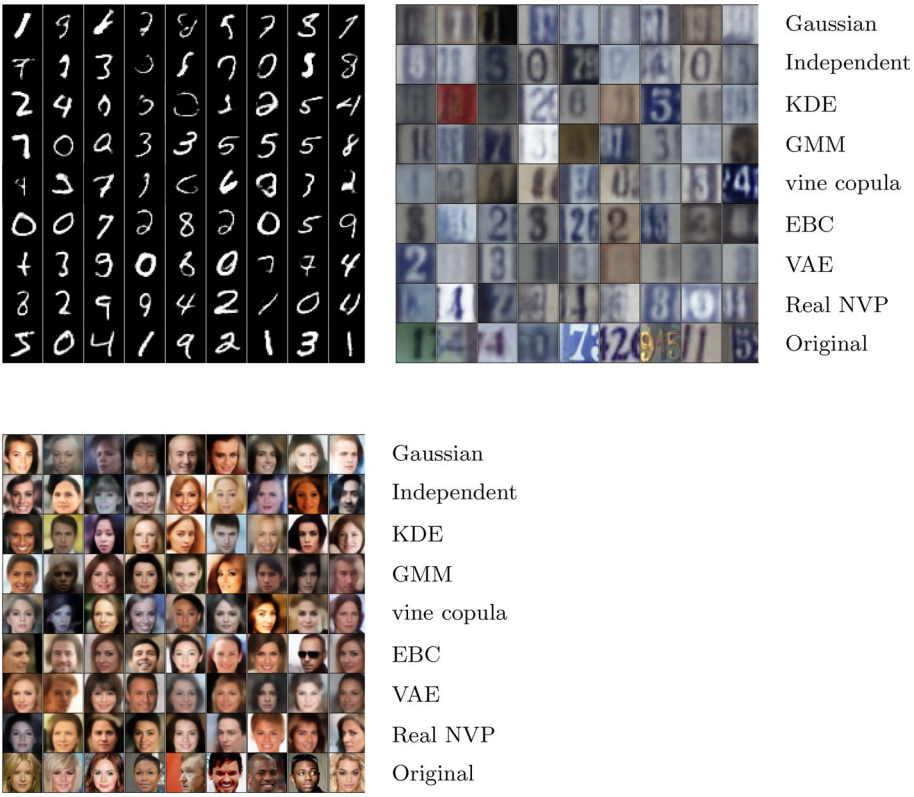

Fig. 4 Comparison of random, synthetic samples of different autoencoder models row by row for MNIST (top left), SVHN (top right), and CelebA (bottom left). Original input samples are given in the last row. EBC stands for empirical beta copula

incorporate dependence. The Gaussian model sometimes performs close to the independent model, sometimes clusters in the middle, which indicates that a multivariate normal distribution might be too simple to model the latent space adequately. GMM and KDE perform consistently well across metrics and data sets. The vine copula performs somewhere in the middle. The empirical beta copula also shows strong results, which are comparable to GMM and KDE. Of the two benchmark models the VAE results are mixed. On the other hand, the Real NVP performs well. However, it is the most complex latent space modeling technique employed. In summary, GMM, KDE, and empirical beta copula perform consistently well compared to the other methods.

3.2.4 Nearest neighbour and latent space evaluation

Next, we evaluate the different modeling techniques in their ability to generate new, realistic images. For this, we focus on images from the CelebA dataset in Fig. 8. First, we create new, random samples with the respective method (top row) and then compare these with their decoded nearest neighbor in the latent space (middle row). The bottom row displays the latent space nearest neighbor in the original data space before applying the AE. By doing so, we are able to disentangle two effects. First, the effect from purely

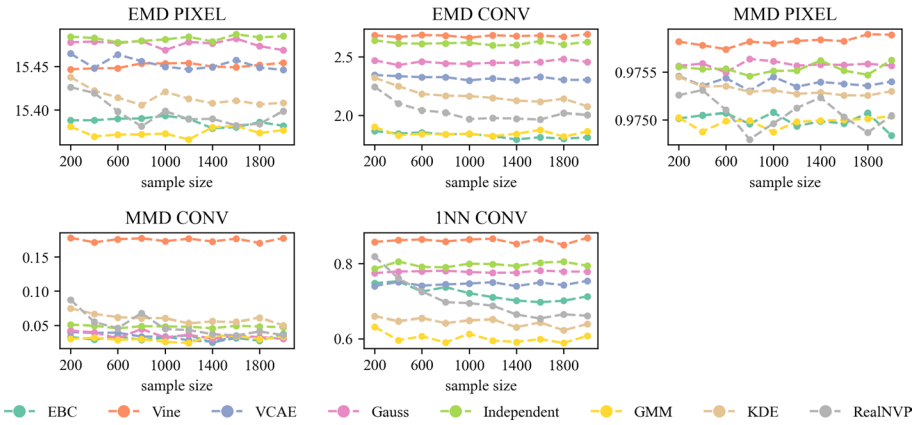

Fig. 5 Performance metrics of generative models on MNIST, reported over latent space sample size. Note that they only differ in the latent space sampling and share the same autoencoder. EBC stands for empirical beta copula

encoding-decoding an image and, second, the effect of modeling the latent space. Thus, we can check whether new images are significantly different from the input, i.e., whether the distribution modeling the latent space merely reproduces images or generalizes to some extent.

We observe that the samples from GMM, the vine copula and the Real NVP substantially differ from their nearest neighbors. However, again they sometimes exhibit unrealistic colors and blurry backgrounds. The samples created from KDE and the empirical beta copula look much more similar to their nearest neighbors in the latent space, indicating that these methods do not generalize to the extent of the other methods. However, their samples do not include unrealistic colors or features and seem to avoid sampling from areas where no data point of the original data is present. Thus, they stay in 'natural bounds'. Note that this effect apparently is not reflected in the numerical evaluation metrics. We, therefore, recommend that, in addition to a quantitative evaluation, a qualitative evaluation of the resulting images should always be performed.

To further underpin this point, Fig. 9 shows 2-dimensional TSNE embeddings (van der Maaten & Hinton, 2008) of the latent space for all six versions of the AE on the MNIST dataset. Black points indicate original input data, and colored points are synthetic samples from the corresponding method. We see that the KDE, as well as the empirical beta copula, stay close to the original space. The samples from the GMM and Real NVP also seem to closely mimic the original data, whereas the other methods fail to do so. This visualization confirms our previous conjecture that some algorithms tend to sample from 'empty' areas in the latent space, leading to unrealistic results.

3.2.5 Computing times

We also report computing times for learning and sampling of the different models for MNIST and CelebA in Table 1. Unsurprisingly, the more straightforward methods such as multivariate Gaussian, Independent, KDE, and GMM, exhibit the lowest sampling times. The Real NVP shows the highest learning time as a neural network is fitted. However, we expect the difference to be much smaller once trained on an appropriate GPU.

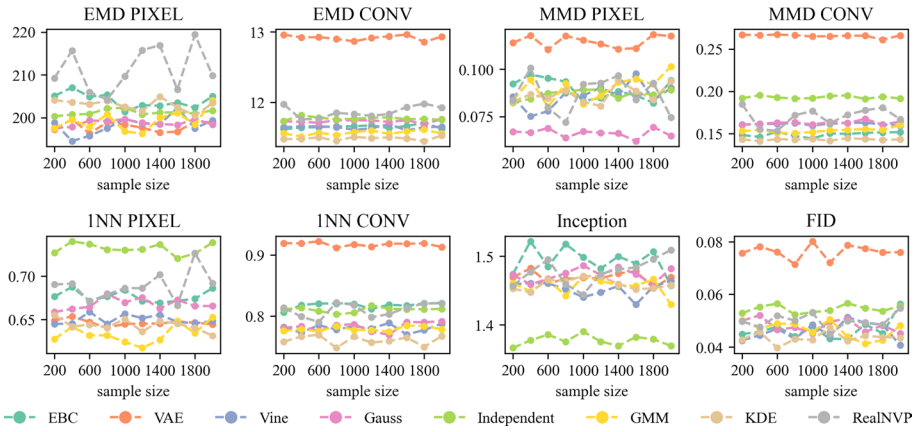

Fig. 6 Performance metrics of generative models on SVHN, reported over latent space sample size. Note that they only differ in the latent space sampling and share the same autoencoder. EBC stands for empirical beta copula

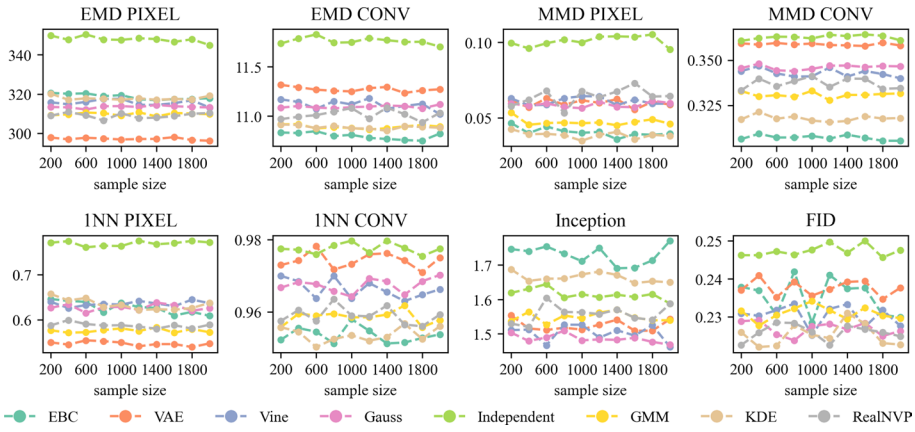

Fig. 7 Performance metrics of generative models on CelebA, reported over latent space sample size. Note that they only differ in the latent space sampling and share the same autoencoder. EBC stands for empirical beta copula

The times also reflect the complexities of the methods in the latent space dimensions. MNIST has a latent space dimensionality d of 10, whereas CelebA has $d = 100$. This change in dimension by one magnitude is approximately reflected by all methods for learning and sampling except for the vine copula, which scales more than linear in d. Note that sampling is linear in sample size n for all methods since we have to sample point by point. For learning on the other hand scaling in n depends on the implementation used. For example, one could use the expectation-maximization algorithm for GMM and scaling in n would depend on the implementation. Also, the vine copula uses maximum likelihood, which can be solved by several numerical methods that scale differently in n. Learning for the empirical beta copula basically consists of sorting the data matrix for each dimension, and thus, has a complexity of $\mathcal{O}(dn \log n)$.

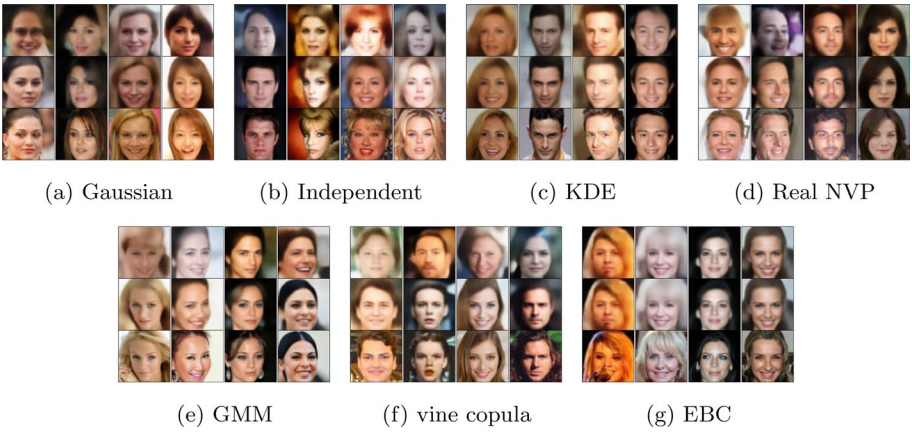

Fig. 8 Nearest neighbor evaluation of the six investigated modeling methods after decoding. EBC stands for empirical beta copula. Top row: Newly generated images. Middle row: Nearest neighbor of new image in the latent space of training samples after decoding. Bottom row: Original input training image of nearest neighbor in latent space

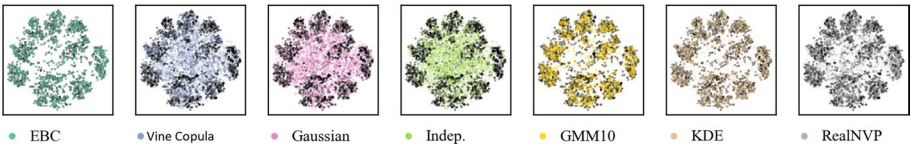

Fig. 9 TSNE embeddings of samples in the latent space of the MNIST dataset. Points from the original input training data Y are shown in black, whereas new, synthetic samples Y' in the latent space stemming from the different modeling methods are colored. EBC stands for empirical beta copula

3.2.6 Targeted sampling and recombination

Last, we discuss other features of the tested methods, such as targeted sampling and recombination. In contrast to the other techniques, the KDE and the empirical beta copula allow for targeted sampling. Thus, we can generate new images with any desired characteristic directly, e.g., only ones in a data set of images of numbers. Note that this can be done after the training, i.e., during training time the characteristic can be unknown and is only needed as soon as it should be sampled from. In the case of the KDE, this simply works by sampling from the estimated density of the corresponding sub-group. In the case of the empirical beta copula, we randomly choose among rows in the rank matrix of original samples that share the desired specific attribute, i.e., we sample I in the first for-loop in Algorithm 2 conditional on the sub-group. Thus, newly generated samples stay close to the original input and therefore share the same main characteristics. Other approaches are also possible, however, they need further tweaks to the model, training, or sampling such as the conditional variational autoencoder (Sohn et al., 2015).

The second feature we discuss is recombination. By using copula-based models, we can facilitate the decomposition idea and split the latent space in its dependence structure and marginal distributions, i.e., we combine the dependence structure of images with a specific attribute with the marginal distributions of images with different attributes. Therefore,

Table 1 Modeling and sampling time in the CelebA and MNIST dataset of 2000 artificial samples based on a latent space of size $n = 2000$ in [s]

Method	CelebA Learn	CelebA Sample	MNIST Learn	MNIST Sample
Gaussian	<0.01	0.01	0.002	0.002
Independent	4.10	0.07	0.393	0.003
KDE	75.25	0.01	13.958	0.001
GMM	1.35	0.03	0.115	0.004
Vine copula	306.97	148.48	10.345	4.590
Empirical beta copula	3.41	59.36	0.328	5.738
Real NVP	2541.19	3.69	341.608	0.477

copula-based methods might allow controlling the attributes of created samples to some extent. Our experiments suggest that the dependence structure provides the basic properties of an image, while the marginal distributions are responsible for details (see, e.g., Fig. 10). However, we want to point out that it is not generally clear what information is embedded in the dependence structure and what information is in the marginal distributions of the latent space. This might also depend on the AE and the dataset at hand and certainly is a topic for future research. That being said, using such a decomposition enables higher flexibility and hopefully fuels new methodological developments in this field.

4 Discussion

In this section, we want to discuss the results of our experiments and want to express some further thoughts. In summary, we observed that sampling from the latent space via the investigated methods is indeed a viable approach to turn a plain-vanilla AE into a generative model and may be promising for application in more advanced AEs. However, each modeling approach in this setting comes with its own restrictions, advantages, and problems. Also, note that we based our analysis and conducted experiments with image data only. Therefore, some of our observations and conclusions might not be transferrable to other data domains. Yet, our hope is that researchers dealing with tabular, text, or audio data might benefit from this work, too.

We witness a trade-off between the ability to generalize, i.e., to create genuinely new images, and sample quality, i.e., to avoid unrealistic colors or artefacts. In cases where new data points are sampled in the neighborhood to existing points (as in the KDE or the empirical beta copula), the newly generated data stay in somehow natural bounds and provide realistic, but not completely new, decoded samples. On the other hand, modeling the latent space too generically leads to bad-quality images. We believe this is similar to leaving the feasible set of an optimization problem or sampling from a wrong prior. While being close to actual points of the original latent space, new samples stay within the feasible set. By moving away from these points, the risk of sampling from an unfeasible region and, thus, creating unrealistic new samples increases. Recombination via a copula-based approach of marginal distributions and dependence structures offers the possibility to detect new feasible regions in the latent space for the creation of realistic images. Also, interpolating by building convex combinations of two points in the latent space seems reasonable. However, without further restrictions during training (see, e.g., the discussion in Ghosh et al. (2020)), we cannot guarantee proper interpolation results. Further, we observe that the mentioned

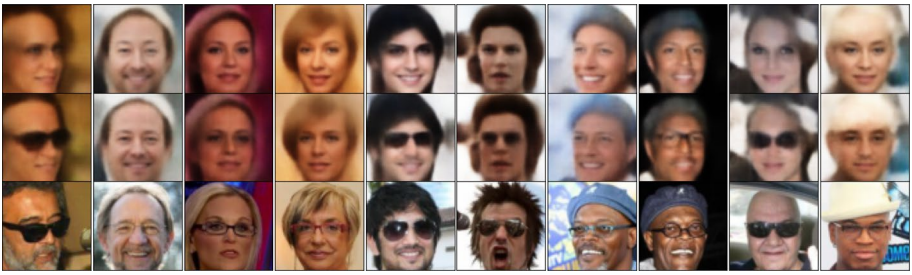

Fig. 10 Samples from recombination experiments with the empirical beta copula. Glasses are removed by using the marginal distribution of the training data without glasses in the latent space. Top row: Samples created with the dependence structure in latent space from samples with glasses and marginal distributions in latent space from samples without glasses. Middle row: Nearest neighbor of newly created sample in the training data after decoding. Bottom row: Original input image of nearest neighbor in latent space

trade-off is not reflected by the performance metrics. Therefore, we strongly recommend not only checking quantitative results but also finding and analyzing the nearest neighbor in the original data to detect the pure reproduction of images. This also reveals that the development of further evaluation metrics could be beneficial.

A closely related issue is the choice of a parametric vs. a non-parametric modeling method in the latent space. Parametric methods can place probability mass in the latent space, where no data point of the original input data was observed. Thus, parametric methods are able to generate (truly) new data, subject to their assumptions. However, if the parametric assumption is wrong, the model creates samples from 'forbidden' areas in the latent space leading to unrealistic images. In spite of this, carefully chosen parametric models can be beneficial, and even a log-likelihood is computable and traceable (although here we do not use it for training). Non-parametric methods avoid a model decision and possible source of error completely but are closely bound to the empirical distribution of the given input data. Consequently, such methods can miss important areas of the latent space but create more realistic images. Furthermore, adjusting parameters of the non-parametric models, such as increasing bandwidths or lowering truncation levels, offer possibilities to slowly overcome these limitations.

Besides the major points above, the empirical beta copula and KDE offer an easy way of targeted sampling without additional training effort. This can be beneficial for various applications and is not as straightforward with other methods. Lastly, the investigated methods differ in their runtime. While vine copula learning and sampling is very time-intensive for high dimensions, the empirical beta copula is much faster but still outperformed by the competitors. For the non-copula methods, the GMM is really fast in both datasets while still capturing the dependence structure to some extent. In contrast to that, the Real NVP needs more time for training but is rather quick in generating new samples.

We conclude that the optimal method to sample from the latent space of an AE depends on the goals of the user. Besides runtime considerations, the specific application of the AE matters. For example, if one is interested in targeted sampling, the empirical beta copula or KDE should be applied. Recombination experiments call for a copula-based approach, whereas in all cases, the trade-off between generalization and out-of-bound sampling should be considered. In the following, we sum this up into take-away messages:

- If runtime for training or sampling is an issue, choose a multivariate Gaussian distribution or GMM.
- If targeted sampling is required without amending the training phase, choose KDE or empirical beta copula.
- If you want to experiment with feature recombination and with the synthesis of data, choose a copula-related method.

In particular, for image-related data:

- Always visually inspect generated images and do not rely on quantitative metrics alone.
- If sample quality is required, but newly generated images can be close to original images, choose a non-parametric method.
- If sample quality is of minor interest and images can be unrealistic, choose a parametric method.

Appendix A Table Versions of Figs. 5, 6 and 7

See Tables 2, 3, 4.

Table 2 Performance metrics of generative models on MNIST, reported over latent space sample size

	$n =$	200	400	600	800	1000	1200	1400	1600	1800	2000
EMD PIXEL	EBC	15.3881	15.3883	15.39	15.3902	15.3933	15.3899	15.379	15.3802	15.3861	15.3818
	VAE	15.4466	15.4475	15.4478	15.4533	15.4532	15.4538	15.4496	15.4488	15.4515	15.4539
	Vine	15.4646	15.4481	15.4635	15.456	15.4496	15.4464	15.4492	15.457	15.4485	15.4458
	Gauss	15.4777	15.4784	15.4767	15.4791	15.4685	15.478	15.4762	15.4819	15.4731	15.4685
	Independent	15.484	15.4825	15.4776	15.4791	15.4809	15.4839	15.4785	15.4869	15.4832	15.4846
	GMM	15.3809	15.3696	15.3717	15.3719	15.3728	15.3661	15.3799	15.3817	15.3735	15.3768
	KDE	15.4376	15.4219	15.4139	15.4053	15.4205	15.4127	15.4074	15.4105	15.4061	15.4081
	RealNVP	15.4259	15.4192	15.3982	15.3812	15.3984	15.3895	15.3898	15.3824	15.3841	15.3987
EMD CONV	EBC	1.8647	1.8441	1.8539	1.8367	1.8417	1.822	1.7934	1.8121	1.8001	1.8116
	VAE	2.6853	2.6692	2.6867	2.683	2.6634	2.6851	2.6775	2.6824	2.6727	2.6934
	Vine	2.3441	2.3336	2.3251	2.3255	2.2975	2.3157	2.2986	2.3292	2.3013	2.3023
	Gauss	2.4696	2.4285	2.46	2.4432	2.4393	2.4484	2.4489	2.4564	2.4819	2.4553
	Independent	2.641	2.6155	2.6128	2.6163	2.621	2.5981	2.6007	2.635	2.6042	2.6286
	GMM	1.9002	1.8286	1.8419	1.8371	1.8426	1.8225	1.8421	1.8755	1.8173	1.8626
	KDE	2.3201	2.248	2.1814	2.1662	2.1629	2.1473	2.1244	2.1146	2.1403	2.0747
	RealNVP	2.2428	2.1001	2.0404	2.022	1.9688	1.9769	1.9725	1.9639	2.0175	2.0033
MMD PIXEL	EBC	0.975	0.975	0.9751	0.975	0.9751	0.9749	0.975	0.975	0.9751	0.9748
	VAE	0.9758	0.9758	0.9757	0.9758	0.9758	0.9758	0.9758	0.9758	0.9759	0.9759
	Vine	0.9755	0.9754	0.9754	0.9753	0.9754	0.9753	0.9754	0.9754	0.9754	0.9754
	Gauss	0.9756	0.9756	0.9755	0.9756	0.9756	0.9756	0.9756	0.9756	0.9756	0.9756
	Independent	0.9756	0.9755	0.9755	0.9755	0.9755	0.9755	0.9756	0.9755	0.9755	0.9756
	GMM	0.975	0.9749	0.975	0.975	0.9749	0.975	0.975	0.975	0.975	0.975
	KDE	0.9754	0.9753	0.9754	0.9753	0.9753	0.9753	0.9753	0.9753	0.9753	0.9753
	RealNVP	0.9753	0.9753	0.9751	0.9748	0.975	0.9751	0.9752	0.975	0.9749	0.975

Table 2 (continued)

	$n =$	200	400	600	800	1000	1200	1400	1600	1800	2000
MMD CONV	EBC	0.0332	0.0294	0.0344	0.0289	0.032	0.0358	0.0252	0.0318	0.0279	0.0379
	VAE	0.1777	0.171	0.1756	0.1769	0.1728	0.1764	0.1724	0.1764	0.1703	0.1772
	Vine	0.0405	0.0393	0.0395	0.0338	0.0339	0.0283	0.0264	0.0333	0.0309	0.0322
	Gauss	0.0426	0.04	0.0312	0.0449	0.0327	0.0366	0.0307	0.035	0.033	0.0305
	Independent	0.0513	0.0496	0.0458	0.049	0.0486	0.0482	0.046	0.0495	0.0486	0.0476
	GMM	0.0304	0.0321	0.029	0.0301	0.0261	0.0244	0.0356	0.0345	0.0302	0.0337
	KDE	0.0746	0.0665	0.0617	0.0603	0.0603	0.0532	0.0559	0.0548	0.0612	0.0503
	RealNVP	0.0872	0.0549	0.0462	0.0675	0.045	0.0433	0.0375	0.0356	0.0412	0.036
1NN CONV	EBC	0.7473	0.7525	0.7247	0.7368	0.7205	0.71	0.7007	0.6965	0.7003	0.712
	VAE	0.8575	0.862	0.864	0.8587	0.864	0.866	0.8525	0.8652	0.8497	0.8685
	Vine	0.7398	0.7505	0.7408	0.744	0.7462	0.7498	0.7393	0.749	0.742	0.7535
	Gauss	0.7742	0.7785	0.779	0.7805	0.777	0.775	0.7755	0.7807	0.7785	0.778
	Independent	0.7857	0.805	0.7908	0.7895	0.7993	0.7983	0.7933	0.8023	0.8045	0.7937
	GMM	0.6315	0.5957	0.606	0.59	0.6127	0.595	0.591	0.5987	0.5882	0.6075
	KDE	0.6593	0.6463	0.6543	0.6415	0.6482	0.651	0.6305	0.6432	0.6223	0.639
	RealNVP	0.8188	0.7605	0.7255	0.6965	0.694	0.687	0.6635	0.653	0.664	0.6603

Note that they only differ in the latent space sampling and share the same autoencoder. EBC stands for empirical beta copula

Table 3 Performance metrics of generative models on SVHN, reported over latent space sample size

	$n =$	200	400	600	800	1000	1200	1400	1600	1800	2000
EMD PIXEL	EBC	205.0646	207.0425	204.8999	205.3097	201.3688	202.9168	202.7768	203.4898	202.3456	204.9837
	VAE	197.235	199.6321	196.2942	198.9154	198.3806	197.6376	196.5287	196.7168	199.1068	198.8802
	Vine	198.8596	194.5207	195.767	197.523	199.6563	198.6431	198.4679	202.2577	197.5066	199.354
	Gauss	197.5914	197.8129	199.3459	199.204	199.6569	198.7876	198.522	198.3022	199.668	198.4056
	Independent	200.3096	200.7186	200.8574	202.1628	202.2843	200.7765	201.1615	201.2908	200.9588	201.6758
	GMM	197.5212	199.5104	197.8386	200.8033	196.8094	196.2459	199.9359	201.121	198.4195	203.9762
	KDE	204.0967	203.6038	203.1661	203.909	202.5437	201.5294	204.9247	202.3946	200.4683	203.373
	RealNVP	209.2376	215.6357	205.987	204.2074	209.6158	215.7505	216.9105	206.5466	219.4032	209.8562
EMD CONV	EBC	11.6176	11.6416	11.6464	11.6586	11.6598	11.6731	11.633	11.6631	11.7083	11.6258
	VAE	12.9606	12.9229	12.9245	12.9014	12.8654	12.9145	12.9376	12.9625	12.858	12.9327
	Vine	11.6457	11.6574	11.6583	11.6475	11.6112	11.6302	11.6899	11.6002	11.6456	11.6529
	Gauss	11.7297	11.723	11.7107	11.7456	11.7445	11.7436	11.7215	11.746	11.6906	11.7609
	Independent	11.7336	11.817	11.7822	11.7569	11.7666	11.7799	11.7862	11.7616	11.7646	11.7523
	GMM	11.5597	11.5128	11.5678	11.5331	11.5763	11.5984	11.5527	11.5863	11.6151	11.5537
	KDE	11.4787	11.4785	11.5019	11.4534	11.5094	11.4896	11.5022	11.4876	11.4489	11.5271
	RealNVP	11.9716	11.7588	11.7654	11.8492	11.8337	11.8076	11.834	11.9321	11.981	11.9246
MMD PIXEL	EBC	0.0922	0.0971	0.0953	0.0934	0.083	0.0848	0.0882	0.0862	0.0821	0.0941
	VAE	0.1139	0.1177	0.1103	0.1176	0.1153	0.1133	0.1106	0.111	0.1182	0.1174
	Vine	0.0855	0.0751	0.0778	0.0875	0.086	0.0903	0.086	0.0977	0.0837	0.09
	Gauss	0.067	0.0665	0.0688	0.0638	0.0671	0.0663	0.0661	0.062	0.0694	0.0647
	Independent	0.0823	0.0844	0.0873	0.0883	0.0891	0.0892	0.0847	0.088	0.0864	0.0891
	GMM	0.0833	0.0944	0.08	0.0924	0.0815	0.0858	0.093	0.0948	0.0919	0.1014
	KDE	0.0828	0.0866	0.0848	0.0889	0.0826	0.0806	0.0942	0.0885	0.0839	0.0936
	RealNVP	0.0835	0.1005	0.0797	0.0718	0.0921	0.0927	0.0966	0.0838	0.0927	0.0745

Table 3 (continued)

	$n =$	200	400	600	800	1000	1200	1400	1600	1800	2000
MMD CONV	EBC	0.148	0.146	0.1519	0.1471	0.144	0.1501	0.1494	0.1507	0.1513	0.1515
	VAE	0.2669	0.2664	0.2671	0.2664	0.2648	0.2652	0.2657	0.2658	0.2611	0.2661
	Vine	0.1605	0.1611	0.1614	0.1625	0.1593	0.1616	0.1625	0.1627	0.1607	0.1591
	Gauss	0.1602	0.1615	0.1627	0.1632	0.1606	0.1635	0.1622	0.1673	0.1602	0.1641
	Independent	0.1917	0.1953	0.1925	0.1914	0.1918	0.1944	0.1951	0.1908	0.1936	0.1914
	GMM	0.1527	0.1543	0.1508	0.1502	0.1517	0.1537	0.1543	0.1551	0.1543	0.1599
	KDE	0.143	0.1409	0.1433	0.1427	0.1434	0.1414	0.1445	0.1434	0.1424	0.1429
	RealNVP	0.1845	0.155	0.1531	0.1716	0.1765	0.1622	0.1721	0.1774	0.1806	0.1669
1NN PIXEL	EBC	0.6765	0.6875	0.6715	0.6765	0.684	0.6715	0.6693	0.6715	0.674	0.6862
	VAE	0.6488	0.6535	0.647	0.6403	0.6447	0.645	0.6453	0.6475	0.6463	0.6442
	Vine	0.6447	0.645	0.6587	0.6447	0.6565	0.6515	0.6553	0.6472	0.6455	0.648
	Gauss	0.6585	0.6622	0.6645	0.6795	0.6697	0.6755	0.6625	0.6727	0.6655	0.6655
	Independent	0.7272	0.7405	0.7372	0.731	0.7303	0.7312	0.737	0.7207	0.7262	0.739
	GMM	0.6273	0.6413	0.6315	0.6315	0.624	0.617	0.6263	0.648	0.6342	0.6525
	KDE	0.6553	0.6417	0.644	0.6405	0.6505	0.6357	0.6482	0.645	0.6403	0.6313
	RealNVP	0.6905	0.6915	0.6708	0.6787	0.6862	0.6862	0.7017	0.6662	0.7268	0.6917
1NN CONV	EBC	0.8062	0.818	0.8205	0.8205	0.82	0.8115	0.8185	0.817	0.8207	0.8213
	VAE	0.9195	0.9193	0.9225	0.912	0.9175	0.914	0.9187	0.9185	0.9193	0.9133
	Vine	0.7818	0.7728	0.785	0.7778	0.7832	0.7797	0.7885	0.7697	0.777	0.7875
	Gauss	0.7807	0.783	0.7753	0.7853	0.7865	0.7747	0.7692	0.7903	0.7897	0.7915
	Independent	0.8112	0.8133	0.8083	0.8027	0.8055	0.8175	0.8117	0.8112	0.811	0.8117
	GMM	0.7755	0.7782	0.7747	0.7835	0.7772	0.777	0.7747	0.7845	0.7835	0.7785
	KDE	0.758	0.7663	0.7692	0.7477	0.7663	0.7565	0.759	0.7648	0.7495	0.767
	RealNVP	0.8138	0.799	0.7903	0.822	0.818	0.798	0.8033	0.8138	0.8205	0.8207

Table 3 (continued)

	$n =$	200	400	600	800	1000	1200	1400	1600	1800	2000
Inception	EBC	1.4736	1.5214	1.4847	1.5175	1.4982	1.4817	1.4994	1.4891	1.5065	1.4568
	VAE	1.4689	1.4821	1.4602	1.4666	1.4683	1.4671	1.4747	1.4778	1.4532	1.4692
	Vine	1.4587	1.4605	1.4602	1.4516	1.44	1.4472	1.4562	1.4293	1.4554	1.4638
	Gauss	1.4726	1.4588	1.4667	1.4749	1.4862	1.4723	1.4839	1.4737	1.4591	1.4818
	Independent	1.3661	1.3771	1.3854	1.3751	1.39	1.3754	1.369	1.3819	1.3787	1.3691
	GMM	1.4604	1.4485	1.4654	1.4423	1.4704	1.4634	1.4577	1.4563	1.4666	1.4296
	KDE	1.4524	1.4471	1.4657	1.4629	1.471	1.4702	1.4592	1.4514	1.4531	1.4573
	RealNVP	1.4565	1.4769	1.4949	1.4589	1.4438	1.4739	1.4806	1.4833	1.4957	1.5088
FID	EBC	0.0449	0.0458	0.0478	0.0441	0.0476	0.0431	0.043	0.0486	0.0452	0.0564
	VAE	0.0757	0.0781	0.0761	0.0714	0.0802	0.0721	0.0786	0.0774	0.0759	0.076
	Vine	0.0425	0.0445	0.0469	0.0431	0.0485	0.0442	0.0514	0.0492	0.0484	0.0407
	Gauss	0.05	0.0521	0.0466	0.0474	0.0452	0.0497	0.0507	0.0457	0.0482	0.0452
	Independent	0.0529	0.0552	0.0565	0.0523	0.0532	0.0539	0.0566	0.0547	0.0537	0.0556
	GMM	0.0499	0.047	0.0489	0.0488	0.0457	0.0505	0.0439	0.0412	0.0426	0.0482
	KDE	0.0428	0.0458	0.0397	0.043	0.0427	0.0475	0.0423	0.0442	0.0438	0.0435
	RealNVP	0.0497	0.0476	0.0518	0.0501	0.053	0.0449	0.0499	0.0493	0.0485	0.0548

Note that they only differ in the latent space sampling and share the same autoencoder. EBC stands for empirical beta copula

Table 4 Performance metrics of generative models on CelebA, reported over latent space sample size

	$n =$	200	400	600	800	1000	1200	1400	1600	1800	2000
EMD PIXEL	EBC	320.5845	320.1634	320.3571	318.926	319.4495	317.3742	316.8738	317.4624	317.5348	317.8643
	VAE	297.8469	296.9506	297.7283	297.3804	296.851	297.0947	297.1659	298.071	296.5796	296.2151
	Vine	315.8104	314.7981	316.0198	317.4935	317.5984	314.5214	314.49	315.3626	316.8519	313.123
	Gauss	313.4091	313.2538	312.1493	313.7859	313.8858	313.3982	314.4509	313.7988	313.4617	313.1714
	Independent	349.7951	347.6929	350.3777	347.8184	347.6334	348.4388	348.0184	346.6166	347.9022	344.9577
	GMM	309.2179	309.6393	310.5812	310.2955	308.9687	310.6914	308.6093	310.395	310.0534	309.7396
	KDE	320.2163	317.1157	317.8165	317.6042	317.1128	317.838	317.0763	317.5011	317.2143	319.1484
	RealNVP	308.979	310.8978	309.1692	306.5391	310.2386	307.7524	307.9808	308.7533	309.9704	310.3374
EMD CONV	EBC	10.831	10.8261	10.8447	10.7952	10.8075	10.7763	10.7654	10.7545	10.7473	10.8207
	VAE	11.3155	11.2914	11.2668	11.2549	11.2492	11.2823	11.2938	11.2332	11.2591	11.2708
	Vine	11.1676	11.1425	11.0721	11.1482	11.1163	11.1792	11.0727	11.1005	11.1231	11.0175
	Gauss	11.0837	11.1007	11.085	11.0766	11.0888	11.0944	11.1055	11.0995	11.077	11.1168
	Independent	11.7363	11.787	11.8279	11.7445	11.7481	11.7894	11.7699	11.7511	11.7538	11.7045
	GMM	10.9141	10.9168	10.8723	10.8861	10.8768	10.8624	10.8747	10.8936	10.9063	10.8912
	KDE	10.9129	10.9079	10.8765	10.9004	10.8732	10.8715	10.849	10.8941	10.8878	10.8795
	RealNVP	10.9668	10.9914	11.0079	11.0425	11.0792	10.9727	11.0825	11.0191	10.9334	11.0307
MMD PIXEL	EBC	0.0466	0.0398	0.0442	0.0411	0.0401	0.0407	0.0358	0.0392	0.0387	0.0395
	VAE	0.0613	0.0569	0.0629	0.0586	0.0616	0.0599	0.0623	0.0554	0.0607	0.0587
	Vine	0.0629	0.0582	0.0586	0.0629	0.0644	0.0642	0.0584	0.062	0.0593	0.0594
	Gauss	0.0595	0.0579	0.0596	0.0571	0.0564	0.0606	0.0573	0.057	0.0613	0.0598
	Independent	0.0996	0.096	0.0994	0.1017	0.0999	0.1037	0.104	0.1036	0.1052	0.0953
	GMM	0.0535	0.0454	0.0465	0.0468	0.0464	0.0469	0.045	0.0473	0.049	0.046
	KDE	0.0425	0.0391	0.0394	0.0385	0.0348	0.0388	0.0407	0.0357	0.0387	0.038
	RealNVP	0.0578	0.0621	0.0679	0.0532	0.0677	0.0637	0.0675	0.073	0.0641	0.0644

Table 4 (continued)

	$n =$	200	400	600	800	1000	1200	1400	1600	1800	2000
MMD CONV	EBC	0.3063	0.3092	0.3071	0.3072	0.308	0.3066	0.3087	0.3069	0.3053	0.3053
	VAE	0.3593	0.3586	0.3595	0.3587	0.3592	0.3584	0.3582	0.3578	0.3597	0.3581
	Vine	0.3439	0.3468	0.3426	0.3409	0.341	0.3459	0.3409	0.344	0.3423	0.34
	Gauss	0.3456	0.348	0.3444	0.344	0.3452	0.3469	0.347	0.3462	0.3468	0.3466
	Independent	0.3607	0.3622	0.363	0.3629	0.3621	0.3641	0.3633	0.3642	0.3635	0.361
	GMM	0.3329	0.3299	0.3306	0.3297	0.333	0.3278	0.3308	0.3307	0.3313	0.3316
	KDE	0.3172	0.3213	0.3175	0.3188	0.3165	0.3157	0.3163	0.319	0.3176	0.3179
	RealNVP	0.3333	0.3398	0.3355	0.3384	0.3412	0.3352	0.3398	0.3397	0.3341	0.3345
1NN PIXEL	EBC	0.646	0.6417	0.6395	0.6165	0.637	0.6283	0.6263	0.6095	0.6175	0.6087
	VAE	0.55	0.5452	0.5545	0.5527	0.5497	0.5415	0.5462	0.5462	0.5397	0.5477
	Vine	0.6398	0.626	0.6325	0.635	0.6335	0.6417	0.636	0.63	0.6445	0.6367
	Gauss	0.6265	0.6342	0.6152	0.6275	0.6288	0.6215	0.638	0.6325	0.62	0.6258
	Independent	0.7705	0.7738	0.7602	0.7638	0.763	0.7747	0.767	0.7695	0.7742	0.771
	GMM	0.5767	0.5715	0.5725	0.5773	0.581	0.5753	0.5832	0.5715	0.5795	0.5725
	KDE	0.657	0.643	0.648	0.6283	0.633	0.6217	0.6233	0.6223	0.626	0.6373
	RealNVP	0.5875	0.5993	0.5905	0.5875	0.588	0.5847	0.5803	0.588	0.5805	0.5893
1NN CONV	EBC	0.9523	0.9555	0.9545	0.9513	0.958	0.9548	0.9513	0.9515	0.9528	0.9538
	VAE	0.973	0.9743	0.9783	0.9718	0.9732	0.976	0.9762	0.9743	0.971	0.975
	Vine	0.97	0.9685	0.9638	0.97	0.9638	0.968	0.9655	0.963	0.9647	0.9663
	Gauss	0.9668	0.9682	0.9678	0.9657	0.9643	0.9693	0.9685	0.9647	0.9685	0.9703
	Independent	0.9775	0.9772	0.976	0.9785	0.9797	0.9765	0.9797	0.9778	0.9755	0.9775
	GMM	0.9557	0.9595	0.959	0.9595	0.9585	0.9585	0.9592	0.9617	0.9557	0.9578
	KDE	0.9557	0.9545	0.9503	0.9525	0.9535	0.952	0.953	0.9567	0.9525	0.956
	RealNVP	0.9575	0.9605	0.9575	0.9635	0.959	0.9588	0.9617	0.9565	0.956	0.9592

Table 4 (continued)

	$n =$	200	400	600	800	1000	1200	1400	1600	1800	2000
Inception	EBC	1.7461	1.74	1.7538	1.7328	1.7113	1.7493	1.6904	1.692	1.7132	1.7704
	VAE	1.5541	1.5154	1.511	1.5176	1.5123	1.527	1.541	1.5089	1.509	1.5428
	Vine	1.5109	1.5122	1.4666	1.5264	1.5263	1.4888	1.5102	1.4949	1.5251	1.4616
	Gauss	1.5035	1.4791	1.4895	1.509	1.481	1.4838	1.4825	1.4884	1.4766	1.4683
	Independent	1.6189	1.6309	1.6444	1.6054	1.6149	1.6065	1.6135	1.607	1.6155	1.5879
	GMM	1.5367	1.5638	1.5268	1.5531	1.5455	1.5615	1.5697	1.5469	1.5394	1.5394
	KDE	1.6874	1.6538	1.6601	1.6606	1.6724	1.6808	1.6711	1.6473	1.6528	1.6498
	RealNVP	1.5298	1.5197	1.6043	1.5634	1.5624	1.5575	1.5705	1.5464	1.5404	1.5871
FID	EBC	0.2379	0.2369	0.2305	0.2418	0.2285	0.241	0.2374	0.2376	0.2305	0.2299
	VAE	0.237	0.2409	0.2351	0.2392	0.2356	0.2373	0.2392	0.2394	0.2346	0.2376
	Vine	0.2309	0.2303	0.2318	0.2335	0.2252	0.2321	0.2333	0.2274	0.2318	0.2276
	Gauss	0.2289	0.2291	0.2254	0.2238	0.2276	0.2281	0.2267	0.2284	0.225	0.2263
	Independent	0.2462	0.2462	0.2472	0.2463	0.2476	0.2497	0.2468	0.25	0.2456	0.2475
	GMM	0.2316	0.2277	0.2305	0.2322	0.2341	0.2317	0.2294	0.2324	0.2303	0.2296
	KDE	0.2259	0.2221	0.2224	0.2278	0.2253	0.2244	0.231	0.2282	0.2231	0.2227
	RealNVP	0.2226	0.2259	0.2285	0.2284	0.2261	0.2226	0.2276	0.2267	0.2259	0.2249

Note that they only differ in the latent space sampling and share the same autoencoder. EBC stands for empirical beta copula

Acknowledgements We thank three anonymous referees for their insightful comments. MC gratefully acknowledges support from Forschungsstart project P2022-13-009 of the Carl-Zeiss-Stiftung.

Author Contributions All authors contributed equally.

Funding Open Access funding enabled and organized by Projekt DEAL. The authors declare no competing interests.

Data Availability Code for replication is available at https://github.com/FabianKaechele/SamplingFromAutoencoders.

Declarations

Conflict of interest The authors declare no Conflict of interest.

References

Aas, K. (2016). Pair-copula constructions for financial applications: A review. *Econometrics, 4*(4), 43.

Baldi, P., & Hornik, K. (1989). Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks, 2*(1), 53–58.

Berthelot, D., Raffel, C., Roy, A. & Goodfellow, I. (2019). Understanding and improving interpolation in autoencoders via an adversarial regularizer. In *International conference on learning representations.*

Bishop, C. M. (2006). *Pattern recognition and machine learning (information science and statistics)*. Springer.

Brehmer, J., & Cranmer, K. (2020). Flows for simultaneous manifold learning and density estimation. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan & H. Lin (Eds.), *Advances in neural information processing systems* (Vol. 33, pp. 442–453). Curran Associates, Inc.

Coblenz, M., Grothe, O., Herrmann, K., & Hofert, M. (2022). Smooth bootstrapping of copula functionals. *Electronic Journal of Statistics, 16*(1), 2550–2606.

Cremer, C., Li, X. & Duvenaud, D. (2018). *Inference suboptimality in variational autoencoders.* arXiv:1801.03558

Czado, C. (2019). *Analyzing dependent data with vine copulas: A practical guide with r*. Springer.

Czado, C., & Nagler, T. (2022). Vine copula based modeling. *Annual Review of Statistics and Its Application, 9*(1), 453–477. https://doi.org/10.1146/annurev-statistics-040220-101153

Dai, B., & Wipf, D. (2019). *Diagnosing and enhancing vae models.* arXiv:1903.05789

David, H. A., & Nagaraja, H. N. (2003). *Order statistics*. Wiley.

Dimitriev, A., & Zhou, M. (2021). CARMS: Categorical-antithetic-REINFORCE multi-sample gradient estimator. In A. Beygelzimer, Y. Dauphin, P. Liang & J. W. Vaughan (Eds.), *Advances in neural information processing systems.*

Dinh, L., Sohl-Dickstein, J. & Bengio, S. (2017). Density estimation using real NVP. In *5th international conference on learning representations, ICLR 2017, toulon, france, april 24-26, 2017, conference track proceedings.* OpenReview.net.

Durante, F., & Sempi, C. (2015). *Principles of copula theory*. CRC Press LLC.

Fajtl, J., Argyriou, V., Monekosso, D. & Remagnino, P. (2020). Latent bernoulli autoencoder. In H. D. III & A. Singh (Eds.), *Proceedings of the 37th international conference on machine learning* (Vol. 119, pp. 2964–2974). PMLR.

Ghosh, P., Sajjadi, M. S. M., Vergari, A., Black, M. & Scholkopf, B. (2020). From variational to deterministic autoencoders. In *International conference on learning representations.*

Goodfellow, I., Bengio, Y. & Courville, A. (2016). *Deep learning*. MIT Press. (http://www.deeplearningbook.org)

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S. & Bengio, Y. (2014). Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence & K. Q. Weinberger (Eds.), *Advances in neural information processing systems* (Vol. 27). Curran Associates, Inc.

Gregor, K., Danihelka, I., Graves, A., Rezende, D. & Wierstra, D. (2015). Draw: A recurrent neural network for image generation. In F. Bach & D. Blei (Eds.), *Proceedings of the 32nd international conference on machine learning* (Vol. 37, pp. 1462–1471). Lille, France: PMLR.

Gretton, A., Borgwardt, K., Rasch, M., Schölkopf, B. & Smola, A. (2007). A kernel method for the two-sample-problem. In B. Schölkopf, J. Platt & T. Hoffman (Eds.), *Advances in neural information processing systems* (Vol. 19). MIT Press.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., & Oliphant, T. E. (2020). Array programming with NumPy. *Nature, 585*, 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The elements of statistical learning*. Springer, New York Inc.

Havtorn, J. D. D., Frellsen, J., Hauberg, S. & Maaløe, L. (2021). Hierarchical vaes know what they don't know. In M. Meila & T. Zhang (Eds.), *Proceedings of the 38th international conference on machine learning* (Vol. 139, pp. 4117–4128). PMLR.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B. & Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a local nash equilibrium. In I. Guyon et al. (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc.

Hofert, M., Prasad, A., & Zhu, M. (2021). Quasi-random sampling for multivariate distributions via generative neural networks. *Journal of Computational and Graphical Statistics, 30*(3), 647–670. https://doi.org/10.1080/10618600.2020.1868302

Hudson, D. A., & Zitnick, L. (2021). Generative adversarial transformers. In M. Meila & T. Zhang (Eds.), *Proceedings of the 38th international conference on machine learning* (Vol. 139, pp. 4487–4499). PMLR.

Janke, T., Ghanmi, M. & Steinke, F. (2021). Implicit generative copulas. In *Thirty-fifth conference on neural information processing systems.*

Joe, H. (2014). *Dependence modeling with copulas*. Chapman & Hall/CRC.

Karras, T., Aittala, M., Laine, S., Härkönen, E., Hellsten, J., Lehtinen, J. & Aila, T. (2021). Alias-free generative adversarial networks. In *Thirty-fifth conference on neural information processing systems.*

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I. & Welling, M. (2016). Improved variational inference with inverse autoregressive flow. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 29). Curran Associates, Inc.

Kingma, D. P., & Welling, M. (2014). Auto-Encoding Variational Bayes. In *2nd international conference on learning representations, ICLR 2014, banff, ab, canada, april 14-16, 2014, conference track proceedings.*

Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal, 37*(2), 233–243.

Kulkarni, V., Tagasovska, N., Vatter, T. & Garbinato, B. (2018). Generative models for simulating mobility trajectories. arXiv:1811.12801 [cs.LG]

Larsen, A. B. L., Sønderby, S. K., Larochelle, H. & Winther, O. (2016). Autoencoding beyond pixels using a learned similarity metric. In M. F. Balcan & K. Q. Weinberger (Eds.), *Proceedings of the 33rd international conference on machine learning* (Vol. 48, pp. 1558–1566). New York, New York, USA: PMLR.

LeCun, Y., Cortes, C., Burges, C. (2010). Mnist handwritten digit database. *ATT Labs [Online]. Available:* http://yann.lecun.com/exdb/mnist, *2*.

Lee, J., Kim, H., Hong, Y., Chung, H.W. (2021). Self-diagnosing GAN: Diagnosing underrepresented samples in generative adversarial networks. In *Thirty-fifth conference on neural information processing systems.*

Letizia, N. A., & Tonello, A. M. (2020). Segmented generative networks: Data generation in the uniform probability space. *IEEE Transactions on Neural Networks and Learning Systems*. https://doi.org/10.1109/TNNLS.2020.3042380

Liu, W. (2019). Copula multi-label learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32). Curran Associates, Inc.

Liu, Z., Luo, P., Wang, X. & Tang, X. (2015). Deep learning face attributes in the wild. In *Proceedings of international conference on computer vision (iccv)*.

Lopez-Paz, D., & Oquab, M. (2017). Revisiting classifier two-sample tests. In *5th international conference on learning representations, ICLR 2017, toulon, france, april 24-26, 2017, conference track proceedings*. OpenReview.net.

Ma, J., Chang, B., Zhang, X. & Mei, Q. (2021). CopulaGNN: Towards integrating representational and correlational roles of graphs in graph neural networks. In *International conference on learning representations*.

Maaløe, L., Fraccaro, M., Liévin, V. & Winther, O. (2019). *Biva: A very deep hierarchy of latent variables for generative modeling*.

Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I. & Frey, B. (2016). *Adversarial autoencoders*.

Marino, J., Yue, Y. & Mandt, S. (2018). Iterative amortized inference. In J. Dy & A. Krause (Eds.), *Proceedings of the 35th international conference on machine learning* (Vol. 80, pp. 3403–3412). PMLR.

Masrani, V., Le, T.A. & Wood, F. (2019). The thermodynamic variational objective. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32). Curran Associates, Inc.

Messoudi, S., Destercke, S., & Rousseau, S. (2021). Copula-based conformal prediction for multi-target regression. *Pattern Recognition, 120*, Article 108101.

Mishne, G., Shaham, U., Cloninger, A., & Cohen, I. (2019). Diffusion nets. *Applied and Computational Harmonic Analysis, 47*(2), 259–285. https://doi.org/10.1016/j.acha.2017.08.007

Mondal, A. K., Asnani, H., Singla, P. & Prathosh, A. (2021). Flexae: flexibly learning latent priors for wasserstein auto-encoders. In C. de Campos & M. H. Maathuis (Eds.), *Proceedings of the thirty-seventh conference on uncertainty in artificial intelligence* (Vol. 161, pp. 525–535). PMLR.

Moor, M., Horn, M., Rieck, B. & Borgwardt, K. (2020). Topological autoencoders. In H. D. III & A. Singh (Eds.), *Proceedings of the 37th international conference on machine learning* (Vol. 119, pp. 7045–7054). PMLR.

Nagler, T., & Czado, C. (2016). Evading the curse of dimensionality in nonparametric density estimation with simplified vine copulas. *Journal of Multivariate Analysis, 151*, 69–89. https://doi.org/10.1016/j.jmva.2016.07.003

Nelsen, R. B. (2006). *An introduction to copulas*. Springer.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B. & Ng, A.Y. (2011). Reading digits in natural images with unsupervised feature learning. *Nips workshop on deep learning and unsupervised feature learning 2011*. http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf

Oja, E. (1982). Simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology, 15*(3), 267–273.

Oring, A., Yakhini, Z. & Hel-Or, Y. (2021). Autoencoder image interpolation by shaping the latent space. In M. Meila & T. Zhang (Eds.), *Proceedings of the 38th international conference on machine learning* (Vol. 139, pp. 8281–8290). PMLR.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G. & Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32). Curran Associates, Inc.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830.

Qin, H., Ding, Y., Zhang, X., Wang, J., Liu, X. & Lu, J. (2023). Diverse sample generation: Pushing the limit of generative data-free quantization. In *Ieee transactions on pattern analysis and machine intelligence* (Vol. 45, pp. 11689–11706).

Qin, H., Ma, X., Zheng, X., Li, X., Zhang, Y., Liu, S. & Magno, M. (2024). *Accurate lora-finetuning quantization of llms via information retention*. (arXiv preprint arXiv:2402.05445)

Qin, H., Zhang, M., Ding, Y., Li, A., Cai, Z., Liu, Z. & Liu, X. (2023). Bibench: Benchmarking and analyzing network binarization. In *International conference on machine learning* (pp. 28351–28388).

Qin, H., Zhang, X., Gong, R., Ding, Y., Xu, Y., & Liu, X. (2023). Distribution-sensitive information retention for accurate binary neural network. *International Journal of Computer Vision, 131*(1), 26–47.

Qin, H., Zhang, Y., Ding, Y., Liu, X., Danelljan, M., Yu, F., et al. (2024). Quantsr: accurate low-bit quantization for efficient image super-resolution. In *Advances in neural information processing systems* (Vol. 36).

Rezende, D., & Mohamed, S. (2015). Variational inference with normalizing flows. In F. Bach & D. Blei (Eds.), *Proceedings of the 32nd international conference on machine learning* (Vol. 37, pp. 1530–1538). Lille, France: PMLR.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P. & Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. In *Proceedings of the ieee conference on computer vision and pattern recognition (cvpr)*.

Saha, S., Elhabian, S., & Whitaker, R. (2022). GENs: Generative encoding networks. *Machine Learning, 111*, 4003–4038.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X. & Chen, X. (2016). Improved techniques for training gans. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 29). Curran Associates, Inc.

Segers, J., Sibuya, M., & Tsukahara, H. (2017). The empirical beta copula. *Journal of Multivariate Analysis, 155*, 35–51. https://doi.org/10.1016/j.jmva.2016.11.010

Shen, T., Mueller, J., Barzilay, R. & Jaakkola, T.S. (2020). Educating text autoencoders: Latent representation guidance via denoising. In *Proceedings of the thirty-seventh international conference on machine learning* (p.8719-8729).

Silverman, B. W. (1986). *Density estimation for statistics and data analysis / b.w. silverman*. Chapman and Hall.

Sklar, A. (1959). Fonctions de répartition à n dimensions et leurs marges. *Publications de l'Institut de Statistique de l'Université de Paris, 8*, 229–231.

Sohn, K., Lee, H., Yan, X. (2015). Learning structured output representation using deep conditional generative models. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 28). Curran Associates, Inc.

Tagasovska, N., Ackerer, D., Vatter, T. (2019). Copulas as high-dimensional generative models: Vine copula autoencoders. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32). Curran Associates, Inc.

Takahashi, H., Iwata, T., Yamanaka, Y., Yamada, M. & Yagi, S. (2019). *Variational autoencoder with implicit optimal priors*. AAAI Press.

Theis, L., van den Oord, A. & Bethge, M. (2016). A note on the evaluation of generative models. In *International conference on learning representations.*

Tolstikhin, I., Bousquet, O., Gelly, S. & Schoelkopf, B. (2019). *Wasserstein auto-encoders.*

Tomczak, J., & Welling, M. (2018). Vae with a vampprior. In A. Storkey & F. Perez-Cruz (Eds.), *Proceedings of the twenty-first international conference on artificial intelligence and statistics* (Vol. 84, pp. 1214–1223). PMLR.

Tran, D., Blei, D. & Airoldi, E. M. (2015). Copula variational inference. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 28). Curran Associates, Inc.

Vahdat, A., Kreis, K. & Kautz, J. (2021). Score-based generative modeling in latent space. *Neural information processing systems (neurips).*

Vallender, S. S. (1974). Calculation of the wasserstein distance between probability distributions on the line. *Theory of Probability and Its Applications, 18*, 435–435.

van den Oord, A., Vinyals, O. & Kavukcuoglu, K. (2017). Neural discrete representation learning. In *Proceedings of the 31st international conference on neural information processing systems* (pp. 6309–6318). Curran Associates Inc.

van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research, 9*(86), 2579–2605.

Van Rossum, G., & Drake Jr, F. L. (1995). *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam.

Varshney, S., Verma, V. K., Carin, L. & Rai, P. (2021). CAM-GAN: Continual adaptation modules for generative adversarial networks. In *Thirty-fifth conference on neural information processing systems.*

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., & Cournapeau, D. (2020). SciPy 1.0 Contributors. SciPy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods, 17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2

Xu, H., Chen, W., Lai, J., Li, Z., Zhao, Y. & Pei, D. (2019). *On the necessity and effectiveness of learning the prior of variational auto-encoder.* arXiv:1905.13452

Xu, Q., Huang, G., Yuan, Y., Guo, C., Sun, Y., Wu, F., & Weinberger, K. Q. (2018). An empirical study on evaluation metrics of generative adversarial networks. ArXiv:1806.07755

Yoon, S., Noh, Y.-K. & Park, F. (2021). Autoencoding under normalization constraints. In M. Meila & T. Zhang (Eds.), *Proceedings of the 38th international conference on machine learning* (Vol. 139, pp. 12087–12097). PMLR.

Zhang, Z., Zhang, R., Li, Z., Bengio, Y. & Paull, L. (2020). Perceptual generative autoencoders. In H. D. III & A. Singh (Eds.), *Proceedings of the 37th international conference on machine learning* (Vol. 119, pp. 11298–11306). PMLR.