Chapter 5

Generative Adversarial Networks and Other Generative Models

Markus Wenzel

Abstract

Generative networks are fundamentally different in their aim and methods compared to CNNs for classification, segmentation, or object detection. They have initially been meant not to be an image analysis tool but to produce naturally looking images. The adversarial training paradigm has been proposed to stabilize generative methods and has proven to be highly successful—though by no means from the first attempt.

This chapter gives a basic introduction into the motivation for generative adversarial networks (GANs) and traces the path of their success by abstracting the basic task and working mechanism and deriving the difficulty of early practical approaches. Methods for a more stable training will be shown, as well as typical signs for poor convergence and their reasons.

Though this chapter focuses on GANs that are meant for image generation and image analysis, the adversarial training paradigm itself is not specific to images and also generalizes to tasks in image analysis. Examples of architectures for image semantic segmentation and abnormality detection will be acclaimed, before contrasting GANs with further generative modeling approaches lately entering the scene. This will allow a contextualized view on the limits but also benefits of GANs.

Key words Generative models, Generative adversarial networks, GAN, CycleGAN, StyleGAN, VQGAN, Diffusion models, Deep learning

1 Introduction

Generative adversarial networks are a type of neural network architecture, in which one network part generates solutions to a task and another part compares and rates the generated solutions against a priori known solutions. While at first glimpse this does not sound much different from any loss function, which essentially also compares a generated solution with the gold standard, there is one fundamental difference. A loss function is static, but the "judge" or "discriminator" network part is trainable (Fig. 1). This means that it can be trained to distinguish the generated from the true solutions and, as long as it succeeds in its task, a training signal for the generative part can be derived. This is how the notion of adversaries came into the name GAN. The discriminator part is

Olivier Colliot (ed.), *Machine Learning for Brain Disorders*, Neuromethods, vol. 197, https://doi.org/10.1007/978-1-0716-3195-9_5,

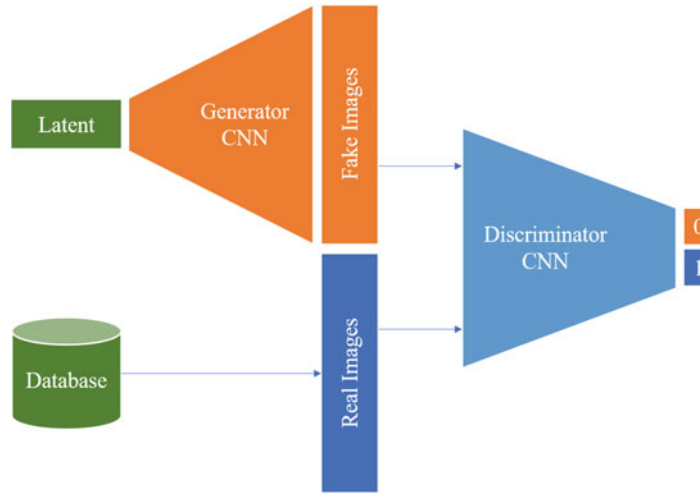

Fig. 1 The fundamental GAN setup for image generation consisting of a generator and a discriminator network; here, CNNs

trained to distinguish true from generated solutions, while the generative part is trained to arrive at the most realistic-appearing solutions, making them adversaries with regard to their aims.

Generative adversarial networks are now among the most powerful tools to create naturally looking images from many domains. While they have been created in the context of image generation, the original publication describes the general idea of how to make two networks learn by competing, regardless of the application domain. This key idea can be applied to generative tasks beyond image creation, including text generation, music generation, and many more.

The research interest skyrocketed in the years after the first publication proposing an adversarial training paradigm [1]. Looking at the number of web searches for the topic "generative adversarial networks" shows how the interest in the topic has rapidly grown but also the starting decline of the last years. Authors since 2014 have cast all kinds of problems into the GAN framework, to enable this powerful training mechanism for a variety of tasks, including image analysis tasks as well. This is surprising at first, since there is no immediate similarity between a generative task and, for example, a segmentation or detection task. Still, as evidenced by the success in these application areas, the adversarial training approach can be applied with benefits. Clearly, the decline in interest can to some degree be attributed to the emergence of best practices and proven implementations, while simultaneously the scientific interest has recently shifted to successor approaches. However, similar to the persistent relevance of CNN architectures like ResNets for classification, Mask R-CNNs for detection, or basic transformer architectures for sequence processing, GANs will

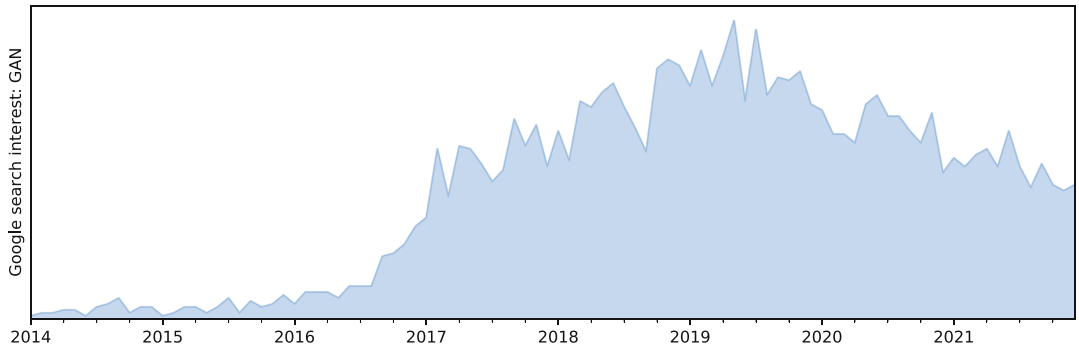

Fig. 2 Google web search-based interest estimate for "generative adversarial networks" since 2014. Relative scale

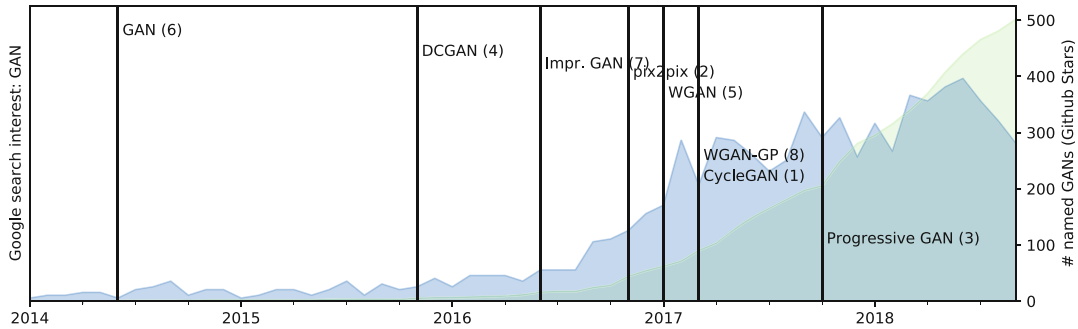

Fig. 3 Some of the most-starred shared GAN code repositories on Github, until 2018. Ranking within this selection in brackets

remain an important tool for image creation and image analysis. The adversarial training paradigm has become an ingredient to models apart from generative aims, providing flexible ways to - custom-tailor loss components for given tasks (compare Figs. 2 and 3).

2 Generative Models

Generative processes are fundamentally hard to grasp computationally. Their nature and purpose is to create something "meaningful" out of something less meaningful (even random). The first question to ask therefore is how this can even be possible for a computer program since, intuitively, creation requires an inventive spirit—call it creativity, to use the term humans tend to associate with this. To introduce some of the terminology and basic concepts that we will use in the remainder of this section, some remarks on human creativity will set the scene.

In fact, creative human acts are inherently limited by our concepts of the world, acquired by learning and experience through the

sensory means we have available, and by the available expressive means (tools, instruments, . . .) with which we can even conceive of creating something. This is true for any kind of creative act, including writing, painting, wood carving, or any other art, and similarly also for computer programming, algorithm development, or science in general. Our limited internal representation of the world around us frames our creative scope.

This is very comparable to the way computerized, programmed, or learned generative processes create output. They have either an in-build mechanism, or a way to acquire such a mechanism, that represents the tools by which creation is possible, as well as a model of the world that defines the scope of outputs. Practically, a CNN-based generative process uses convolutions as the in-built tool and is by this tool geared to produce image-like outputs. The convolutional layers, if not a priori defined, will represent a set of operations defined by a training process and limited in their expressiveness by the training material—by the fraction of the world that was presented. This will lead us to the fundamental notion of how to capture the variability of the "fraction of the world" that is interesting and how to make a neural network represent this partial world knowledge. It is interesting to note at this point that neither for human creative artists nor for neural networks the ability to (re)create convincing results implies an understanding of the way the templates (in the real world) have come into existence. Generating convincing artifacts does not imply understanding nature. Therefore, GANs cannot explain the parts of nature they are able to generate.

2.1 The Language of Generative Models: Distributions, Density Estimation, and Estimators

Understanding the principles of generative models requires a basic knowledge of distributions. The reason is that—as already hinted at in the previous section—the "fraction of the world" is in fact something that can be thought of as a distribution in a parameter space. If you were to describe a part of the world in a computer-interpretable way, you would define descriptive parameters. To describe persons, you could characterize them by simple measures like age, height, weight, hair and eye color, and many more. You could add blood pressure, heart rate, muscle mass, maximum strength, and more, and even a whole-genome sequencing result might be a parameter. Each of the parameters individually can be collected for the world population, and you will obtain a picture of how this parameter is "distributed" worldwide. In addition, parameters will be in relation with each other, for example, age and maximum strength. Countless such relationships exist, of which the majority are and probably will remain unknown. Those interrelationships are called a joint distribution. Would you know the joint distribution, you could "create" a plausible parameter combination of a nonexisting human. Let us formalize these thoughts now.

2.1.1 Distributions

A distribution describes the frequency of particular observations when watching a random process. Plotting the number of occurrences over an axis of all possible observations creates a histogram. If the possible observations can be arranged on a continuous scale, one can see that observations cluster in certain areas, and we say that they create a "density" or are "dense" there. Hence, when trying to describe where densities are in parameter space, this is associated with the desire to reproduce or sample from distributions, like we want to do it to generate instances from a domain. Before being able to reproduce the function that generates observations, estimating where the dense areas are is required. This will in the most general sense be called density estimation.

Sometimes, the shape of the distribution follows an analytical formula, for example, the normal distribution. If such a closed-form description of the distribution can be given, for instance, the normal distribution, this distribution generalizes the shape of the histogram of observations and makes it possible to produce new observations very easily, by simply sampling from the distribution. When our observations follow a normal distribution, we mean that we expect to observe instances more frequently around the mean of the normal distribution than toward the tails. In addition, the standard deviation quantifies how much more likely observations close to the mean are compared to observations in the tails. We describe our observations with a parametric description of the observed density.

In the remainder of this section, rather than providing a rigorous mathematical definition and description of the mathematics of distributions and (probability) density estimation, we will introduce the basic concepts and terminology in an intuitive way (also compare Box 1). Readers with the wish for a more in-depth treatment can find tutoring material in the references [2–6].

Box 1: Probability Distributions: Terminology

Several common terms regarding distributions have intuitive interpretations which are given in the following. Let a be an event from the probability distribution A, written as $a \sim A$, and $b \sim B$ an event from another probability distribution.

In a medical example, A might be the distribution of possible neurological diseases and B the distribution of all possible variations of smoking behavior.

Conditional Probability $P(A|B)$ The conditional probability of a certain $a \sim A$, for example, a stroke, might depend on the concrete smoking history of a person,

(continued)

Box 1 (continued)

	described by $b \sim B$. The conditional probability is written as $p(a\|b)$ for the concrete instances or $P(A\|B)$ if talking about the entire probability distributions A and B.
Joint Probability $P(A, B)$	The probability of seeing instantiations of A and B together is termed the joint probability. Notably, if expanded, this will lead to a large table of probabilities, joining each possible $a \sim A$ (e.g., stroke, dementia, Parkinson's disease, etc.) with each possible $b \sim B$ (casual smoker, frequent smoker, nonsmoker, etc.).
Marginal Probability	The marginal probabilities of A and B (denoted, respectively, $P(A)$ and $P(B)$) are the probabilities of each possible outcome across (and independent of) all of the possible outcomes of the other distribution. For example, it is the probability of seeing nonsmokers across all neurological diseases or seeing a specific disease regardless of smoking status. It is said to be the probability of one distribution marginalized over the other probability distributions.

2.1.2 Density Estimation

We assume in the following that our observations have been produced by a function or process that is not known to us and that cannot be guessed from an arrangement of the observations. In a practical example, the images from a CT or MRI scanner are produced by such a function. Notably, the concern is less about the intractability of the imaging physics but about the appearance of the human body. The imaging physics might be modeled analytically up to a certain error. But the outer shape and inner structure of the

human body and its organs depend on a large amount of mutually influencing factors. Some of these factors are known and can even be modeled, but many are not. In particular, the interdependence of factors must be assumed to be intractable. What we can accumulate is measured data providing information about the body, its shape, and its function. While many measurement instruments exist in medicine, for this chapter, we will be concerned with images as our observations. In the following thought experiment, we will explore a naïve way to model the distribution and try to generate images.

The first step is to examine the gray value distribution or, in other words, estimate the density of values. The most basic way for estimating a density is plotting a histogram. Let the value on the x axis be the image gray value of the medical image in question (in CT expressed in Hounsfield units (HU) and in arbitrary units for MRI). Two plots show histograms of a head MRI (Fig. 4) and an abdominal CT (Fig. 5). While the brain MRI suggests three or four major "bumps" of the histogram at about values 25, 450, and 600, the abdominal CT doesn't lend itself to such a description.

In the next step, we want to describe the histograms through analytical functions, to make them amenable for computational

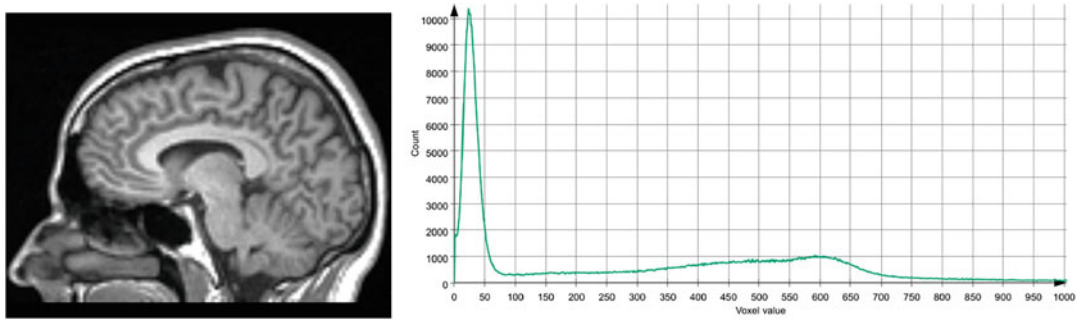

Fig. 4 Brain MRI (left) and histogram of gray values for one slice of a brain MRI

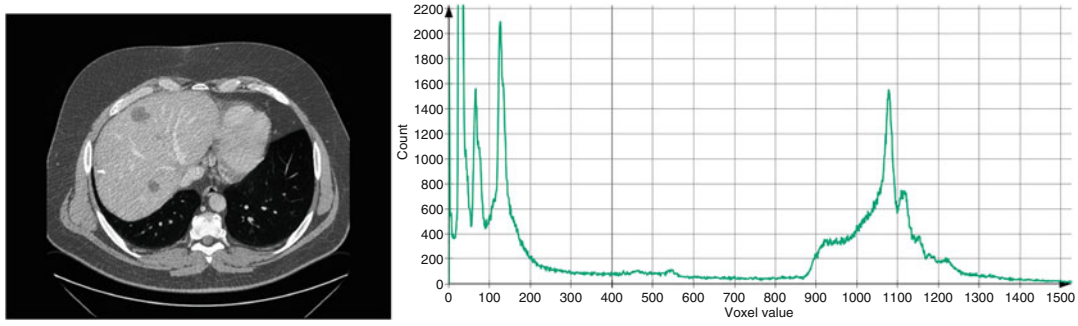

Fig. 5 Abdominal CT (left) and histogram of gray values for one slice of an abdominal CT

ends. This means we will aim to estimate an analytical description of the observations.

Expectation maximization (EM; *see* Box 2) is an algorithm suitable for this task. EM enables us to perform maximum likelihood estimation in the presence of unobserved ("latent) variables and incomplete data—this being the default assumption when dealing with real data. Maximum likelihood estimation (MLE) is the process of finding parameters of a parametric distribution to most accurately match the distribution to the observations. In MLE, this is achieved by adapting the parameters steered by an error metric that indicates the closeness of the fit; in short, a parameter optimization algorithm.

Box 2: Expectation Maximization—Example

Focusing on our density estimate of the MRI data, we want to use expectation maximization (EM) to optimize the parameters of a fixed number of Gaussian functions adding up to the closest possible fit to the empirical shape of the histogram.

In our data, we observe "bumps" of the histogram. We can by image analysis determine that certain organs imaged by MRI lead to certain bumps in the histogram, since they are of different material and create different signal intensities. This, however, is unknown to EM—the so-called "latent" variables.

The EM algorithm has two parts, the expectation step and the maximization step. They can, with quite far-reaching omission of details, be sketched as follows:

Expectation takes each point (or a number of sampled points) of the distribution and *estimates the expectation* to which of the parameterized distribution to assign it to. Figuring out this assignment is the step of dealing with the "latent" variable of the observations.

Maximization iterates over all parameterized distributions and adjusts their parameters to match the assigned points as well as possible.

This process is iterated until a fitting error cannot be improved anymore.

A short introductory treatment of EM with examples and applications is presented in [7]. The standard reference for the algorithm is [8].

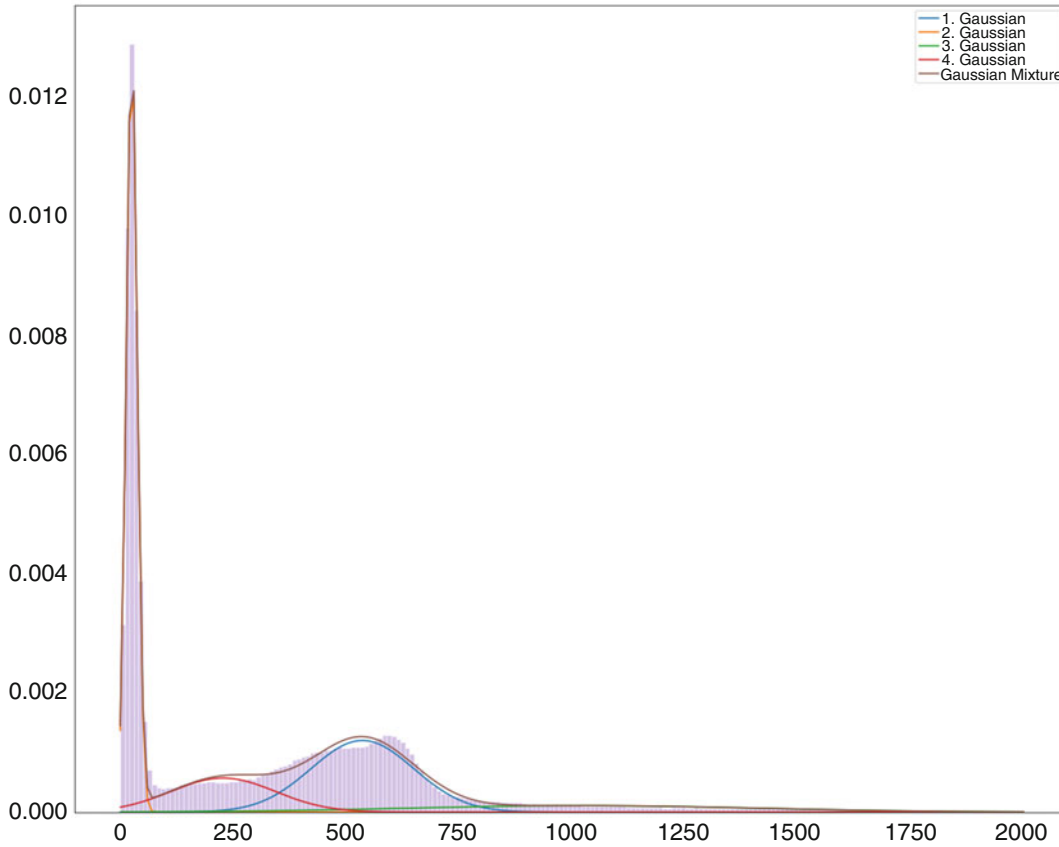

Fig. 6 A Gaussian mixture model (GMM) of four Gaussians was fit to the brain MRI data we have visualized as a histogram in Fig. 4

In Fig. 6, a mixture of four Gaussian distributions has been fit to the brain MRI voxel value data seen before.

It is tempting to model even more complex observations by mixing simple analytical distributions (e.g., Gaussian mixture models (GMMs)), but in general this will be intractable for two reasons. Firstly, realistic joint distributions will have an abundance of mixed maxima and therefore require a vast number of basic distributions to fit. Even basic normal distributions in high-dimensional parameter spaces are no longer functions with two parameters (μ, σ), but with a vector of means and a covariance matrix. Secondly, it is no longer trivial to sample from such high-dimensional joint distributions, and while some methods, among others Markov chain Monte Carlo methods, allow to sample from them, such numerical approaches are of such high computational complexity that it makes their use difficult in the context of deep neural network parameter estimation.

We will learn about alternatives. In principle, there are different approaches for density (distribution) estimation, direct distribution estimation, distribution approximation, or even more indirectly, by

using a simple surrogate distribution that is made to resemble the unknown distribution as good as possible through a mapping function. We will see this in the further elaboration of generative modeling approaches.

2.1.3 Estimators and the Expected Value

Assume we have found suitable mean values and standard deviations for three normal distributions that together approximate the shape of the MRI data density estimate to our satisfaction. Such a combination of normal (Gaussian) distributions is called a Gaussian mixture model (GMM), and sampling from such a GMM is straightforward. We are thus able to sample single pixels in any number, and over time we will sample them such that their density estimate or histogram will look similar to the one we started with.

However, if we want to generate a brain MRI image using a sampling process from our closed-form GMM representation of the distribution, we will notice that a very important notion wasn't respected in our approach. We start with one slice of 512×512 voxels and therefore randomly draw the required number of voxel values from the distribution. However, this will not yield an image that resembles one slice of a brain MRI, but will almost look like random noise, because we did not model the spatial relation of the gray values with respect to each other. Since the majority of voxels of a brain MRI are not independent of each other, drawing one new voxel from the distribution needs to depend on the spatial locations and gray values of all voxels drawn before. Neighboring voxels will have a higher likelihood of similar gray values than voxels far apart from each other, for example. More crucially, underneath the interdependence lies the image generation process: the image values observed in a real brain MRI stem from actual tissue—and this is what defines their interdependence. This means the anatomy of the brain indirectly reflects itself in the rules describing the dependency of gray values of one another.

For the modeling process, this implies that we cannot argue about single-voxel values and their likelihood, but we need to approach the generative process differently. One idea for a generative process has been implied in the above description already: pick a random location of the to-be-generated image and predict the gray value depending on all existing voxel values. Implemented with the method of mixture models, this results in unfathomably many distributions to be estimated, as for each possible "next voxel" location, any possible combination of already existing voxel numbers and positions needs to be considered. We will see in Subheading 5.1 on diffusion models how this general approach to image generation can still be made to work.

A different sequential approach to image generation has also been attempted, in which pixels are generated in a defined order, starting at the top left and scanning the image row by row across the columns. Again, the knowledge about the already produced

pixels is memorized and used to predict the next voxel. This has been dubbed the PixelRNN (Pixel Recurrent Neural Network), which lends its general idea from text processing networks [9].

Lastly, a direct approach to image generation could be formulated by representing or approximating the full joint distribution of all voxels in one distribution that is tangible and to sample all voxels *at once* from this. The full joint distribution in this approach remains implicit, and we use a surrogate. This will actually be the approach implemented in GANs, though not in a naïve way.

Running the numbers of what a likelihood-based naïve approach implies, the difficulties of making it work will become obvious. Consider an MRI image as the joint distribution of 512×512 voxels (one slice of our brain MRI), where we approximated the gray value distribution of one voxel with a GMM with six parameters. This results in a joint distribution of $512 \times 512 \times 6 = 1,572,864$ parameters. Conceptually, this representation therefore spans a 1,572,864-dimensional space, in which every one brain MRI slice will be one data point. Referring back to the histograms of CT and MRI images in the figures above, we have seen continuous lines with densities because we have collected all voxels of an entire medical image, which are many million. Still, we only covered one single dimension out of the roughly 1.5 million. Searching for the density in the 1,572,864-dimensional MRI-slice-space that is given by all collected brain MRI slices is the difficult task any generative algorithm has to solve. In this vastly large space, the brain MRI slices "live" in a very tiny region that is extremely hard to find. We say the images occupy a low-dimensional manifold within the high-dimensional space.

Consider the maximum likelihood formulation

$$\hat{\theta} = \arg\max_{\theta} \mathbb{E}_{x \sim P_{\text{data}}} \log Q_{\theta}(x|\theta) \tag{1}$$

where P_{data} is the unknown data distribution and Q_θ the distribution generated by the model which is parameterized by θ. θ can, for example, be the weights and biases of a deep neural network.[1] In other words, the result of maximum likelihood estimation is parameters $\hat{\theta}$ so that the product of two terms, out of which only the second depends on the choice of θ, is maximal. The first term is the expectation of x with regard to the real data distribution. The second term is the (log of) the conditional probability (likelihood) of seeing the example x given the choice of θ under the model Q_θ. Hence, maximizing the likelihood function means maximizing the probability that x is seen in Q_θ, which will be the case when Q matches P as closely as possible given the parametric form of Q.

[1] We will use θ when referring to parameters of models in general but designate parameters of neural networks with w in accordance with literature.

The maximum likelihood mechanism is very nicely illustrated in [10]. Here, it is also visually shown how finding the maximum likelihood estimate of parameters of the distribution can be done by working with partial derivatives of the likelihood function with respect to μ and σ^2 and seeking their extrema. The partial derivatives are called the score function and will make a reappearance when we discuss score-based and diffusion models later in Subheading 5.1 on advanced generative models.

2.1.4 Sampling from Distributions

When a distribution is a model of how observed values occur, then sampling from this distribution is the process of generating random new values that could have been observed, with a probability similar to the probability to observe this value in reality. There are two basic approaches to sampling from distributions: generating a random number from the uniform distribution (this is what a random number generator is always doing underneath) and feeding this number through the inverse cumulative density function (iCDF) of the distribution, which is the function that integrates the probability density function (PDF) of the distribution. This can only be achieved if the CDF is given in closed form. If it is not, the second approach to sampling can be used, which is called acceptance (or rejection) sampling. With f being the PDF, two random numbers x and y are drawn from the uniform distribution. The random x is accepted, if $f(x) > y$, and rejected otherwise.

Our use case, as we have seen, involves not only high-dimensional (multivariate) distributions but even more their joints, and they are not given in closed form. In such scenarios, sampling can be done still, using Markov chain Monte Carlo (MCMC) sampling, which is a framework using rejection sampling with added mechanisms to increase efficiency. While MCMC has favorable theoretic properties, it is still computationally very demanding for complex joint distributions, which leads to important difficulties in the context of sampling from distributions we are facing in the domain of image analysis and generation.

We are therefore at this point facing two problems: we can hardly hope to be able to estimate the density, and even if we could, we could practically not sample from it.

3 Generative Adversarial Networks

3.1 Generative vs. Discriminative Models

To emphasize the difficulty that generative models are facing, compare them to discriminative models. Discriminative models solve tasks like classification, detection, and segmentation, to name some of the most prominent examples. How classification models are in the class of discriminative models is obvious: discriminating examples is exactly classifying them. Detection models are also discriminative models, though in a broader sense, in that they classify the

detection proposals into accepted object detections or rejected proposals, and even the bounding box estimation, which is often solved through bounding box regression, typically involves the discriminative prediction of template boxes. Segmentation, on the other hand, for example, using a U-Net, is only the extension of classic discriminative approaches into a fast framework that avoids pixel-wise inference through the model. It is common to all these models that they yield output corresponding to their input, in the sense that they extract information from the input image (e.g., an organ segmentation, a classification, or even a textual description of the image content) or infer additional knowledge about it (e.g., a volume measurement or an assessment or prediction of a treatment success given the appearance of the image).

Generative models are fundamentally different, in that they generate output potentially without any concrete input, out of randomness. Still, they are supposed to generate output that conforms to certain criteria. In the most general form and intuitive formulation, their output should "look natural." We want to further formalize the difference between the models in the following by using the perspective of distributions again. Figure 7 shows how discriminative and generative models have to construct differently complex boundaries in the representation space of the domain to accomplish their tasks.

Discriminative models take one example and map it to a label—e.g., the class. This is also true for segmentation models: they do this for each image voxel. The conceptual process is that the model has to estimate the probabilities that the example (or the voxel) comes from the distribution of the different available classes. The distributions of all possible appearances of objects of all classes do

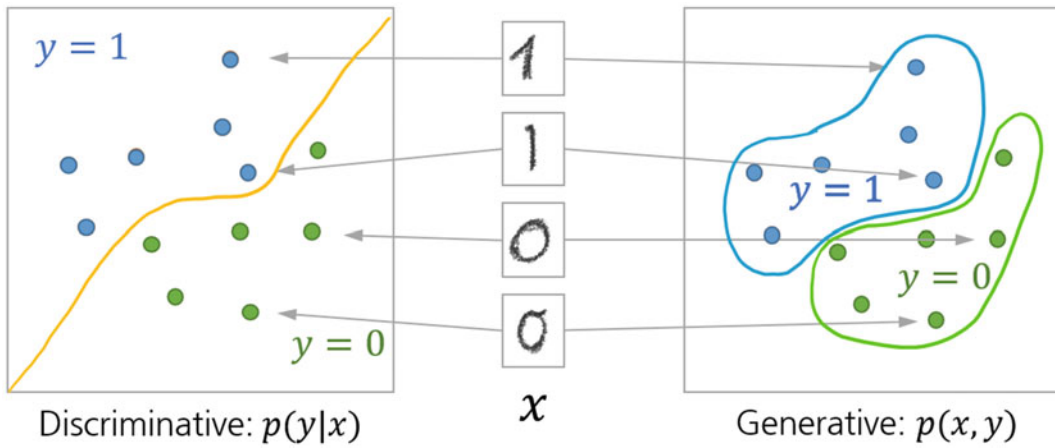

Fig. 7 The discriminative task compared to the generative task. Discriminative models only need to find the separating line between classes, while generative models need to delineate the part of space covering the classes (figure inspired by: https:/developers.google.com/machine-learning/gan/generative)

not need to be modeled analytically for this to be successful. It is only important to know them locally—for example, it is sufficient to delineate their borders or overlaps with other distributions of other classes, but not all boundaries are important.

Generative models, on the other hand, are tasked to produce an example that is within a desired distribution. For this to work, the network has to learn the complete shape of this distribution. This is immensely complex, since all domains of practical importance in medical imaging are extremely high-dimensional and the distributions defining examples of interest within these domains are very small and hard to find. Also, they are neither analytically given nor normally distributed in their multidimensional space. But they have as many parameters as the output image of interest has voxels.

As already remarked, different other approaches were devised to generate output before GANs entered the scene. Among the trainable ones, approaches comprised (restricted) Boltzmann machines, deep belief networks, or generative stochastic networks, variational autoencoders, and others. Some of them involved feedback loops in the inference process (the prediction of a generated example) and were therefore unstable to train using backpropagation.

This was solved with the adversarial net framework proposed in 2014 by Goodfellow et al. [1]. They tried to solve the downsides like computational intractability or instability of such previous generative models by introducing the adversarial training framework.

To understand how GANs relate to one of the closest predecessors, the variational autoencoder, we will review their basic layout next. We will learn how elegantly the GAN paradigm turns the previously unsupervised approach to generative modeling into a supervised one, with the benefit of much more control over the training process.

3.2 Before GANs: Variational Autoencoders

Generative adversarial networks (GANs) haven't been the first or only attempt at generating realistically looking images (or any type of output, generally speaking). Apart from GANs, a related neural network-based approach to generative modeling is the variational autoencoder, which will be treated in more details below. Among other generative models with different approaches are as follows:

Flow-based models This category of generative models attempt to model the data-generating distribution explicitly through an iterative process known as the normalizing flow [11], in which through repeated changes of variables a sequence of differentiable basis distributions is stacked to model the target distribution. The process is fully invertible, yielding models with desirable properties, since an

analytical solution to the data-generating distribution allows to directly estimate densities to predict the likelihood of future events, impute missing data points, and of course generate new samples. Flow-based models are computation-intensive. They can be categorized as a method that returns an explicit, tractable density. Another method in this category is, for example, the PixelRNN [9] or the PixelCNN [12] which also serves for conditional image generation. RealNVP [13] also uses a chain of invertible functions.

Boltzmann machines work fundamentally differently. They also return explicit densities but this time only approximate the true target distribution. In this regard, they are similar to variational autoencoders, though their method is based on Markov chains, and not a variational approach. Deep Boltzmann machines have been proposed already in 2009, uniting a Markov chain-based loss component with a maximum likelihood-based component and showing good results on, at that time, highly complex datasets. [14] Boltzmann machines are very attractive but harder to train and use than other comparably powerful alternatives that exist today. This might change with future research, however.

Variational autoencoders (VAE) are a follow-up development of plain autoencoders, autoregressive models that in their essence try to reconstruct their input after transforming it, usually into a low-dimensional representation (*see* Fig. 8). This low-dimensional

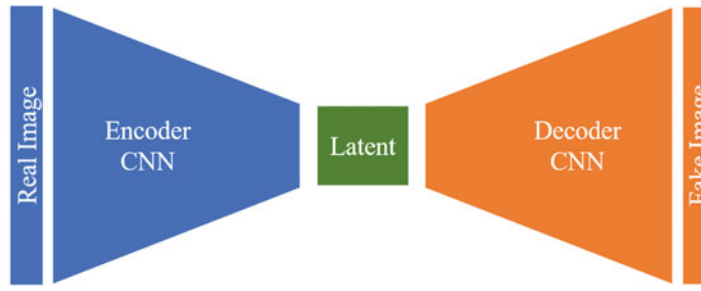

Fig. 8 Schematic of an autoencoder network. The encoder, for images, for example, a CNN with a number of convolutional and pooling layers, condenses the defining information of the input image into the variables of the latent space. The decoder, again convolutions, but this time with upsampling layers, recreates a representation in image space. Input and output images are compared in the loss function, which drives the gradient descent

representation is often termed the "latent space," implying that here hidden traits of the data-generating process are coded, which are essential to the reconstruction process. This is very akin to the latent variables estimated by EM. In the autoencoder, the encoder will learn to code its input in terms of these latent variables, while the decoder will learn to represent them again in the source domain. In the following, we will be discussing the application to images though, in principle, both autoencoders and their variational variant are general mechanisms working for any domain.

We will later be interested in a behind-the-scene understanding of their modeling approach, which will be related to the employed loss function. We will then look at VAEs more extensively from the same vantage point: to understand their loss function—which is closest to the loss formulation of early GANs, the Kullback-Leibler divergence or KL divergence, D_{KL}.

With this tool in hand, we will examine how to optimize (train) a network with regard to KL divergence as the loss and understand key problems with this particular loss function. This will lead us to the motivation for a more powerful alternative.

3.2.1 From AE to VAE

VAEs are an interesting subject to study to emphasize the limits a loss function like KL divergence may place on a model. We will begin with a recourse to plain autoencoders to introduce the concept of learning a latent representation. We will then proceed to modify the autoencoder into a variational formulation which brings about the switch to a divergence measure as a loss function. From these grounds, we will then show how GANs again modified the loss function to succeed in high-quality image generation.

Figure 8 shows the schematic of a plain autoencoder (AE). As indicated in the sketch, input and output are of potentially very high dimensionality, like images. In between the encoder and decoder networks lies a "bottleneck" representation, which is, for example, a convolutional layer of orders of magnitude lower dimensionality (represented, for example, by a convolutional layer with only a few channels or a dense layer with a given low number of weights), which forces the network to find an encoding that preserves all information required for reconstruction.

A typical loss function to use when training the autoencoder is, for example, cross entropy, which is applicable for sigmoid activation functions, or simply the mean squared error (MSE). Any loss shall essentially force the AE to learn the identity function between input and output.

Let us introduce the notation for this. Let X be the input image tensor and X' the output image tensor. With f_w being the encoder function given as a neural network parameterized by weights and biases w and g_v the decoder function parameterized by v, the loss hence works to make $X = X' = g_v(f_w(X))$.

In a *variational* autoencoder,[2] things work differently. Autoencoders like before use a fixed (deterministic) latent code to map the input to, while variational autoencoders will replace this with a distribution. We can call this distribution p_w, indicating the parameterization by w. It is crucial to understand that a choice was made here that imposes conditions on the latent code. It is meant to represent the input data in a variational way: in a way following Bayes' laws. Our mapping of the input image tensor X to the latent variable $\mathbf{z}$ is by this choice defined by

- The prior probability $p_w(\mathbf{z})$
- The likelihood (conditional probability) $p_w(X|\mathbf{z})$
- The posterior probability $p_w(\mathbf{z}|X)$

Therefore, once we have obtained the correct parameters $\hat{w}$ by training the VAE, we can produce a new output X' by sampling a $\mathbf{z}^{(i)}$ from the prior probability $p_{\hat{w}}(\mathbf{z})$ and then generate the example from the conditional probability through $X^{(i)} = p_{\hat{w}}(X|\mathbf{z} = \mathbf{z}^{(i)})$.

Obtaining the optimal parameters, however, isn't possible directly. The searched optimal parameters are those that maximize the probability that the generated example X' looks real. This probability can be rewritten as the aggregated conditional probabilities:

$$p_w(X^{(i)}) = \int p_w(X^{(i)}|\mathbf{z}) p_w(\mathbf{z}) d\mathbf{z}.$$

This, however, does not make the search any easier since we need to enumerate and sum up all $\mathbf{z}$. Therefore, an approximation is made through a surrogate distribution, parameterized by another set of parameters, q_v. Weng [15] shows in her explanation of the VAE the graphical model highlighting how q_v is a stand-in for the unknown searched p_w (*see* Fig. 9).

The reason to introduce this surrogate distribution actually comes from our wish to train neural networks for the decoding/encoding functions, and this requires us to back-propagate through the random variable, $\mathbf{z}$, which of course cannot be done. Instead, if we have control over the distribution, we can select it such that the reparameterization trick can be employed. We define q_v to be a multivariate Gaussian distribution with means and a covariance matrix that can be learned and a stochastic element multiplied to the covariance matrix for sampling [15, 16]. With this, we can back-propagate through the sampling process.

[2] Though variational autoencoders are in general not necessarily neural networks, in our context, we restrict ourselves to this implementation and stick to the notation with parameters w and v, where in many publications they are denoted θ and ϕ.

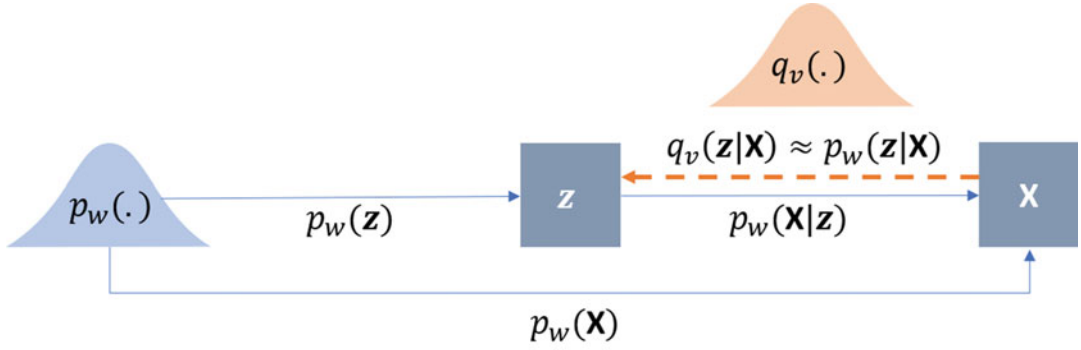

Fig. 9 The graphical model of the variational autoencoder. In a VAE, the *variational decoder* is $p_w(X|\mathbf{z})$, while the *variational encoder* is $q_v(\mathbf{z}|X)$ (Figure after [15])

At this point, the two distributions need to be made to match: q_v should be as similar to p_w as possible. Measuring their similarity can be done in a variety of ways, of which Kulback-Leibler divergence (KL divergence or KLD) is one.

3.2.2 KL Divergence

A divergence can be thought of as an asymmetric distance function between two probability distributions, P and Q, measuring the similarity between them. It is a statistical distance which is not symmetric, which means it will not yield the same value if measured from P to Q or the other way around:

$$D_{\mathrm{KL}}(P\|Q) \neq D_{\mathrm{KL}}(Q\|P)$$

This can be seen when looking at the definition of KL divergence:

$$D_{\mathrm{KL}}(P\|Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)} \tag{2}$$

Sometimes, the measure D_{KL} is also called the relative entropy or information gain of P over Q, which also indicates the asymmetry.

To give the two distributions more meaning, let us associate them with a use case. P is usually the probability distribution of the example data, which can be our real images we wish to model, and is assumed to be unknown and high-dimensional. Q, on the other hand, is the modeled distribution, for example, parameterized by θ, similar to Eq. 1. Hence, Q is the distribution we can play with (in our case, optimize its parameters) to make them more similar to P. This means Q will get more informative with respect to the true P when we approach the optimal parameters.

Box 3: Example: Calculating D_{KL}

When comparing the two distributions given in Fig. 10, the calculation of the Kullback-Leibler divergence, D_{KL}, can explicitly be given by reading off the y values of the nine elements (columns) from Fig. 11 and inserting them into Eq. 2.

The result of this calculation is for

$$\begin{aligned} D_{\mathrm{KL}}(P\|Q) &= \sum_x P(x) \log \frac{P(x)}{Q(x)} \\ &= 0.02 * \log \frac{.02}{.01} + 0.04 * \log \frac{.04}{.12} + \cdots + 0.02 * \log \frac{.02}{.022} \\ &= 0.004 - 0.01 + \cdots - 0.0002 \\ &= 0.0801 \end{aligned}$$

which we call "forward KL" as it calculates in the direction from the actual distribution P to the model distribution Q and for

$$\begin{aligned} D_{\mathrm{KL}}(Q\|P) &= \sum_x Q(x) \log \frac{Q(x)}{P(x)} \\ &= 0.01 * \log \frac{0.01}{0.02} + 0.12 * \log \frac{0.12}{0.04} + \cdots + 0.022 * \log \frac{0.022}{0.02} \\ &= -0.002 - 0.05 + \cdots + 0.0002 \\ &= 0.0899 \end{aligned}$$

which we call "reverse KL."

Note that in the example in Box 3, there is both a $P(X = x_i)$ and $Q(X = x_i)$ for each $i \in \{0, 1, \ldots, 8\}$. This is crucial for KL divergence to work as a loss function.

3.2.3 Optimizing the KL Divergence

Examine what happens in forward and reverse KL if this condition is not satisfied for some i. If in forward KL P has values everywhere but Q has not (or extremely small values), the quotient in the log

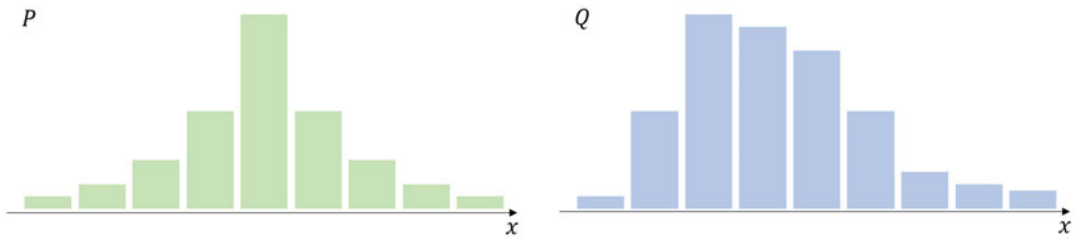

Fig. 10 Two distributions P and Q, here scaled to identical height

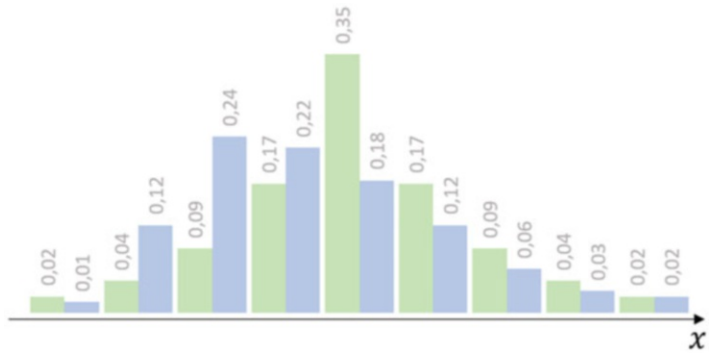

Fig. 11 The distributions P and Q, scaled to unit density, with added labels

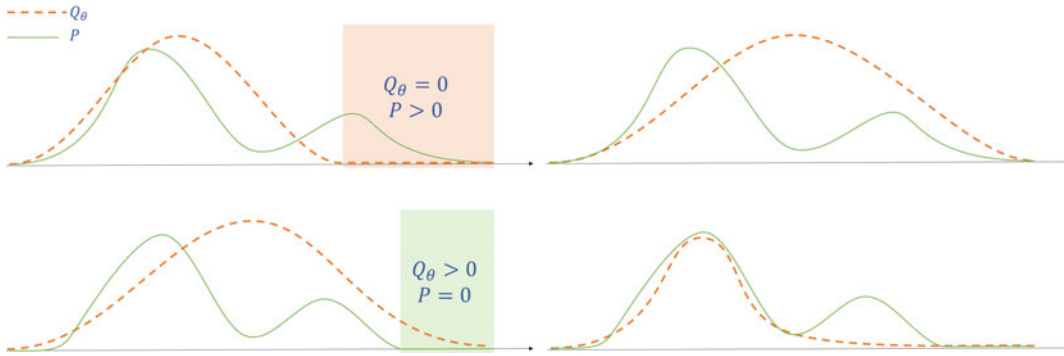

Fig. 12 The distributions P (solid) and Q_θ (dashed), in the initial configuration and after minimizing reverse KL $D_{\mathrm{KL}}(Q_\theta|P)$. This time, in the initial configuration, Q_θ has values greater than 0 where P has not (marked with green shading)

function will tend to infinity by means of the division by almost zero, and the term will be very large.

In Fig. 12, we assume Q_θ to be a unimodal normal distribution, i.e., a Gaussian, while P is any empirical distribution. In the left plots of the figure, we show a situation before minimizing the forward/reverse KL divergence between P and Q_θ, in the right plots, the resulting shape of the Gaussian after minimization.

When in the minimization of forward KL $D_{\mathrm{KL}}(P|Q_\theta)$ Q_θ is zero where P has values greater zero, KL goes to infinity in these regions (marked area in the start configuration of the top row in Fig. 12), since the denominator in the log function goes to zero. This, in turn, drives the parameters of Q_θ to broaden the Gaussian to cover these areas, thereby removing the large loss contributions. This is known as the *mean-seeking* behavior of forward KL.

Conversely, in reverse KL (bottom row in Fig. 12), in the marked areas of the initial configuration, P is zero in regions where Q_θ has values greater than zero. This yields high-loss

contributions from the log denominator, in this case driving the Gaussian to remove these areas from Q_θ. Since we assumed a unimodal Gaussian Q, the minimization will focus on the largest mode of the unknown P. This is known as the *mode-seeking* behavior of reverse KL.

Forward KL tends to overestimate the target distribution, which is exaggerated in the right plot in Fig. 12. In contrast, reverse KL tends to underestimate the target distribution, for example, by dropping some of its modes. Since underestimation is the more desirable property in practical settings, reverse KL is the loss function of choice, for example, in variational autoencoders. The downside is that as soon as target distribution P and model distribution Q_θ have no overlap, KL divergence evaluates to infinity and is therefore uninformative. One countermeasure to take is to add noise to Q_θ, so that there is guaranteed overlap. This noise, however, is not desirable in the model distribution Q_θ since it disturbs the generated output.

Another way to remedy the problem of KL going to infinity is to adjust the calculation of the divergence, which is done in Jensen-Shannon divergence (JS divergence, D_{JS}) defined as

$$D_{\mathrm{JS}} = \frac{1}{2}\left(D_{\mathrm{KL}}(P\|M) + D_{\mathrm{KL}}(Q_\theta\|M)\right), \tag{3}$$

where $M = \frac{P+Q_\theta}{2}$. In the case of nonoverlapping P and Q_θ, this evaluates to constant $\log 2$, which is still not providing information about the closeness but is computationally much friendlier and does not require the addition of a noise term to achieve numerical stability.

3.2.4 The Limits of VAE

In the VAE, reverse KL is used. Our optimization goal is maximizing the likelihood to produce realistic looking examples—ones with a high $p_w(x)$. Simultaneously, we want to minimize the difference between the real and estimated posterior distributions q_v and p_w. This can only be achieved through a reformulation of reverse KL [15]. After some rearranging of reverse KL, the loss of the variational autoencoder becomes

$$\begin{aligned} L_{\mathrm{VAE}}(w, v) &= -\log p_w(X) + D_{\mathrm{KL}}(q_v(\mathbf{z}|X)\|p_w(\mathbf{z}|X)) \\ &= -\mathbb{E}_{\mathbf{z}\sim q_v(\mathbf{z}|X)} \log p_w(X|\mathbf{z}) + D_{\mathrm{KL}}(q_v(\mathbf{z}|X)\|p_w(\mathbf{z})) \end{aligned} \tag{4}$$

$\hat{w}$ and $\hat{v}$ are the parameters maximizing the loss.

We have seen how mode-seeking reverse KL divergence limits the generative capacity of variational autoencoders through the potential underrepresentation of all modes of the original distribution.

KL divergence and minimizing the ELBO also have a second fundamental downside: there is no way to find out how close our solution is to the obtainable optimum. We measure the similarity to the target distribution up to the KL divergence, but since the true $p_{\hat{w}}(.)$ is unknown, the stopping criterion in the optimization has to be set by another metric, e.g., to a maximum number of iterations or corresponding to an improvement of the loss below some ε.

The original presentation of the variational autoencoder was given as one example of the general framework called the autoencoding variational Bayes. This publication presented the above ideas in a thorough mathematical formulation, starting from a directed graphical model that poses the abstract problem. The authors also develop the seminal "reparameterization trick" to make the loss formulation differentiable and with this to make the search for the autoencoder parameters amenable to gradient descent optimizers [16]. The details are beyond this introductory treatment.

3.3 The Fundamental GAN Approach

At the core of the adversarial training paradigm is the idea to create two players competing in a minimax game. In such games, both players have access to the same variables but have opposing goals, so that they will manipulate the variables in different directions.

Referring to Fig. 13, we can see the generative part in orange color, where random numbers are drawn from the latent space and, one by one, converted into a set of "fake images" by the generator

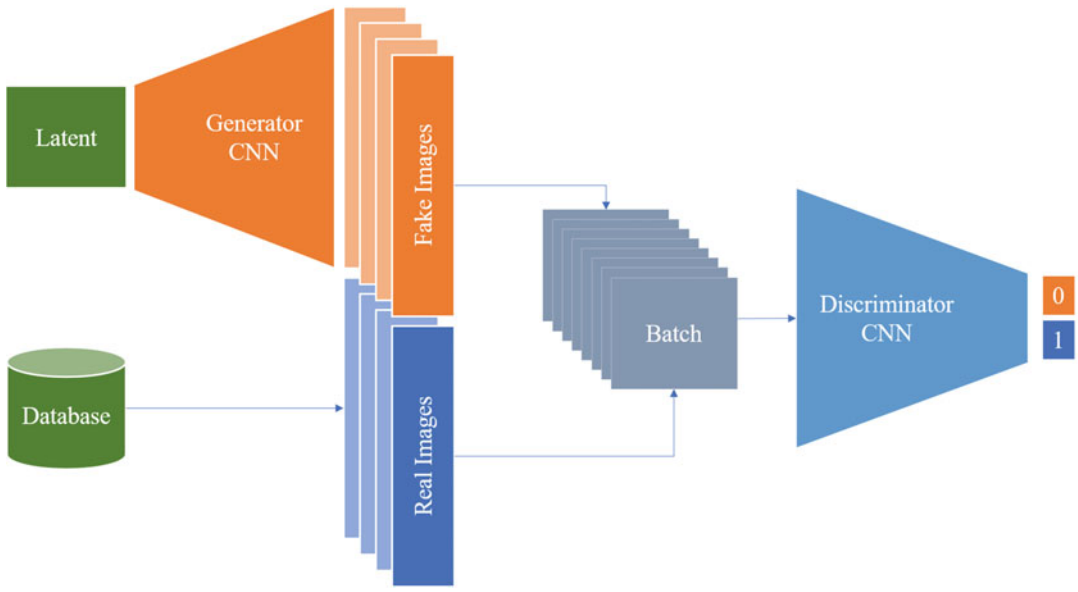

Fig. 13 Schematic of a GAN network. Generator (orange) creates fake images based on random numbers drawn from a latent space. These together with a random sample of real images are fed into the discriminator (blue, right). The discriminator looks at the batch of real/fake images and tries to assign the correct label ("0" for fake, "1" for real)

network, in the figure implemented by a CNN. Simultaneously, from a database of real images, a matching number of examples are randomly drawn. The real and fake images are composed into one batch of images which are fed into the discriminator. On the right side, the discriminator CNN is indicated in blue. It takes the batch of real and fake images and decides for each if it appears real (yielding a value close to "1") or fake ("0").

The error signal is computed from the number of correct assignments the discriminator can do on the batch of generated and real images. Both the generator and the discriminator can then update their parameters based on this same error signal. Crucially, the generator has the aim to *maximize* the error, since this signifies that it has successfully fooled the discriminator into taking the fake images for real, while the discriminator weights are updated to *minimize* the same error, indicating its success in telling true and fake examples apart. This is the core of the competitive game between generator and discriminator.

Let us introduce some abbreviations to designate GAN components. We will denote the generator and discriminator networks with G and D, respectively. The objective of GAN training is a game between generator and discriminator, where both affect a common loss function J, but in opposed directions. Formally, this can be written as

$$\min_G \max_D J(G, D),$$

with the GAN objective function

$$J(G, D) = \mathbb{E}_{x \sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z \sim p_G}[1 - \log D(G(z))] \tag{5}$$

D will attempt to maximize J by maximizing the probability to assign the correct labels to real and generated examples: this is the case if $D(x) = 1$, maximizing the first loss component, and if $D(G(z)) = 0$, maximizing the second loss component. The generator G, instead, will attempt to generate realistic examples that the discriminator labels with "1," which corresponds to a minimization of $\log(1 - D(G(z)))$.

3.4 Why Early GANs Were Hard to Train

GANs with this training objective implicitly use JS divergence for the loss, which can be seen by examining the GAN training objective. Consider the ideal discriminator D for a fixed generator. Its loss is minimal for the optimal discriminator given by [1]

$$\hat{D}(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)}. \tag{6}$$

Substituting $\hat{D}$ in Eq. 5 yields (without proof) the implicit use of the Jensen-Shannon divergence if the above training objective is employed:

$$J(G, \hat{D}) = 2D_{\mathrm{JS}}(p_{\mathrm{data}} \| p_G) - \log 4. \tag{7}$$

This theoretical result shows that a minimum in the GAN training can be found when the Jensen-Shannon divergence is zero. This is achieved for identical probability distributions p_{data} and p_G or, equivalently, when the generator perfectly matches the data distribution [17].

Unfortunately, it also shows that this loss is, like KL divergence, only helpful when target distribution (i.e., data distribution) and model distribution have overlapping support. Therefore, added noise can be required to approximate the target distribution. In addition, the training criterion saturates if the discriminator in the early phase of training perfectly distinguishes between fake and real examples. The generator will therefore no longer obtain a helpful gradient to update its weights. An approach thought to prevent this was proposed by Goodfellow et al. [1]. The generator loss was turned from the minimization problem into a maximization problem that has the same fixed point in the overall minimax game but prevents saturation: instead of minimizing $\log(1 - D(G(z)))$, one maximizes $\log(D(G(z)))$ [1].

3.5 Improving GANs

GAN training has quickly become notorious for the difficulties it posed upon the researchers attempting to apply the mechanism to real-world problems. We have qualitatively attributed a part of these problems to the inherently difficult task of density estimation and motivated the intuition that while fewer samples might suffice to learn a decision boundary in a discriminative task, many more examples are required to build a powerful generative model.

In the following, some more light shall be shed on the reasons why GAN training might fail. Typical GAN problems comprise the following:

Mode dropping is the phenomenon in forward KL caused by regions of the data distribution not being covered by the generator distribution, which implies large probabilities of samples coming from P_{data} and very small probabilities of originating from P_G. This drives forward KL toward infinity and punishes the generator for not covering the entire data distribution [18]. If all modes but one are dropped, one can call this mode collapse: the generator only generates examples from one mode of the distribution.

Poor convergence can be caused by a discriminator learning to distinguish real and fake examples very early—which is also very likely to happen throughout the GAN training. This is rooted in the

observation that by the generative process that projects from a low-dimensional latent space into the high-dimensional p_G, the samples in p_G are not close to each other but rather inhabit "islands" [18]. The discriminator can learn to find them and thereby differentiate between true and false samples easily, which causes the gradients driving generator optimization to vanish [17].

Poor sample quality despite a high log likelihood of the model is a consequence of the practical independence of sample quality and model log likelihood. Theis et al. [19] show that neither does a high log likelihood imply generated sample fidelity nor do visually pleasing samples imply a high log likelihood. Therefore, training a GAN with a loss function that effectively implements maximizing a log likelihood term is not an ideal choice—but exactly corresponds to KL minimization.

Unstable training is a consequence of reformulating the generator loss into maximizing $\log D(G(z))$. It can be shown [18] that this choice effectively makes the generator struggle between a reverse KL divergence favoring mode-seeking behavior and a negative JS divergence actually driving the generator into examples different from the real data distribution.

There have been many subsequent authors touching these topics, but already Arjovsky and Bottou [18] have shown best practices of how to overcome these problems.

Among the solutions proposed for GAN improvements are some that prevent the generator from producing only too similar samples in one batch, some that keep the discriminator insecure about the true labels of real and fake examples, and more, which Creswell et al. [17] have summarized in their GAN overview. A collection of best practices compiled from these sources is presented in Box 4. It is almost impossible to write a cookbook for successful, converging, stable GAN training. For almost every tip, there is a caveat or situation where it cannot be applied. The suggestions below therefore are to be taken with a grain of salt but have been used by many authors successfully.

Box 4: Best Practices for Stable GAN Training

General measures. GAN training is sensitive to hyperparameters, most importantly the learning rate. Mode collapse might already be mitigated by a lower learning rate. Also, different learning rates for generator and discriminator might help. Other typical measures are batch normalization (or instance normalization in case of small batch sizes; mind however that batch normalization can taint the randomness of latent vector sampling and in general should not be used in combination with certain GAN loss functions), use of transposed convolutions instead of parameter-free upsampling, and strided convolutions instead of down-sampling.

Feature matching. One typical observation is that neither discriminator nor generator converges. They play their "cat-and-mouse" game too effectively. The generator produces a good image, but the discriminator learns to figure it out, and the generator shifts to another good image, and so on.

A remedy for this is feature matching, where the ℓ_2 distance between the average feature vectors of real and fake examples is computed instead of a cross-entropy loss on the logits. Because per batch the feature vectors change slightly, this introduces randomness that helps to prevent discriminator overconfidence.

Minibatch discrimination. When the generator only produces very convincing but extremely similar images, this is an indication for mode collapse.

This can be counteracted by calculating a similarity metric between generated samples and penalizing the generator for too little variation. Minibatch discrimination is considered to be superior in performance to feature matching.

One-sided label smoothing. Deep classification models often suffer from overconfidence, focusing on only very few features to classify an image. If this happens in a GAN, the generator might figure this out and only produce the feature the discriminator uses to decide for a real example.

A simple measure to counteract this is to provide not a "1" as a label for the real images in the batch but a lower value. This way, the discriminator is penalized for overconfidence (when it returns a value close to "1").

Cost function selection. Several sources list possible GAN cost functions. Randomly trying them one by one might work, but often some of the above measures, in particular learning rate and hyperparameter tuning, might be more successful first steps.

Besides these methods, one area of discussion concerned the question if there is a need of balancing discriminator and generator learning and convergence at all. The argument was that a converged discriminator will as well yield a training signal to the generator as a non-converged discriminator. Practically, however, many authors described carefully designed update schedules, e.g., updating the generator once per a given number of discriminator updates.

Many more ideas exist: weight updating in the generator using an exponential moving average of previous weights to avoid "forgetting," different regularization and conditioning techniques, and injecting randomness into generator layers anew. Some we will encounter later, as they have proven to be useful in more recent GAN architectures.

Despite the recent advances in stabilizing GAN training, even the basic method described so far, with the improvements made in the seminal DCGAN publication [20], finds application until today, e.g., for the de novo generation of PET color images [21]. The usefulness of an approach as presented in their publication might be doubted, since the native PET data is obviously not colored. The authors use 2D histograms of the three-color channel combinations to compare true and fake examples. As we have discussed earlier, this is likely a poor metric since it does not allow insights into the high-dimension joint probability distribution underlying the data-generating process. Figure 14 shows an example comparison of some generated examples compared to original PET images.

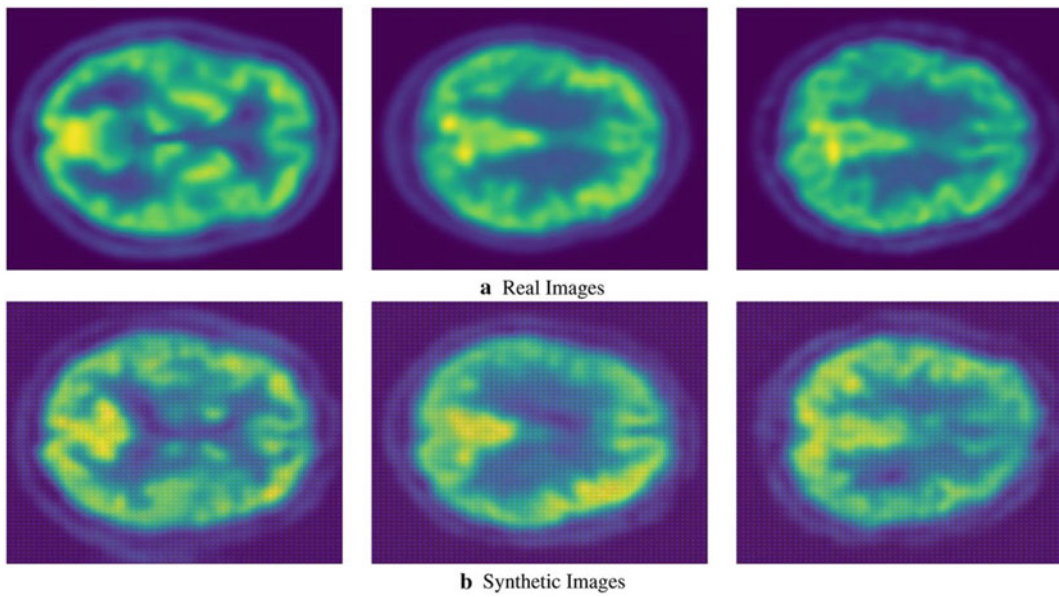

Fig. 14 PET images generated from random noise using a DCGAN architecture. Image taken from [21] (CC-BY4.0)

To address many of the GAN training dilemmas, Arjovsky and Bottou [18] have proposed to employ the Wasserstein distance as a replacement for KL or JS divergence already in their examination of the root causes of poor GAN training results and have later extended this into their widely anticipated approach we will focus on next [22, 23]. We will also see more involved and recent approaches to stabilize and speed up GAN training in later sections of this chapter (Subheading 4).

3.6 Wasserstein GANs

Wasserstein GANs were appealing to the deep learning and GAN scene very quickly after Arjovsky et al.'s [22] seminal publication because of a number of traits their inventors claimed they'd have. For one, Wasserstein GANs are based on the theoretical idea that the change of the loss function to the Wasserstein distance should lead to improved results. This combined with the reported benchmark performance would already justify attention. But Wasserstein GANs additionally were reported to train much more stably, because, as opposed to previous GANs, the discriminator would be trained to convergence in every iteration, instead of demanding a carefully and heuristically found update schedule for generator and discriminator. In addition, the loss was directly reported to correlate with visual quality of generated results, instead of being essentially meaningless in a minimax game.

Wasserstein GANs are therefore worth an in-depth treatment in the following sections.

3.6.1 The Wasserstein (Earthmover) Distance

The Wasserstein distance figuratively measures how, with an optimal transport plan, mass can be moved from one configuration to another configuration with minimal work. Think, for example, of heaps of earth. Figure 15 shows two heaps of earth, P and Q (discrete probability distributions), both containing the same amount of earth in total, but in different concrete states x and y out of all possible states.

Work is defined as the shovelfuls of earth times the distance it is moved. In the three rows of the figure, earth is moved (only within one of P or Q, not from one to the other), in order to make the configuration identical. First, one shovelful of earth is moved one pile further, which adds one to the Wasserstein distance. Then, two shovelfuls are moved three piles, adding six to the final Wasserstein distance of $D_{\mathrm{W}} = 7$.

Note that in an alternative plan, it would have been possible to move two shovelfuls of earth from p_4 to p_1 (costing six) and one from p_4 to p_3, which is the inverse transport plan of the above, executed on P, and leading to the same Wasserstein distance. The Wasserstein distance is in fact a distance, not a divergence, because it yields the same result regardless of the direction. Also note that

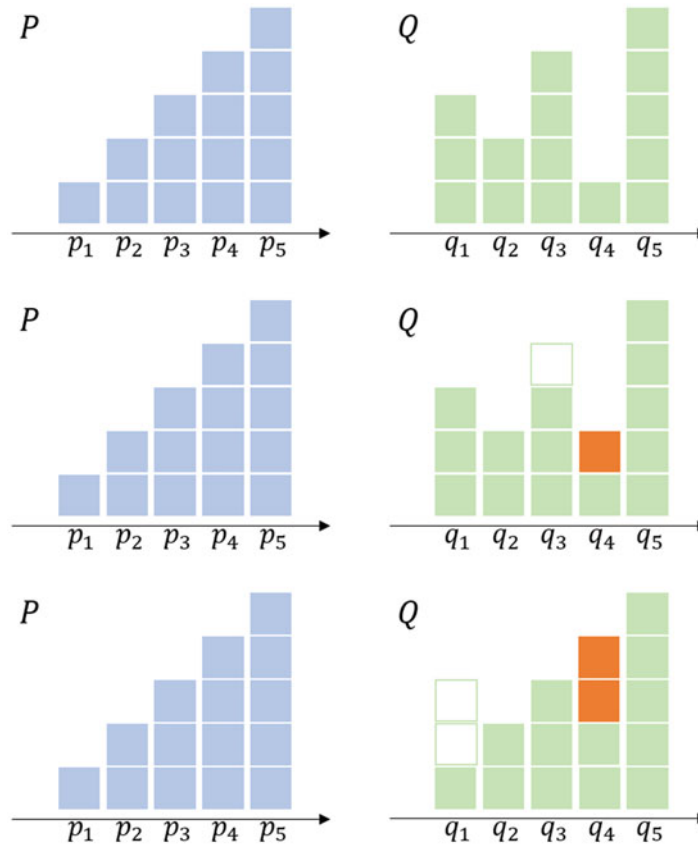

Fig. 15 One square is one shovel full of earth. Transporting the earth shovel-wise from pile to pile amasses performed work: the Wasserstein (earthmover) distance. The example shows a Wasserstein distance of $D_W = 7$

we implicitly assumed that P and Q share their support,[3] but that in case of disjunct support, only a constant term would have to be added, which grows with the distance between the support regions.

Many other transport plans are possible, and others can be equally cheap (or even cheaper—it is left to the reader to try this out). Transport plans need not modify only one of the stocks but can modify both to reach the optimal strategy to make them identical. Algorithmically, the optimal solution to the question of the optimal transport plan can be found by formulating it as a linear programming problem. However, enumerating all transport plans and computing the linear programming algorithm are intractable for larger and more complex "heaps of earth." Any nontrivial GAN will need to estimate transport of such complex "heaps," so they

[3] The support, graphically, is the region where the distribution is not equal to zero.

suffer this intractability problem. Consequently, in practice, a different approach must be taken, which we will sketch below.[4]

Formalizing the search for the optimal transport plan, we look at all possible joint distributions of our P and Q, forming the set of all possible transport plans, and denote this set $\Pi(P, Q)$, implying that for all $\gamma \in \Pi(P, Q)$, P and Q will be their marginal distributions.[5] This, in turn, means that by definition $\Sigma_x \gamma(x, y) = P(y)$ and $\Sigma_y \gamma(x, y) = Q(x)$.

For one concrete transport plan γ that works between a state x in P and a state y in Q, we are interested in the optimal transport plan $\gamma(x, y)$. Let $\|x - y\|$ be the Euclidian distance to shift earth between x and y, and then multiplying this with every value of γ (the amount of earth shifted) leads to

$$D_{\mathrm{W}}(P, Q) = \inf_{\gamma \in \Pi} \sum_{x,y} \|x - y\| \gamma(x, y),$$

which can be rewritten to obtain

$$D_{\mathrm{W}}(P, Q) = \inf_{\gamma \sim \Pi(P, Q)} \mathbb{E}_{(x,y)\sim\gamma} \|x - y\|. \tag{8}$$

It measures both the distance of two distributions with disjunct support and the difference between distributions with perfectly overlapping support because it includes both, the shifting of earth and the distance to move it.

Practically, though, this result cannot be used directly, since the Linear Programming problem scales exponentially with the number of dimensions of the domain of P and Q, which are high for images. To our disadvantage, we additionally need to differentiate the distance function if we want to use it for deep neural network training using backpropagation. However, we cannot obtain a derivative from our distance function in the given form, since, in the linear programming (LP) formulation, our optimized distribution (as well as the target distribution) end up as constraints, not parameters.

Fortunately, we are not interested in the transport plan γ itself, but only in the distance (of the optimal transport plan). We can therefore use the dual form of the LP problem, in which the constraints of the primal form become parameters. With some clever definitions, the problem can be cast into the dual form, finally yielding

[4] An extensive treatment of Wasserstein distance and optimal transport in general is given in the 1.000-page treatment of Villani's book [24], which is freely available for download.

[5] This section owes to the excellent blog post of Vincent Herrmann, at https://vincentherrmann.github.io/blog/wasserstein/. Also recommended is the treatment of the "Wasserstein GAN" paper by Alex Irpan at https://www.alexirpan.com/2017/02/22/wasserstein-gan.html. An introductory treatment of Wasserstein distance is also found in [25, 26].

$$D_{\mathrm{W}}(P,Q) = \|f\|_{L\leq 1} \sup\, \mathbb{E}_{x\sim P} f(x) - \mathbb{E}_{x\sim Q} f(x)$$

with a function f that has to adhere to a constraint called the 1-Lipschitz continuity constraint, which requires f to have a slope of at most magnitude 1 everywhere. f is the neural network, and more specifically for a GAN, the discriminator network. 1-Lipschitzness can be achieved trivially by clipping the weights to a very small interval around 0.

3.6.2 Implementing WGANs

To implement the distance as a loss function, we rewrite the last result again as

$$D_{\mathrm{W}}(P,Q) = \max_{w\in W} \mathbb{E}_{x\sim P}[D_w(x)] - \mathbb{E}_{z\sim Q}[D_w(G_w(z))]. \tag{9}$$

Note that in opposition to other GAN losses we have seen before, there is no logarithm anymore, because, this time, the "discriminator" is no longer a classification network that should learn to discriminate true and fake samples but rather serves as a "blank" helper function that during training learns to estimate the Wasserstein distance between the sets of true and fake samples.

Box 5: Spectral Normalization

Spectral normalization is applied to the weight matrices of a neural network to ensure a boundedness of the error function (e.g., Lipschitzness of the discriminator network in the WGAN context). This helps convergence like any other normalization method, as it provides a guaranty that gradient directions are stable around the current point, allowing larger step widths.

The **spectral norm** (or matrix norm) measures how far a matrix $\boldsymbol{A}$ can stretch a vector $\boldsymbol{x}$:

$$||\boldsymbol{A}|| = \max_{\boldsymbol{x}\neq 0} \frac{||\boldsymbol{A}\boldsymbol{x}||}{||\boldsymbol{x}||}$$

The numerical value of the spectral norm of $\boldsymbol{A}$ can be shown to be just its maximum singular value. To compute the maximum singular value, an algorithmic idea helps: the power iteration method, which yields the maximal eigenvector.

Power iteration uses the fact that any matrix will rotate a random vector toward its largest eigenvector. Therefore, by iteratively calculating $\frac{\boldsymbol{A}\boldsymbol{X}}{|\boldsymbol{A}\boldsymbol{x}|}$, the largest eigenvector is obtained eventually.

In practice, it is observed that a single iteration is already sufficient to achieve the desired normalizing behavior.

Consequently, the key ingredient is the Lipschitzness constraint of the discriminator network,[6] and how to enforce this in a stable and regularized way. It soon turned out that weight clipping is not an ideal choice. Rather, two other methods have been proposed: the gradient penalty approach and normalizing the weights with the spectral norm of the weight matrices.

Both have been added to the standard catalogue of performance-boosting measures in GAN training ever since, where in particular spectral normalization (cf. Box 5) is attractive as it can be implemented very efficiently, has a sound theoretical and mathematical foundation, and ensures stable and efficient training.

3.6.3 Example Application: Brain Abnormality Detection Using WGAN

One of the first applications of Wasserstein GANs in a practical use case was presented in the medical domain, specifically in the context of attributing visible changes of a diseased patient with respect to a normal control to locations in the images [27]. The way this detection problem was cast into a GAN approach (and then solved with a Wasserstein GAN) was to delineate the regions that make the images of a diseased patient look "diseased," i.e., find the residual region, that, if subtracted from the diseased-looking image, would make it look "normal."

Figure 16 shows the construction of the VA-GAN architecture with images from a mocked dataset for illustration. For the authors' results, see their publication and code repository.[7]

For their implementation, the authors note that neither batch normalization nor layer normalization helped convergence and hypothesize that the difference between real and generated examples may be a reason that in particular batch normalization may in fact have an adverse effect especially during the early training phase. Instead, they impose an ℓ_1 norm loss component on the U-Net-generated "visual (feature) attribution" (VA) map to ensure it to be a minimal change to the subject. This serves to prevent the generator from changing the subject into some "average normal" image that it may otherwise learn. They employ an update regime that trains the critic network for more iterations than the generator, but doesn't train it to convergence as proposed in the original WGAN publications. Apart from these measures, in their code repository, the authors give several practical hints and heuristics that may stabilize the training, e.g., using a *tanh* activation for the generator or exploring other dropout settings and in general using a large enough dataset. They also point out that the Wasserstein distance isn't suited for model selection since it is too unstable and not directly correlated to the actual usefulness of the trained model.

[6] The discriminator network in the context of continuous generator loss functions like the Wasserstein-based loss is called a "critique" network, as it no longer discriminates but yields a metric. For ease of reading, this chapter sticks to the term "discriminator."

[7] https://github.com/baumgach/vagan-code.

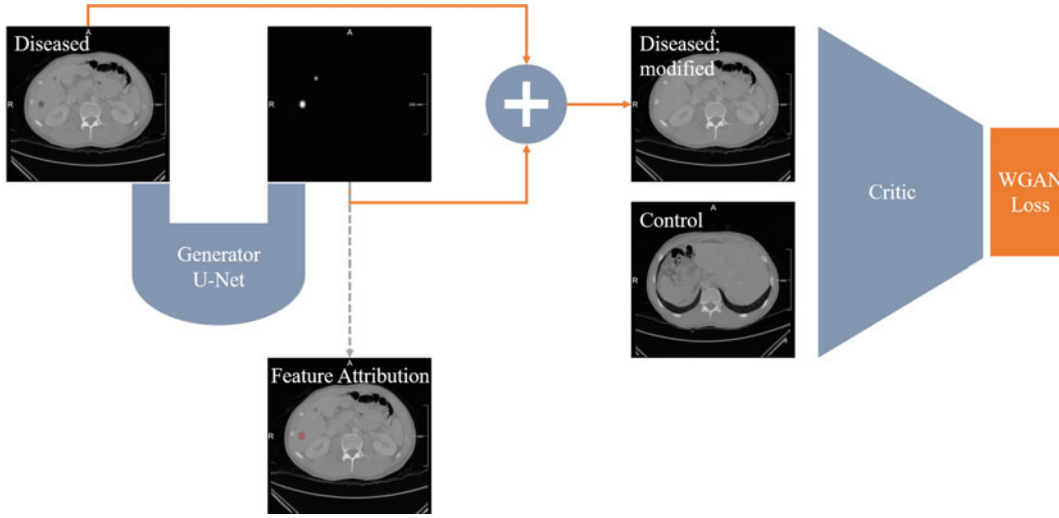

Fig. 16 An image of a diseased patient is run through a U-Net with the goal to yield a map that, if added to the input image, results in a modified image that fools the discriminator ("critique") network into classifying it as a "normal" control. The map can be interpreted as the regions attributed to appear abnormal, giving rise to the name of the architecture: visual attribution GAN (VA-GAN)

This is one more reason to turn in the next section to an important topic in the context of validation for generative models: How to quantify their results?

3.7 GAN Performance Metrics

One imminent question has so far been postponed, though it implicitly plays a crucial role in the quest for "better" GANs: How to actually measure the success of a GAN or the performance in terms of result quality?

GANs can be adapted to solve image analysis tasks like segmentation or detection (cf. Subheading 3.6.3). In such cases, the quality and success can be measured in terms of task-related performance (Jaccard/Dice coefficient for segmentation, overlap metrics for detection etc.).

Performance assessment is less trivial if the GAN is meant to generate unseen images from random vectors. In such scenarios, the intuitive criterion is how convincing the generated results are. But convincing to whom? One could expose human observers to the real and fake images, ask them to tell them apart, and call a GAN better than a competing GAN if it fools the observer more consistently.[8] Since this is practically infeasible, metrics were sought that provide a more objective assessment.

[8] In fact, there is only very little research on the actual performance of GANs in fooling human observers, though guides exist on how to spot "typical" GAN artifacts in generated images. These are older than the latest GAN models, and it can be hypothesized that the lack of such literature is indirect confirmation of the overwhelming capacity of GANs to fool human observers.

The most widely used way to assess GAN image quality is the Fréchet inception distance (FID). This distance is conceptually related to the Wasserstein distance. It has an analytical solution to calculate the distance of Gaussian (normal) distributions. In the multivariate case, the Fréchet distance between two distributions X and Y is given by the squared distance of their means μ_X (resp. μ_Y) and a term depending on the covariance matrix describing their variances Σ_X (resp. Σ_Y):

$$d(X, Y) = ||\mu_X - \mu_Y||^2 + \mathrm{Tr}(\Sigma_X + \Sigma_Y - 2\sqrt{\Sigma_X \Sigma_Y}). \tag{10}$$

The way this distance function is being used is often the score, which is computed as follows:

- Take two batches of images (real/fake, respectively).
- Run them through a feature extraction or embedding model. For FID, the inception model is used, pretrained on ImageNet. Retain the embeddings for all examples.
- Fit each one multivariate normal distribution to the embedded real/fake examples.
- Calculate their Fréchet distance according to the analytical formula in Eq. 10.

This metric has a number of downsides. Typically, if computed for a larger batch of images, it decreases, although the same model is being evaluated. This bias can be remedied, but FID remains the most used metric still. Also, if the inception network cannot capture the features of the data FID should be used on, it might simply be uninformative. This is obviously a grave concern in the medical domain where imaging features look much different from natural images (although, on the other hand, transfer learning for medical classification problems proved to work surprisingly well, so that apparently convolutional filters trained on photographs also extract applicable features from medical images). In any case, the selection of the pretrained embedding model brings a bias into the validation results. Lastly, the assumption of a multivariate normal distribution for the inception features might not be accurate, and only describing it through their means and covariances is a severe reduction of information. Therefore, a qualitative evaluation is still required.

One obvious additional question arises: If the ultimate metric to judge the quality of the generator is given by, for example, the FID, why can't it be used as the optimization goal instead of minimizing a discriminator loss? In particular, as the Fréchet distance is a variant of the Wasserstein distance, an answer to this question is not obvious. In fact, feature matching as described in Box 4 exactly uses this type of idea, and likewise, it has been partially adopted in recent GAN architectures to enhance the stability of training with a more fine-grained loss component than a pure categorical cross-entropy loss on the "real/fake" classification of the discriminator.

Related recent research is concerned with the question how generated results can automatically be detected to counteract fraudulent authors. So-called forensic algorithms detect patterns that point out generated images. This research puts up the question how to detect fake images reliably. Solutions based on different analysis directions encompass image fingerprinting and frequency-domain analysis [28–31].

4 Selected GAN Architectures You Should Know

In the following, we will examine some GAN architectures and GAN developments that were taken up by the medical community or that address specific needs that might make them appealing, e.g., for limited data scenarios.

4.1 Conditional GAN

GANs cannot be told what to produce—at least that was the case with early implementations. It was obvious, though, that a properly trained GAN would imprint the semantics of the domain onto its latent space, which was evidenced by experiments in which the latent space was traversed and images of certain characteristics could be produced by sampling accordingly. Also, it was found that certain dimensions of the latent space can correspond to certain features of the images, like hair color or glasses, so that modifying them alone can add or take away such visible traits.

With the improved development of conditional GANs [32] following a number of GANs that modeled the conditioning input more explicitly, another approach was introduced that was based on the U-Net architecture as a generator and a favorable discriminator network that values local style over a full-image assessment.

Technically, the formulation of a conditional GAN is straightforward. Recalling the value function (learning objective) of GANs from Eq. 5,

$$J(G, D) = \mathbb{E}_{x \sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z \sim p_G}[1 - \log D(G(z))],$$

We now want to condition the generation on some additional knowledge or input. Consequently, both the generator G and the discriminator D will receive an additional "conditioning" input, which we call x. This can be a class label but also any other associated information. Very commonly, the additional input will be an image, as, for example, for image translation application (e.g., transforming from one image modality to another such as, for instance, MRI to CT). The result is the cGAN objective function:

$$J_{\text{cGAN}}(G, D) = \mathbb{E}_{x \sim p_{\text{data}}}[\log D(x|y)] + \mathbb{E}_{z \sim p_G}[1 - \log D(G(z|y))] \tag{11}$$

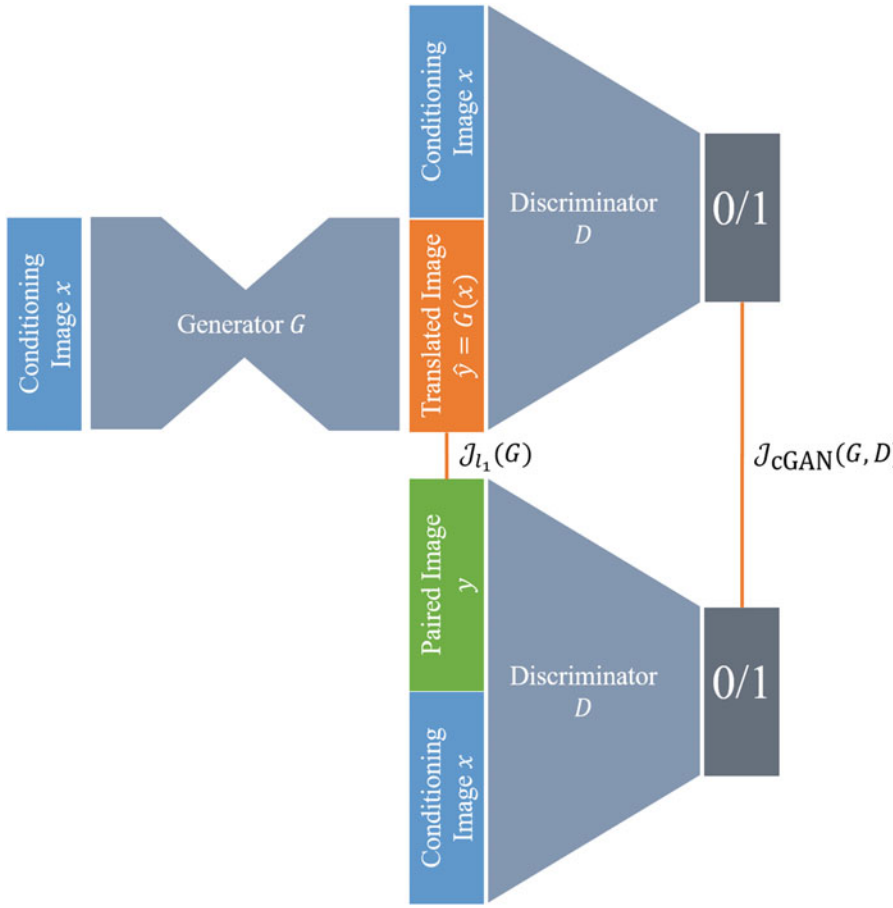

Fig. 17 A possible architecture for a cGAN. Left: the generator network takes the base images x as input and generates a translated image $\hat{y}$. The discriminator receives either this pair of images or a true pair x, y (right). The additional generator reconstruction loss (often a ℓ_1 loss) is calculated between y and $\hat{y}$

Isola et al. [32] describe experiments with MNIST handwritten digits, where a simple generator with two layers of fully connected neurons was used, and similarly for the discriminator. x was set to be the class label. In a second experiment, a CNN creates a feature representation of images, and the generator is trained to generate textual labels (choosing from a vocabulary of about 250.000 encoded terms) for the images conditioned on this feature representation.

Figure 17 shows a possible architecture to employ a cGAN architecture for image-to-image translation. In this diagram, the conditioning input is the target image that the trained network shall be able to produce based on some image input. The generator network therefore is a U-Net. The discriminator network can be implemented, for example, by a classification network. This network always receives two inputs: the conditioning image (x in Fig. 17) and either the generated output $\hat{y}$ or the true paired image y.

Fig. 18 Input and output of a pix2pix experiment. Online demo at https://affinelayer.com/pixsrv/

Note that the work of Isola et al. [32] introduces an additional loss term on the generator that measures the ℓ_1 distance between the generated and ground truth image, which is (with variables as in Eq. 11)

$$\mathcal{J}_{\ell_1}(G) = \mathbb{E}_{x,y,z}\|y - G(x,z)\|_1,$$

where $\|\cdot\|_1$ is the ℓ_1 norm.

The authors do not further justify this loss term apart from stating that ℓ_1 is preferred over ℓ_2 to encourage less blurry results. It can be expected that this loss component provides a good training signal to the generator when the discriminator loss doesn't, e.g., in the beginning of the training with little or no overlap of target and parameterized distributions. The authors propose to give the ℓ_1 loss orders of magnitudes more weight than the discriminator loss component to value accurate translations of images over "just" very plausible images in the target domain.

The cGAN, namely, in the configuration with a U-Net serving as the generative network, was very quickly adopted by artists and scientists, thanks to the free implementation pix2pix.[9] One example created with pix2pix is given in Fig. 18, where the cGAN was trained to produce cat images from line drawings.

One application in the medical domain was proposed, for example, by Senaras et al. [33]. The authors used a U-Net as a generator to produce a stained histopathology image from a label image that has two distinct labels for two kinds of cell nuclei. Here, the label image is the conditioning input to the network. Consequently, the discriminator network, a classification CNN tailored to

[9] https://github.com/phillipi/pix2pix.

the patch-based classification of slides, receives two inputs: the histopathology image and a label image.

Another example employed an augmented version of the conditional GAN to translate CT to MR images of the brain, including a localized uncertainty estimate about the image translation success. In this work, a Bayesian approach to model the uncertainty was taken by including dropout layers in the generator model [34].

Lastly, a 3D version of the pix2pix approach with a 3D U-Net as a generative network was devised to segment gliomas in multimodal brain MRI using data from the 2020 International Multimodal Brain Tumor Segmentation (BraTS) challenge [35]). The authors called their derived model vox2vox, alluding to the extension to 3D data [36].

More conditioning methods have been developed over the years, some of which will be sketched further on. It is common to this type of GANs that paired images are required to train the network.

4.2 CycleGAN

While cGANs require paired data for the gold standard and conditioning input, this is often hard to come by, in particular in medical use cases. Therefore, the development of the CycleGAN set a milestone as it alleviates this requirement and allows to train image-to-image translation networks without paired input samples.

The basic idea in this architecture is to train two mapping functions between two domains and to execute them in sequence so that the resulting output is considered to be in the origin domain again. The output is compared against the original input, and their ℓ_1 or ℓ_2 distance establishes a novel addition to the otherwise usual adversarial GAN loss. This might conceptually remind one of the autoencoder objectives: reproduce the input signal after encoding and decoding; only this time, there is no bottleneck but another interpretable image space. This can be exploited to stabilize the training, since the sequential concatenation of image translation functions, which we will call G and F, can be reversed. Figure 19 shows a schematic of the overall process (left) and one incarnation of the cycle, here from image domain X to $\mathcal{Y}$ and back (middle).

CycleGANs employ several loss terms in training: two adversarial losses $\mathcal{J}(G, D_{\mathcal{Y}})$ and $\mathcal{J}(F, D_X)$ and two cycle consistency losses, of which one $\mathcal{J}_{\text{cyc}}(G, F)$ is indicated rightmost in Fig. 19. Zhu et al. [37] presented the initial publication with a participation of the cGAN author Isola [37]. The cycle consistency losses are ℓ_1 losses in their implementation, and the GAN losses are least square losses instead of negative log likelihood, since more stable training was observed with this choice.

Almahairi et al. [38] provided an augmented version [38], noting that the original implementation suffers from the inability to generate stochastic results in the target domain $\mathcal{Y}$ but rather learns a one-to-one mapping between X and $\mathcal{Y}$ and vice versa. To

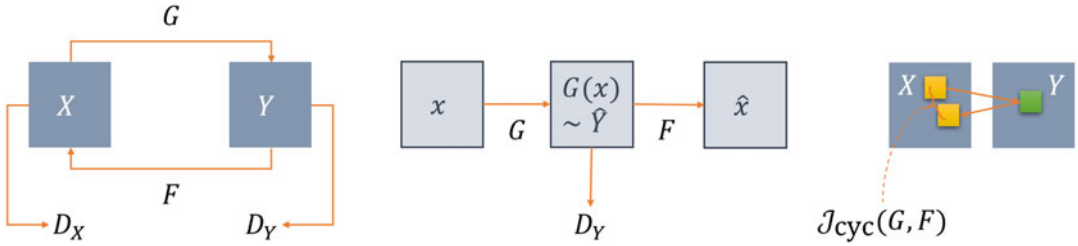

Fig. 19 Cycle GAN. Left: image translation functions G and F convert between two domains. Discriminators D_X and D_Y give adversarial losses in both domains. Middle: for one concrete translation of an image x, the translation to Y and back to X is depicted. Right: after the translation cycle, the original and back-translated result are compared in the *cycle consistency loss*

alleviate this problem, the generators are conditioned on one latent space each for both directions, so that, for the same input $x \in X$, G will now produce multiple generated outputs in Y depending on the sample from the auxiliary latent space (and similarly in reverse). Still, F has to recreate a $\hat{x}$ minimizing the cycle consistency loss for each of these samples. This also remedies a second criticism brought forward against vanilla CycleGANs: these networks can learn to hide information in the (intermediate) target image domain that fool the discriminator but help the backward generator to minimize the cycle consistency loss more efficiently [39]. Chu et al. [39] use adaptive histogram equalization to show that in visually empty regions of the intermediate images information is present. This is a finding reminiscent of adversarial attacks, which the authors elaborate on in their publication.

Zhang et al. [40] show a medical application. In their work, a CycleGAN has been used to train image translation and segmentation models on unpaired images of the heart, acquired with MRI and CT and with gold standard expert segmentations available for both imaging datasets. The authors proposed to learn more powerful segmentation models by enriching both datasets with artificially generated data. To this end, MRIs are converted into CT contrast images and vice versa using GANs. Segmentation models for MRI and CT are then trained on dataset consisting of original images and their expert segmentations and augmented by the converted images, for which expert segmentations can be carried over from their original domain. To achieve this, it is of importance that the converted (translated) images accurately depict the shape of the organs as expected in the target domain, which is enforced using the shape consistency loss.

In the extended setup of the CycleGAN with shape and cycle consistency, three different loss types instead of the original two are combined during training:

Adversarial GAN losses $\mathcal{J}_{\text{GAN}}$. This loss term is the same as defined, e.g., in Eq. 5.

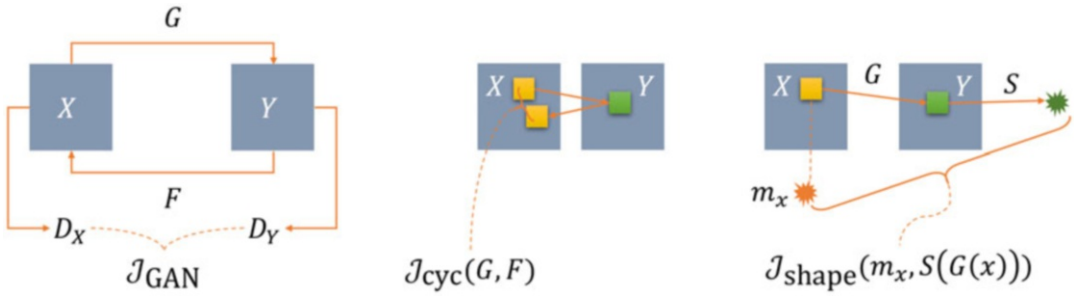

Fig. 20 Cycle GAN with shape consistency loss (rightmost part of figure). Note that the figure shows only one direction to ease readability

Cycle consistency losses $\mathcal{J}_{\text{cyc}}$. This is the ℓ_1 loss presented by the original CycleGAN authors discussed above.

Shape consistency losses $\mathcal{J}_{\text{shape}}$. The shape consistency loss is a new addition proposed by the authors. A cross-correlation loss takes into account two segmentations, the first being the gold standard segmentation m_x for an $x \in X$ and one segmentation produced by a segmenter network S that was trained on domain Y and receives the translated image $\hat{y} = G(x)$.

Figure 20 depicts the three loss components, of which the first two are known already from Fig. 19.

Note that the description as well as Fig. 20 only show one direction for cycle and shape consistency loss. Both are duplicated into the other direction and combined into the overall training objective, which then consists of six components.

In several other works, the CycleGAN approach was extended and combined with domain adaption methods for various segmentation tasks and also extended to volumetric data [41–43].

4.3 StyleGAN and Successor

One of the most powerful image synthesis GANs to date is the successor of StyleGAN, StyleGAN2 [44, 45]. The authors, at the time of writing researching at Nvidia, deviate from the usual GAN approach in which an image is generated from a randomly sampled vector from a latent space. Instead, they use a latent space that is created by a mapping function f which is in their architecture implemented as a multilayer perceptron which maps from a 512-dimensional space Z into a 512-dimensional space W. The second major change consisted of the so-called adaptive instance normalization layer, AdaIN, which implements a normalization to zero-mean and unit variance of each feature map, followed by a multiplicative factor and an additive bias term. This serves to

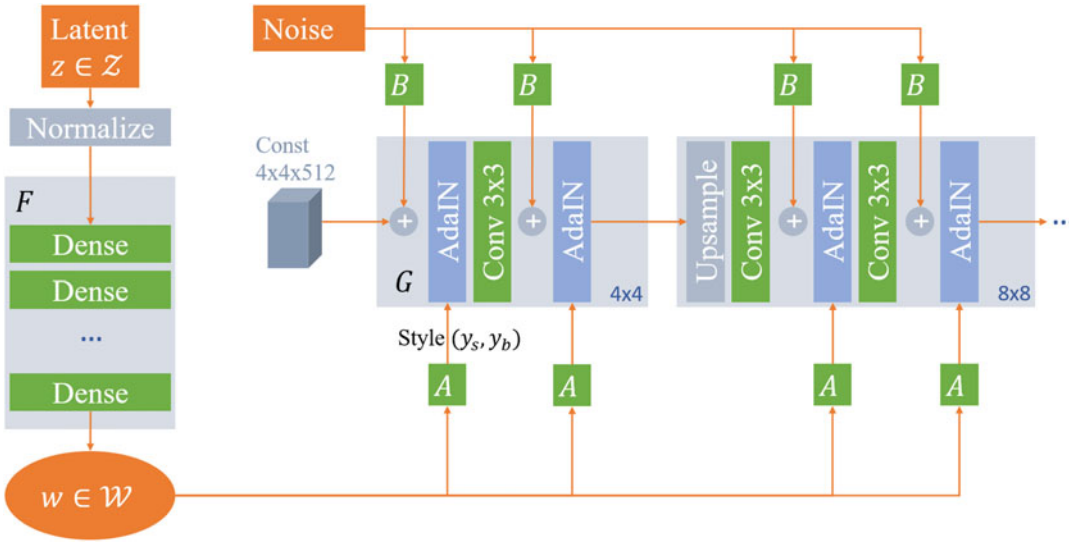

Fig. 21 StyleGAN architecture, after [44]. Learnable layers and transformations are shown in green, the AdaIN function in blue

reweight the importance of feature maps in one layer. To ensure the locality of the reweighting, the operation is followed by the non-linearity. The scaling and bias are two components of $\mathbf{y} = (y_s, y_b)$, which is the result of a learnable affine transformation A applied to a sample from W.

In their experiments, Karras et al. [44] recognized that after these changes, the GAN actually no longer depended on the input vector drawn from W itself, so the random latent vector was replaced by a static vector fed into the GAN. The $\mathbf{y}$, which they call *styles*, remained to be results from a vector randomly sampled from the new embedding space W.

Lastly, noise is added in each layer, which serves to allow the GAN to produce more variation without learning to produce it from actual image content. The noise, like the latent vector, is fed through learnable transformations B, before it is added to the unnormalized feature maps. The overall architecture is sketched in Fig. 21.

In the basic setup, one sample is drawn from W and fed through per-layer learned A to gain per-layer different interpretations of the style, $\mathbf{y} = (y_s, y_b)$. This can be changed, however, and the authors show how using one random sample w_1 in some of the layer blocks and another sample w_2 in the remaining; the result will be a mixture of styles of both individual samples. This way, the coarse attributes of the generated image can stem from one sample and the fine detail from another. Applied to a face generator, for example, pose and shape of the face are determined in the coarse early layers of the network, while hair structure and skin texture are the fine

Fig. 22 Images created with StyleGAN; https:/this{person—artwork—cat—horse—chemical}doesnotexist.com. Last accessed: 2022-01-14

details of the last layers. The architecture and results gained widespread attention through a website,[10] which recently was followed up by further similar pages. Results are depicted in Fig. 22.

The crucial finding in StyleGAN was that the mapping function F transforming the latent space vector from Z to W serves to ensure a disentangled (flattened) latent space. Practically, this means that if interpolating points z_i between two points z_1 and z_2 drawn from Z and reconstructing images from these interpolated points z_i, semantic objects might appear (in a StyleGAN-generating faces, for example, a hat or glasses) that are neither part of the generated images from the first point z_1 nor the second point z_2 between which it has been interpolated. Conversely, if interpolating in W, this "semantic discontinuity" is no longer the case, as the authors show with experiments in which they measure the visual change of resulting images when traversing both latent spaces.

In their follow-up publications, the same authors improve the performance even further. They stick to the basic architecture but redesign the generative network pertaining to the AdaIN function. In addition, they add their metric from [44] that was meant to quantify the entanglement of the latent space as a regularizer. The discriminator network was also enhanced, and the mechanisms of StyleGAN that implement the progressive growing have been successively replaced by more performance-efficient setups. In their experiments, they show a growth of visual and measured quality and removal of several artifacts reported for StyleGAN [45].

4.4 Stabilized GAN for Few-Shot Learning

GAN training was very demanding both regarding GPU power, in particular for high-performance architectures like StyleGAN and StyleGAN2, and, as importantly, availability of data. StyleGAN2, for example, has typical training times of about 10 days on a Nvidia 8-GPU Tesla V100. The datasets comprised at least tens of thousands of images and easily orders of magnitude more. Particularly in the medical domain, such richness of data is typically hard to find.

[10] https://thispersondoesnotexist.com/.

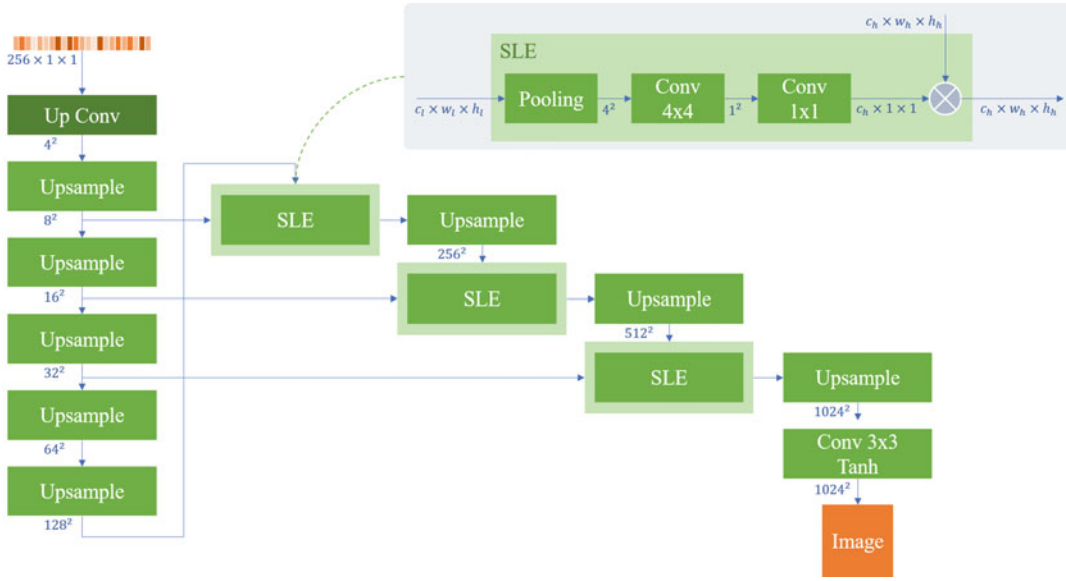

Fig. 23 The FastGAN generator network. Shortcut connections through feature map weighting layers (called skip-layer excitation, SLE) transport information from low-resolution feature maps into high-resolution feature maps. For details regarding the blocks, see text

The authors of [46] propose simple measures to stabilize the training of a specific GAN architecture, which they design from scratch using a replacement for residual blocks, arranged in an architecture with very few convolutional layers, and a loss that drives the discriminator to be less certain when it gets closer to convergence. In sum, this achieves very fast training and yields results competitive with prior GANs [46] and outperforming them in low-data situations.

The key ingredients to the architecture are shortcut connections in the generator model that rescale feature maps of higher resolution with learnable weights derived from low resolutions. The effect is to make fine details simultaneously more independent of direct predecessor feature maps and yet ensure consistency across scales.

A random seed vector of length 256 enters the first block ("Up Conv"), where it is upscaled to a $256 \times 4 \times 4$ tensor. In Fig. 23, the further key blocks of the architecture are "upsample" and "SLE" blocks.

Upsample blocks consist of a nearest-neighbor upsampling followed by a 3×3 convolution, batch normalization, and nonlinearity.

SLE blocks (seen in the top right inset in the architecture diagram) don't touch the incoming high-resolution input (entering from top into the block) but comprise a pooling layer that in each SLE block is set up to yield a

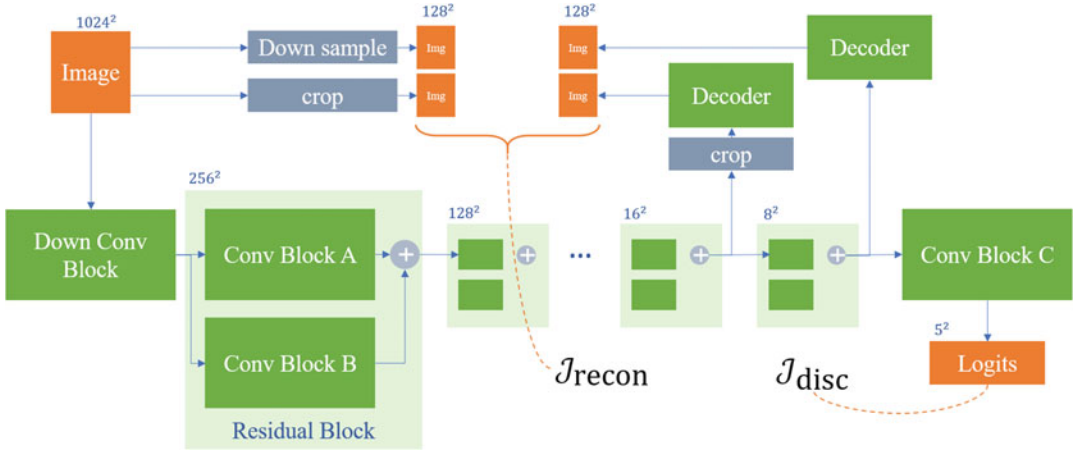

Fig. 24 The FastGAN self-supervision mechanism of the discriminator network. Self-supervision manifests through the loss term indicated by the curly bracket between reconstructions from feature maps and resampled/cropped versions of the original real image, J_{recon}

4×4 stack of feature maps, followed by a convolution to reduce to a 1×1 tensor, which is then in a 1×1 convolution brought to the same number of channels as the high-resolution input. This vector is then multiplied to the channels of the high-resolution input.

Secondly, the architecture introduces a self-supervision feature in the discriminator network. The discriminator network (*see* Fig. 24) is a simple CNN with strided convolutions in each layer, halving resolution in each feature map. In the latest (coarsest) feature maps, simple up-scaling convolutional networks are attached that generate small images, which are then compared in loss functions ($\mathcal{J}_{\text{recon}}$ in Fig. 24) to down-sampled versions of the real input image. This self-supervision of the discriminator is only performed for real images, not for generated ones.

The blocks in the figure spell out as follows:

Down Conv Block consists of two convolutional layers with strided 4×4 convolutions, effectively reducing the resolution from 1024^2 to 256^2.

Residual Blocks have two sub-items, "Conv Block A" being a strided 4×4 convolution to half resolution, followed by a padded 3×3 convolution. "Conv Block B" consists of a strided 2×2 average pooling that quarters resolution, followed by a 1×1 convolution, so that both blocks result in identically shaped tensors, which are then added.

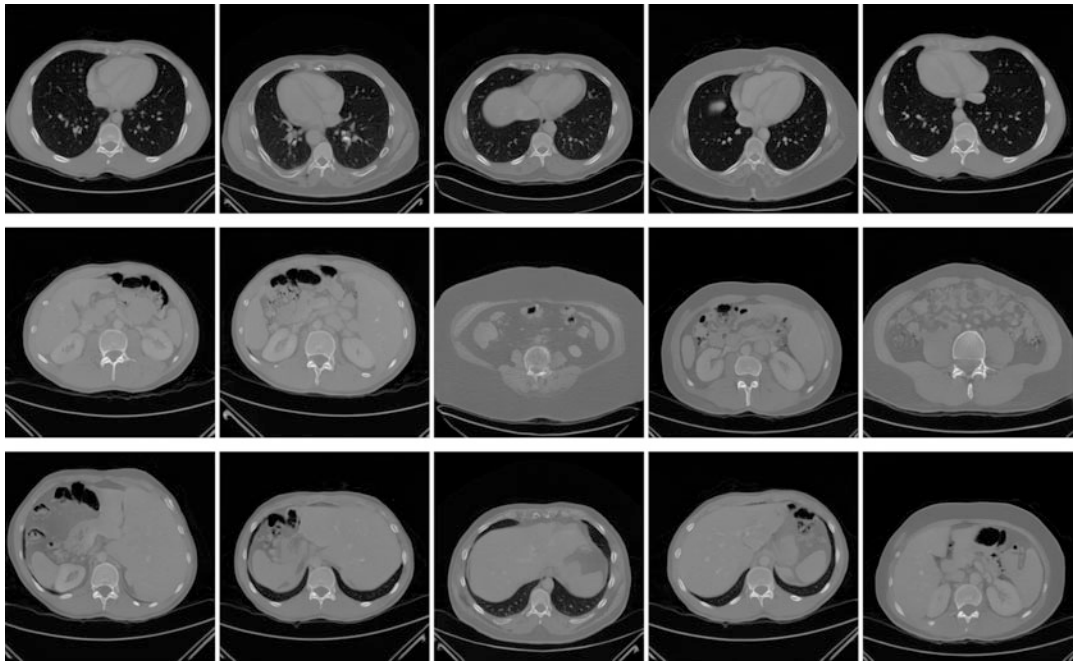

Fig. 25 FastGAN as implemented by the authors has been used to train a CT slice generative model. Images are not cherry-picked, but arranged by similar anatomical regions

Conv Block C consists of a 1×1 convolution followed by a 4×4 convolution without strides or padding, so that the incoming 8^2 feature map is reduced to 5^2.

Decoder The decoder networks are four blocks of upsampling layers each followed by 3×3 convolutions.

The losses employed in the model are the discriminator loss consisting of the hinge version of the usual GAN loss, with the added regularizing reconstruction loss between original real samples and their reconstruction, and the generator loss plainly being $\mathcal{J}_G = \mathbb{E}_{z \sim Z}[D(G(z))]$.

The model is easy to train on modest hardware and little data, as evidenced by own experiments on a set of about 30 chest CTs (about 2500 image slices, converted to RGB). Figure 25 shows randomly picked generated example slices, roughly arranged by anatomical content. It is to be noted that organs appear mirrored in some images. On the other hand, no color artifacts are visible, so that the model has learned to produce only gray scale images. Training time for 50,000 iterations on a Nvidia TitanX GPU was approximately 10 hours.

Fig. 26 The VQGAN+CLIP combination creates images from text inputs, here: "A child drawing of a dark garden full of animals"

4.5 VQGAN

In a recent development, a team of researchers combined techniques for text interpretation with a dictionary of elementary image elements feeding into a generative network. The basic architecture component that is employed goes back to vector quantization variational autoencoders (VQ-VAE), where the latent space is no longer allowed to be continuous, but is quantized. This allows to use the latent space vectors in a look-up table: the visual elements.

Figure 26 was created using code available online, which demonstrates how images of different visual styles can be created using the combination of text-based conditioning and a powerful generative network.

The basis for image generation is the VQGAN ("vector quantization generative adversarial network") [47], which learns representations of input images that can later steer the generative process, in an adversarial framework. The conditioning is achieved with the CLIP ("Contrastive Image-Language Pretraining") model that learns a discriminator that can judge plausible images for a text label or vice versa [48].

The architecture has been developed with an observation in mind that puts the benefits and drawbacks of convolutional and transformer architectures in relation to each other. While the locality bias of convolutional architectures is inappropriate if overall structural image relations should be considered, it is of great help in capturing textural details that can exist anywhere, like fur, hair, pavement, or grass, but where the exact representation of hair

positions or pavement stones is irrelevant. On the other hand, image transformers are known to learn convolutional operators implicitly, posing a severe computational burden without a visible impact on the results. Therefore, Esser et al. [47] suggest to combine convolutional operators for local detail representation and transformer-based components for image structure.

Since the VQGAN as a whole is no longer a pure CNN but for a crucial component uses a transformer architecture, this model will be brought up again briefly in Subheading 5.2.

The VQGAN architecture is derived from the VQ-VAE (vector quantization variational autoencoder) [49], adding a reconstruction loss through a discriminator, which turns it into a GAN. At the core of the architecture is the quantization of estimated codebook entries. Among the quantized entries in the codebook, the closest entry to the query vector coding, an image patch is determined. The found codebook entry is then referred to by its index in the codebook. This quantization operation is non-differentiable, so for end-to-end training, gradients are simply copied through it during backpropagation.

The transformer can then efficiently learn to predict codebook indices from those comprising the current version of the image, and the generative part of the architecture, the decoder, produces a new version of the image. Learning expressive codebook entries is enforced by a perceptual loss that punishes inaccurate local texture, etc. Through this, the authors can show that high compression levels can be achieved—a prerequisite to enable efficient, yet comprehensive, transformer training.

5 Other Generative Models

We have already seen how GANs were not the first approach to image generation but have prevailed for a time when they became computationally feasible and in consequence have been better understood and improved to accomplish tasks in image analysis and image generation with great success. In parallel with GANs, other fundamentally different generative modeling approaches have also been under continued development, most of which have precursors from the "before-GAN" era as well. To give a comprehensive outlook, we will sketch in this last section the state of the art of a selection of these approaches.[11]

[11] The research on the so-called flow-based models, e.g., normalizing flows, has been omitted in this chapter, though acknowledging their emerging relevance also in the context of image generation. Flow-based models are built from sequences of invertible transformations, so that they learn data distributions explicitly at the expense of sometimes higher computational costs due to their sequential architecture. When combined, e.g., with a powerful GAN, they allow innovative applications, for example, to steer the exploration of a GAN's latent space to achieve fine-grained control over semantic attributes for conditional image generation. Interested readers are referred to the literature [11, 13, 50–52].

5.1 Diffusion and Score-Based Models

Diffusion models take a completely different approach to distribution estimation. GANs implicitly represent the target distribution by learning a surrogate distribution. Likelihood-based models like VAE approximate the target distribution explicitly, not requiring the surrogate. In diffusion models, however, the gradient of the log probability density function is estimated, instead of looking at the distribution itself (which would be the unfathomable integral of the gradient). This value is known as the Stein score function, leading to the notion that diffusion models are one variant of score-based models [53].

The simple idea behind this class of models is to revert a sequential noising process. Consider some image. Then, perform a large number of steps. In each step, add a small amount of noise from a known distribution, e.g., the normal distribution. Do this until the result is indistinguishable from random noise.

The denoising process is then formulated as a latent variable model, where $T-1$ latents successively progress from a noise image $\mathbf{x}_T \sim \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$ to the reconstruction that we call $\mathbf{x}_0 \sim q(\mathbf{x}_0)$. The reconstructed image, $\mathbf{x_0}$, is therefore obtained by a *reverse process* $q_\theta(\mathbf{x}_{0:T})$. Note that each step in this chain can be evaluated in closed form [54]. Several model implementations of this approach exist, one being the deep diffusion probabilistic model (DDPM). Here, a deep neural network learns to perform one denoising step given the so-far achieved image and a $t \in \{1, \ldots, T\}$. Iterative application of the model to the result of the last iteration will eventually yield a generated image from noise input.

Autoregressive diffusion models (ARDMs) [55] follow yet another thought model, roughly reminiscent of PixelRNNs we have briefly mentioned above (*see* Subheading 3.2). Both share the approach to condition the prediction of the next pixel or pixels on the already predicted ones. Other than in the PixelRNN, however, the specific ARDM proposed by the authors does not rely on a predetermined schedule of pixel updates, so that these models can be categorized as latent variable models.

As of late, the general topic of score-based methods, among which diffusion models are one variant, received more attention in the research community, fueled by a growing body of publications that report image synthesis results that outperform GANs [53, 56, 57]. Score function-based and diffusion models superficially share the similar concept of sequentially adding/removing noise but achieve their objective with very different means: where score function-based approaches are trained by score-matching and their sampling process uses Langevin dynamics [58], diffusion models are trained using the evidence lower bound (ELBO) and sample with a decoder, which is commonly a neural network. Figure 27 visualizes an example for a score function.

Score function-based (sometimes also score-matching) generative models have been developed to astounding quality levels, and

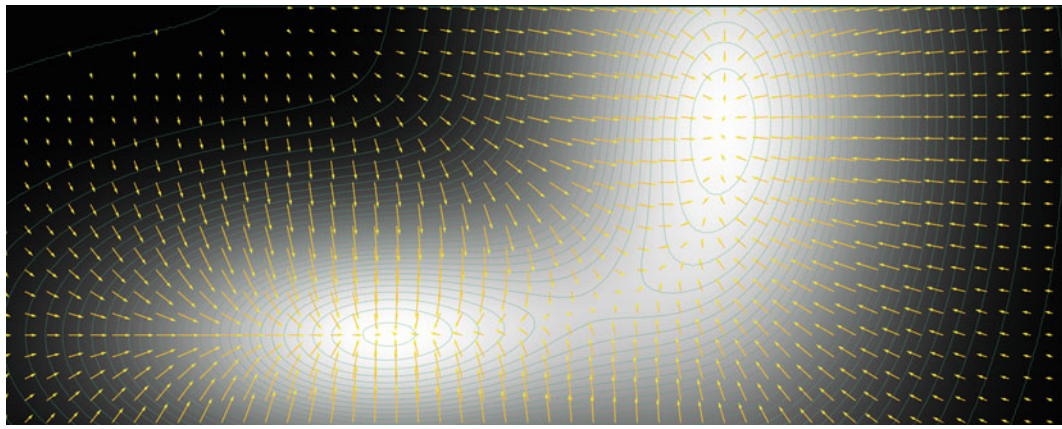

Fig. 27 The Stein score function can be conceived of as the gradient of the log probability density function, here indicated by two Gaussians. The arrows represent the score function

the recent works of Yang Song and others provide accessible blog posts,[12] and a comprehensive treatment of the subject in several publications [53, 58, 59].

In the work of Ho et al. [54], the stepwise reverse (denoising) process is the basis of the denoising diffusion probabilistic models (DDPM). The authors emphasize that a proper selection of the noise schedule is crucial to fast, yet high-quality, results. They point out that their work is a combination of diffusion probabilistic models with score-matching models, in this combination also generalizing and including the ideas of autoregressive denoising models. In an extension of Ho et al.'s [54] work by Nichol and Dhariwal [57], an importance sampling scheme was introduced that lets the denoising process steer the most easy to predict next image elements. Equipped with this new addition, the authors can show that, in comparison to GANs, a wider region of the target distribution is covered by the generative model.

5.2 Transformer-Based Generative Models

The basics of how attention mechanisms and transformer architectures work will be covered in the subsequent chapter on this promising technology (Chapter 6). Attention-based models, predominantly transformers, have been used successfully for some time in sequential data processing and are now considered the superior alternative to recurrent networks like long-short-term memory (LSTM) networks. Transformers have, however, only recently made their way into the image analysis and now also the image generation world. In this section, we will only highlight some developments in the area of generative tasks.

[12] https://yang-song.github.io/blog/.

Google Brain/Google AI's 2018 publication on so-called image transformers [60], among other tasks, shows successful conditional image generation for low-resolution input images to achieve super-resolution output images, and for image inpainting, where missing or removed parts of input images are replaced by content produced by the image transformer.

OpenAI have later shown that even unmodified language transformers can succeed to model image data, by dealing in sheer compute power for hand modeling of domain knowledge, which was the basis for the great success of previous unsupervised image generation models. They have trained Image GPT (or iGPT for short), a multibillion parameter language transformer model, and it excels in several image generation tasks, though only for fairly small image sizes [61]

In the recent past, StyleSwin has been proposed by Microsoft Research Asia [62], enabling high-resolution image generation. However, the approach uses a block-wise attention window, thereby potentially introducing spatial incoherencies at block edges, which they have to correct for.

"Taming transformers" [47], another recent publication already mentioned above, uses what the authors call a learned template code book of image components, which is combined with a vector quantization GAN (VQGAN). The VQGAN is structurally modeled after the VQ-VAE but adds a discriminator network. A transformer model in this architecture composes these code book elements and is interrogated by the GAN variational latent space, conditioned on a textual input, a label image, or other possible inputs. The GAN reconstructs the image from the so-quantized latent space using a combination of a perceptual loss assessing the overall image structure and a patch-based high-resolution reconstruction loss. By using a sliding attention window approach, the authors prevent patch border artifacts known from StyleSwin. Conditioning on textual input makes use of parts of the CLIP [48] idea ("Contrastive Language-Image Pretraining"), where a language model was train in conjunction with an image encoder to learn embeddings of text-image pairs, sufficient to solve many image understanding tasks with competitive precision, without specific domain adaption.

It is evidenced by the lineup of institutions that training image transformer models successfully is nothing that can be achieved with modest hardware or on even a medium-scale image database. In particular for the medical area, where data is comparatively scarce even under best assumptions, the power of such models will only be available in the near future if domain transfer learning can be successfully achieved. This, however, is a known strength of transformer architectures.

Acknowledgements

I thank my colleague at the Fraunhofer Institute for Digital Medicine MEVIS, Till Nicke, for his thorough review of the chapter and many valuable suggestions for improvements. I owe many thanks more to other colleagues for their insights both in targeted discussions and most importantly in everyday work life.

References

[1] Goodfellow IJ, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative adversarial nets. In: Proceedings of the 27th international conference on neural information processing systems - volume, NIPS'14 . MIT Press, Cambridge, pp 2672–2680

[2] Casella G, Berger RL (2021) Statistical inference. Cengage Learning, Boston

[3] Grinstead C, Snell LJ (2006) Introduction to probability. Swarthmore College, Swarthmore

[4] Severini TA (2005) Elements of distribution theory, vol 17. Cambridge University Press, Cambridge

[5] Murphy KP (2012) Machine learning: a probabilistic perspective. MIT Press, Cambridge

[6] Murphy KP (2022) Probabilistic machine learning: an introduction. MIT Press, Cambridge. http://doi.org/probml.ai

[7] Do CB, Batzoglou S (2008) What is the expectation maximization algorithm? Nat Biotechnol 26:8, 26:897–899. https://doi.org/10.1038/nbt1406. https://www.nature.com/articles/nbt1406

[8] Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the em algorithm. J Roy Statist Soc Ser B (Methodolog) 39:1–22. https://doi.org/10.1111/J.2517-6161.1977.TB01600.X.https://onlinelibrary.wiley.com/doi/full/10.1111/j.2517-6161.1977.tb01600.x. https://onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1977.tb01600.x. https://rss.onlinelibrary.wiley.com/doi/10.1111/j.2517-6161.1977.tb01600.x

[9] van den Oord A, Kalchbrenner N, Kavukcuoglu K (2016) Pixel recurrent neural networks. ArXiv abs/1601.06759

[10] Magnusson K (2020) Understanding maximum likelihood: an interactive visualization. https://rpsychologist.com/likelihood/

[11] Rezende DJ, Mohamed S (2015) Variational inference with normalizing flows. In: ICML

[12] van den Oord A, Kalchbrenner N, Espeholt L, Kavukcuoglu K, Vinyals O, Graves A (2016) Conditional image generation with PixelCNN decoders. In: NIPS

[13] Dinh L, Sohl-Dickstein J, Bengio S (2017) Density estimation using Real NVP. ArXiv abs/1605.08803

[14] Salakhutdinov R, Hinton G (2009) Deep Boltzmann machines. In: van Dyk D, Welling M (eds) Proceedings of the twelfth international conference on artificial intelligence and statistics, PMLR, hilton clearwater beach resort, clearwater beach, Florida USA, Proceedings of Machine Learning Research, vol 5, pp 448–455. https://proceedings.mlr.press/v5/salakhutdinov09a.html

[15] Weng L (2018) From autoencoder to Beta-VAE. lilianwenggithubio/lil-log. http://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

[16] Kingma DP, Welling M (2014) Auto-encoding variational bayes. ArXiv 1312.6114

[17] Creswell A, White T, Dumoulin V, Arulkumaran K, Sengupta B, Bharath AA (2018) Generative adversarial networks: an overview. IEEE Signal Process Mag 35(1): 53–65. https://doi.org/10.1109/MSP.2017.2765202

[18] Arjovsky M, Bottou L (2017) Towards principled methods for training generative adversarial networks. ArXiv abs/1701.04862

[19] Theis L, van den Oord A, Bethge M (2016) A note on the evaluation of generative models. CoRR abs/1511.01844

[20] Radford A, Metz L, Chintala S (2015) Unsupervised representation learning with

deep convolutional generative adversarial networks. ArXiV http://arxiv.org/abs/1511.06434

[21] Islam J, Zhang Y (2020) GAN-based synthetic brain PET image generation. Brain Inform 7:1–12. https://doi.org/10.1186/S40708-020-00104-2/FIGURES/9. https://braininformatics.springeropen.com/articles/10.1186/s40708-020-00104-2

[22] Arjovsky M, Chintala S, Bottou L (2017) Wasserstein GAN. ArXiv http://arxiv.org/abs/1701.07875v3. 1701.07875

[23] Gulrajani I, Ahmed F, Arjovsky M, Dumoulin V, Courville A (2017) Improved training of Wasserstein GANs. ArXiV http://arxiv.org/abs/1704.00028v3. nIPS camera-ready, 1704.00028

[24] Villani C (2009) Optimal transport, old and new. Springer, Berlin. https://doi.org/10.1007/978-3-540-71050-9. https://www.cedricvillani.org/wp-content/uploads/2012/08/preprint-1.pdf

[25] Basso G (2015) A Hitchhiker's guide to Wasserstein distances. https://homeweb.unifr.ch/BassoG/pub/A%20Hitchhikers%20guide%20to%20Wasserstein.pdf

[26] Weng L (2019) From GAN to WGAN. ArXiv 1904.08994

[27] Baumgartner CF, Koch LM, Tezcan KC, Ang JX, Konukoglu E (2018) Visual feature attribution using Wasserstein GANs. In: The IEEE conference on computer vision and pattern recognition (CVPR)

[28] Dzanic T, Shah K, Witherden FD (2020) Fourier spectrum discrepancies in deep network generated images. In: 34th conference on neural information processing systems (NeurIPS)

[29] Joslin M, Hao S (2020) Attributing and detecting fake images generated by known GANs. In: Proceedings - 2020 IEEE symposium on security and privacy workshops, SPW 2020. Institute of Electrical and Electronics Engineers, Piscataway, pp 8–14. https://doi.org/10.1109/SPW50608.2020.00019

[30] Le BM, Woo SS (2021) Exploring the asynchronous of the frequency spectra of GAN-generated facial images. ArXiv https://arxiv.org/abs/2112.08050v1. 2112.08050

[31] Goebel M, Nataraj L, Nanjundaswamy T, Mohammed TM, Chandrasekaran S, Manjunath BS, Maya (2021) Detection, attribution and localization of GAN generated images. Electron Imag. https://doi.org/10.2352/ISSN.2470-1173.2021.4.MWSF-276

[32] Isola P, Zhu JY, Zhou T, Efros AA (2016) Image-to-image translation with conditional adversarial networks. ArXiV http://arxiv.org/abs/1611.07004

[33] Senaras C, Sahiner B, Tozbikian G, Lozanski G, Gurcan MN (2018) Creating synthetic digital slides using conditional generative adversarial networks: application to Ki67 staining. In: Medical imaging 2018: digital pathology, society of photo-optical instrumentation engineers (SPIE) conference series, vol 10581, p 1058103. https://doi.org/10.1117/12.2294999

[34] Zhao G, Meyerand ME, Birn RM (2021) Bayesian conditional GAN for MRI brain image synthesis. ArXiV 2005.11875

[35] Bakas S, Reyes M, ..., Menze B (2019) Identifying the best machine learning algorithms for brain tumor segmentation, progression assessment, and overall survival prediction in the BRATS challenge. ArXiV 1811.02629

[36] Cirillo MD, Abramian D, Eklund A (2020) Vox2Vox: 3D-GAN for brain tumour segmentation. ArXiV 2003.13653

[37] Zhu JY, Park T, Isola P, Efros AA (2017) Unpaired image-to-image translation using cycle-consistent adversarial networks. In: 2017 IEEE international conference on computer vision (ICCV), IEEE, pp 2242–2251. http://ieeexplore.ieee.org/document/8237506/papers3://publication/doi/10.1109/ICCV.2017.244

[38] Almahairi A, Rajeswar S, Sordoni A, Bachman P, Courville A (2018) Augmented CycleGAN: Learning many-to-many mappings from unpaired data. ArXiV https://arxiv.org/pdf/1802.10151.pdf. 1802.10151

[39] Chu C, Zhmoginov A, Sandler M (2017) CycleGAN, a master of steganography. ArXiV http://arxiv.org/abs/1712.02950

[40] Zhang Z, Yang L, Zheng Y (2018) Translating and segmenting multimodal medical volumes with cycle- and shape-consistency generative adversarial network. In: 2018 IEEE/CVF conference on computer vision and pattern recognition, IEEE, pp 9242–9251. https://doi.org/10.1109/CVPR.2018.00963. https://ieeexplore.ieee.org/document/8579061/

[41] Hoffman J, Tzeng E, Park T, Zhu JY, Isola P, Saenko K, Efros AA, Darrell T (2017) CyCADA: Cycle-consistent adversarial domain adaptation. ArXiV 1711.03213

[42] Huo Y, Xu Z, Bao S, Assad A, Abramson RG, Landman BA (2018) Adversarial synthesis learning enables segmentation without target modality ground truth. In: 2018 IEEE 15th international symposium on biomedical imaging (ISBI 2018), pp 1217–1220. https://doi.org/10.1109/ISBI.2018.8363790

[43] Yang D, Xiong T, Xu D, Zhou SK (2020) Segmentation using adversarial image-to-image networks. In: Handbook of medical image computing and computer assisted intervention, pp 165–182. https://doi.org/10.1016/B978-0-12-816176-0.00012-0

[44] Karras T, Laine S, Aila T (2018) A style-based generator architecture for generative adversarial networks. IEEE Trans Pattern Analy Mach Intell 43:4217–4228. https://doi.org/10.1109/TPAMI.2020.2970919. https://arxiv.org/abs/1812.04948v3

[45] Karras T, Laine S, Aittala M, Hellsten J, Lehtinen J, Aila T (2020) Analyzing and improving the image quality of StyleGAN. In: Proceedings of the IEEE computer society conference on computer vision and pattern recognition, pp 8107–8116. https://doi.org/10.1109/CVPR42600.2020.00813. https://arxiv.org/abs/1912.04958v2

[46] Liu B, Zhu Y, Song K, Elgammal A (2021) Towards faster and stabilized GAN training for high-fidelity few-shot image synthesis. In: International conference on learning representations. https://openreview.net/forum?id=1Fqg133qRaI

[47] Esser P, Rombach R, Ommer B (2021) Taming transformers for high-resolution image synthesis. In: 2021 IEEE/CVF conference on computer vision and pattern recognition (CVPR), pp 12868–12878. https://doi.org/10.1109/CVPR46437.2021.01268

[48] Radford A, Kim JW, Hallacy C, Ramesh A, Goh G, Agarwal S, Sastry G, Askell A, Mishkin P, Clark J, Krueger G, Sutskever I (2021) Learning transferable visual models from natural language supervision. ArXiV 2103.00020

[49] van den Oord A, Vinyals O, Kavukcuoglu K (2017) Neural discrete representation learning. CoRR abs/1711.00937. http://arxiv.org/abs/1711.00937

[50] Weng L (2018) Flow-based deep generative models. lilianwenggithubio/lil-log. http://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html

[51] Kingma DP, Dhariwal P (2018) Glow: generative flow with invertible 1x1 convolutions. ArXiv https://doi.org/10.48550/ARXIV.1807.03039. https://arxiv.org/abs/1807.03039

[52] Abdal R, Zhu P, Mitra NJ, Wonka P (2021) StyleFlow: attribute-conditioned exploration of StyleGAN-generated images using conditional continuous normalizing flows. ACM Trans Graph 40(3):1–21. https://doi.org/10.1145/3447648. https://doi.org/10.1145%2F3447648

[53] Song Y, Sohl-Dickstein J, Kingma DP, Kumar A, Ermon S, Poole B (2021) Score-based generative modeling through stochastic differential equations. In: International conference on learning representations. https://openreview.net/forum?id=PxTIG12RRHS

[54] Ho J, Jain A, Abbeel P (2020) Denoising diffusion probabilistic models. ArXiV 2006.11239

[55] Hoogeboom E, Gritsenko AA, Bastings J, Poole B, van den Berg R, Salimans T (2021) Autoregressive diffusion models. ArXiV 2110.02037

[56] Dhariwal P, Nichol A (2021) Diffusion models beat GANs on image synthesis. ArXiV http://arxiv.org/abs/2105.05233

[57] Nichol A, Dhariwal P (2021) Improved denoising diffusion probabilistic models. ArXiV http://arxiv.org/abs/2102.09672

[58] Song Y, Ermon S (2019) Generative modeling by estimating gradients of the data distribution. In: Advances in neural information processing systems, pp 11895–11907

[59] Song Y, Garg S, Shi J, Ermon S (2019) Sliced score matching: a scalable approach to density and score estimation. In: Proceedings of the thirty-fifth conference on uncertainty in artificial intelligence, UAI 2019, Tel Aviv, Israel, July 22–25, 2019, p 204. http://auai.org/uai2019/proceedings/papers/204.pdf

[60] Parmar N, Vaswani A, Uszkoreit J, Łukasz Kaiser, Shazeer N, Ku A, Tran D (2018) Image transformer. ArXiV 1802.05751

[61] Chen M, Radford A, Child R, Wu J, Jun H, Luan D, Sutskever I (2020) Generative pretraining from pixels. In: Daumé III H, Singh A (eds) Proceedings of the 37th international conference on machine learning, PMLR, proceedings of machine learning research, vol 119, pp 1691–1703. https://proceedings.mlr.press/v119/chen20s.html

[62] Zhang B, Gu S, Zhang B, Bao J, Chen D, Wen F, Wang Y, Guo B (2021) StyleSwin: transformer-based GAN for high-resolution image generation. ArXiV 2112.10762