JMIV manuscript No.
(will be inserted by the editor)

Processsing Simple Geometric Attributes with Autoencoders

Alasdair Newson[⋆] · **Andrés Almansa**[† ‡] · **Yann Gousseau**[⋆ ‡] · **Saïd Ladjal**[⋆ ‡]

the date of receipt and acceptance should be inserted later

Abstract Image synthesis is a core problem in modern deep learning, and many recent architectures such as autoencoders and Generative Adversarial networks produce spectacular results on highly complex data, such as images of faces or landscapes. While these results open up a wide range of new, advanced synthesis applications, there is also a severe lack of theoretical understanding of how these networks work. This results in a wide range of practical problems, such as difficulties in training, the tendency to sample images with little or no variability, and generalisation problems. In this paper, we propose to analyse the ability of the simplest generative network, the autoencoder, to encode and decode two simple geometric attributes : size and position. We believe that, in order to understand more complicated tasks, it is necessary to first understand how these networks process simple attributes. For the first property, we analyse the case of images of centred disks with variable radii. We explain how the autoencoder projects these images to and from a latent space of smallest possible dimension, a scalar. In particular, we describe both the encoding process and a closed-form solution to the decoding training problem in a network without biases, and show that during training, the network indeed finds this solution. We then investigate the best regularisation approaches which yield networks that generalise well. For the second property, position, we look at the encoding and decoding of Dirac delta functions, also known as "one-hot" vectors. We describe a hand-crafted filter that achieves encoding perfectly, and show that the network naturally finds this filter during training. We also show experimentally that the decoding can be achieved if the dataset is sampled in an appropriate manner. We hope that the insights given here will provide better understanding of the precise mechanisms used by generative networks, and will ultimately contribute to producing more robust and generalisable networks.

Keywords Deep learning · image synthesis · generative models · autoencoders

[⋆] LTCI, Télécom ParisTech, Université Paris Saclay
46 rue Barrault, 75013, Paris
[†] CNRS MAP5, Université Paris Descartes
45, rue des Saints-Pères, 75006, Paris
[‡] : indicates equal contribution

1 Introduction

Image synthesis is a central issue of modern deep learning, and in particular encoder-decoder neural networks (NNs), which include many popular networks such as autoencoders, Generative Adversarial Networks (GANs) [8], variational autoencoders [13] etc. These networks are able to produce truly impressive results [20,25,5,11,12]. However, as in many areas of deep learning, there is a severe lack of theoretical understanding of the networks. In practice, this means that these approaches suffer from a variety of problems, such as difficulty in training the networks [23], the tendency to sample with little or no variety ("mode collapse" [19]), and generalisation problems. There is also the extremely important question of how to interpolate in the latent space (ie how to interpolate between two visual objects via the latent space), which is still an open problem and is mostly done with linear interpolation at the moment [25,24]. Such questions must be answered if these types of networks can be used reliably. For this, we need to understand the inner workings of these networks.

In this paper, we propose to study how the *autoencoder* (the simplest generative neural network) processes two basic image properties:

- size;
- position.

In order to do this, we shall analyse the manner in which the autoencoder works in the case of very simple images. We believe that, in order to understand more complicated synthesis situations, it is necessary to first understand how these networks process simple attributes. For the first property, size, we will look at grey-level images of *disks* with different radii, as such images represent a very simple setting for the notion of size. Secondly, we will look at images containing Dirac delta functions (vectors where one element is non-zero, also called "one-hot vectors"), and determine how the autoencoder can extract the position from such signals. Again, this appears the simplest way to study how the spatial position of an object is processed by such networks. Studying these mechanisms is extremely important, to understand how these networks work . A recent work by Liu et al. [17] also highlighted the importance of studying how NNs work in such simple cases, and their experimental study of one such case lead them to propose the "CoordConv" network layer. In our work, we propose a theoretical investigation of the autoencoder in the two aforementioned cases.

There are several advantages to such an approach. Firstly, since the class of objects we consider has an explicit parametrisation, we know the optimal compression which the autoencoder should obtain. In other words, we know the minimum size of latent space which is sufficient to correctly represent the data. Most applications of autoencoders or similar networks consider relatively high-level input objects, ranging from the MNIST handwritten digits to abstract sketches of conceptual objects [25,9]. Secondly, the nature of our approach fixes certain architecture characteristics of the network, such as the number of layers, leaving fewer free parameters to tune. This means that the conclusions which we obtain are more likely to be robust than in the case of more high-level applications. Finally, we can analyse the generalisation capacity of the autoencoder with greater precision. Indeed, a central problem of deep learning is ensuring that the network is able to generalise outside of the observed data. We are able to study how well the autoencoder does this by removing data from the training set which correspond to a certain region of the parameters, and see whether the autoencoder is able to reconstruct data in that zone.

To summarise, we propose the following contributions in this paper :

- We verify that the autoencoder can correctly learn how to encode and decode a simple shape (the disk) to and from a single scalar, where the size of the disks is the varying parameter.
- We investigate and explain the internal mechanisms of the autoencoder which achieve this. In particular, we show that the encoder extracts the area when a contractive loss is considered, and we describe a closed-form solution to the decoding training problem in a network without biases. These behaviours are verified experimentally.
- We analyse the best regularisation approaches which lead to better generalisation in the case where certain disk sizes are not observed in the training data.
- We show how the autoencoder can process the *position* of an object in an image. For this, we study the simple case of a Dirac (ie a one-hot vector) as an input to the network.

One of the ultimate, long-term, goals in studying the precise properties of autoencoders in simple cases such as these is to identify architectures and regularisations which yield robust autoencoders which can generalise well in regions unobserved during training. We hope that this work can contribute to attaining this goal. The code implementing this work is available at [1].

2 Prior work

The concept of autoencoders has been present for some time in the learning community [15,4]. Autoencoders are neural networks, often convolutional neural networks, whose purpose is twofold. Firstly, to compress some input data by transforming it from the input domain to another space, known as the latent (or code) space, which is learned by the network. The second goal of the autoencoder is to take this latent representation and transform it back to the original space, such that the output is similar, with respect to some criterion, to the input. In most applications, the dimensionality d of the latent space is smaller than that of the original data, so that the autoencoder is encouraged to discover useful features of the data. In practice, we obviously do not know the exact value of d, but we would still like to impose as much structure in the latent space as possible. This idea lead to the regularisation in the latent space of autoencoders, which comes in several flavours. The first is the sparse autoencoder [21], which attempts to have as few active (non-zero) neurons as possible in the network. This can be done either by modifying the loss function to include sparsity-inducing penalisations, or by acting directly on the values of the code z. In the latter option, one can use rectified linear units (ReLUs) to encourage zeros in the code [6] or simply specify a maximum number of non-zero values as in the “k-sparse” autoencoder

[1] https://github.com/alasdairnewson/geometric_autoencoder

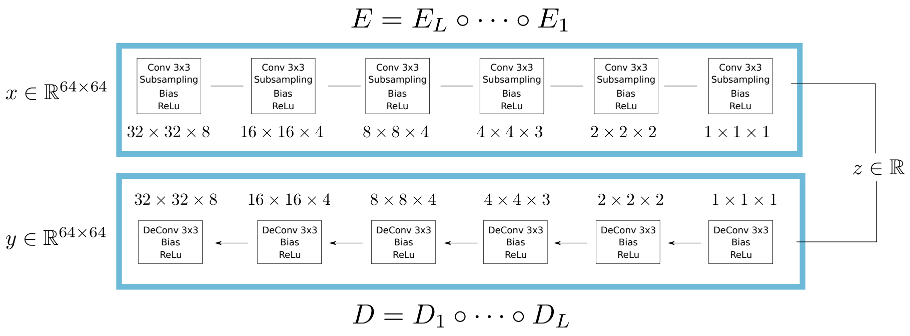


Fig. 1: Generic autoencoder architecture used in the geometric experiments of Section 4.

[18]. Another approach, taken by the variational autoencoder, is to specify the a priori distribution of the code z. [13] use the Kullback-Leibler divergence to achieve this goal, and the authors impose a Gaussian distribution on z. The “contractive” autoencoder [22] encourages the derivatives of the code with respect to the input image to be small, meaning that the representation of the image should be robust to small changes in the input.

Autoencoders can be applied to a variety of problems, such as denoising (“denoising autoencoder”) or image compression ([2]). For a good overview of autoencoders, see the book of Goodfellow et al. [7]. Recently, a great deal of attention has been given to the capacity of GANs [8] and autoencoders, to generate new images. In the past couple of years, increasingly impressive results have been produced by more and more complex networks [25,5,11,12]. It is well-known that these networks have important limitations, such as the tendency to produce low quality images or to reproduce images from the training set because of mode collapse [19]. Overcoming these drawbacks requires us to understand generative networks in greater depth, and the best place to start such an investigation is with simple cases, which we now proceed to analyse.

3 Notation and Autoencoder Architecture

Although autoencoders have been extensively studied, very little is known concerning the actual inner mechanics of these networks, in other words quite simply, how they work. In this work, we aim to discover how, with a cascade of simple operations common in deep networks, an autoencoder can encode and decode very simple images. In view of this goal, we propose to study in depth the case of *disks* of variable radii. There are two advantages to this approach. Firstly, it allows for a full understanding in a simplified case, and secondly, the true dimensionality of the latent space is known, and therefore the architecture is constrained.

Before continuing, we describe our autoencoder in a more formal fashion.

We consider square input images, which we denote with $x \in \mathbb{R}^{m\times m}$, and codes $z \in \mathbb{R}^d$, and d is the dimension of the latent space. The autoencoder consists of the

Layer	Input	Hidden layers					Code (z)
Depths	1	8	4	4	3	2	1

Parameter	Spatial filter size	Non-linearity	Learning rate	Learning algorithm	Batch size
Value	3×3	Leaky ReLU ($\alpha = 0.2$, see Eq. (2))	0.001	Adam	300

Table 1: Parameters of autoencoder designed for processing centred disks of random radii.

couple (E, D), the encoder and decoder which transform to and from the "code" space, with $E : \mathbb{R}^{m \times n} \to \mathbb{R}^d$ and $D : \mathbb{R}^d \to \mathbb{R}^{m \times m}$. As mentioned, the goal of the auto-encoder is to compress and uncompress a signal to (and from) a representation with a smaller dimensionality, while losing as little information as possible. Thus, we search for the parameters of the encoder and the decoder, which we denote with Θ_E and Θ_D respectively, by minimising

$$(\Theta_E, \Theta_D) = \underset{\Theta_E, \Theta_D}{\operatorname{argmin}} \sum_x ||x - D(E(x))||_2^2 \tag{1}$$

The autoencoder consists of a series of convolutions with filters of small compact support, sub-sampling/up-sampling, biases and non-linearities. The values of the filters are termed the weights of the network, and we denote the encoding filters with $w_{\ell,i}$, where ℓ is the layer and i the index of the filter. Similarly, we denote the decoding filters $w'_{\ell,i}$. Since we use *strided convolutions*, the subsampling is carried out just after the convolution. The encoding and decoding biases are denoted with $b_{\ell,i}$ and $b'_{\ell,i}$, and we choose leaky ReLUs for the non-linearities :

$$\phi_\alpha(x) = \begin{cases} x, & \text{for } x \geq 0 \\ \alpha x, & \text{for } x < 0 \end{cases}, \tag{2}$$

with parameter $\alpha = 0.2$. Leaky ReLU non-linearities are commonly used in the literature [20, 12].

Thus, the output of a given encoding layer is given by

$$E_i^{l+1} = \phi_\alpha(E^l * w_{\ell,i} + b_{\ell,i}), \tag{3}$$

and similarly for the decoding layers (except for zero-padding upsampling prior to the convolution), with weights and biases w' and b', respectively. We have used an abuse of notation by not indicating the subsampling here, as this is carried out with the strided convolution.

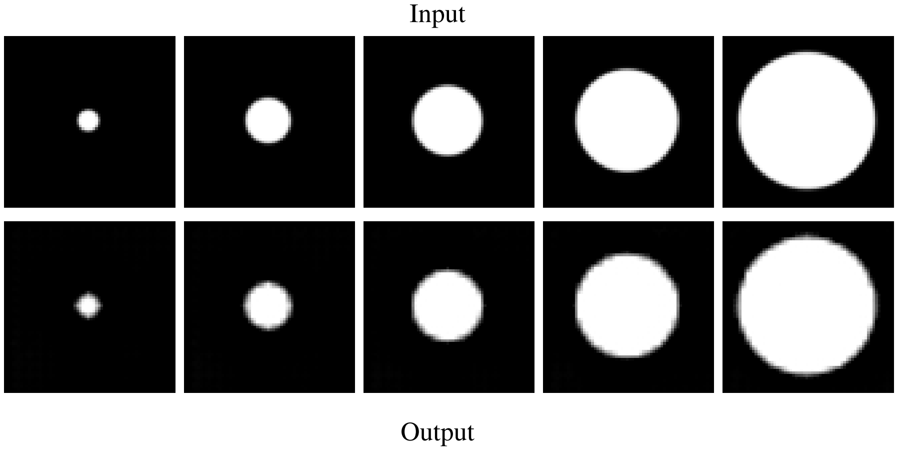


Fig. 2: **Result of autoencoding disks, with a latent space dimension of size** $d = 1$

We consider that the spatial support of the image $\Omega = [0, m-1] \times [0, m-1]$ is fixed throughout this work with $m = 64$, and also that the subsampling rate s is fixed to 2. In the encoder, subsampling is carried out until z achieves the size defined by the problem at hand. In the case of disks with varying radii, it is reasonable to assume that z will be a scalar. Thus, the number of layers in our encoder and decoder is not a free parameter. We set the support of all the convolutional filters in our network to 3×3. The architecture of our autoencoder remains the same throughout the paper, and is shown in Figure 1. We summarise our parameters in Table 1.

4 Autoencoding disks

4.1 Training dataset and preliminary autoencoder results

Our training set consists of grey-level images of centred disks. The radii of the disks are sampled following a uniform distribution $\mathcal{U}\left((0, \frac{m}{2})\right)$. We generate 3000 disks in the training set, so that the radius distribution is quite densely sampled. In order to create a continuous dataset, we slightly blur the disks with a Gaussian filter g_σ, so that $x_r = g_\sigma * \mathbb{1}_{B_r}$, where $\mathbb{1}_{B_r}$ is the indicator function of the ball of radius r. The exact manner in which this is done, using a Monte Carlo simulation, is explained in AppendixB.

Theoretically, an optimal encoder would only need one scalar to represent the image. Therefore the architecture in Figure 1 is set up to ensure a code size $d = 1$. After training, we observe experimentally that the network indeed learns to encode/decode disks correctly with a latent space size of $d = 1$. This can be seen in Figure 2.

We now proceed to see how the autoencoder actually works on a detailed level, starting with the encoding step.

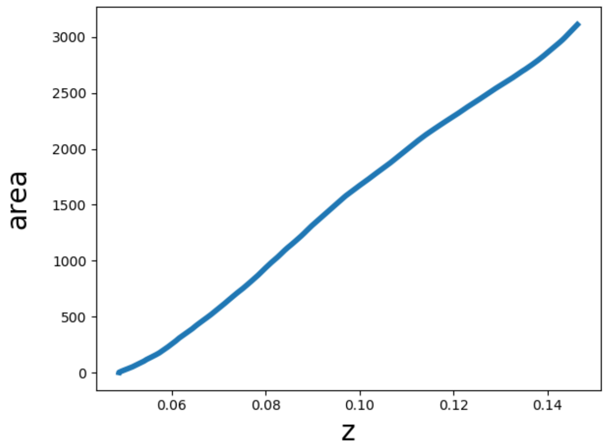


Fig. 3: **Investigating the latent space of disks**. We have plotted the areas (y-axis) of the input disks against their codes $z \in \mathbb{R}$ (x-axis) in the latent space. The code is linearly proportional to the disks' area. This behaviour is formalised in Proposition1.

4.2 Encoding a disk

Encoding a centred disk of a certain radius to a scalar z can be done in several ways, for example calculating the *area* or the *perimeter* of the disk. The empirical evidence given by our experiments points towards the first option, since z seems to represent the area and not the radius of the input disks (see Figure 3). This can be carried out in a variety of ways, the most obvious being a simple integration over the image.

Now, while we have described a simple solution which exists to the encoding network with disks (integration over the image), there is no guarantee that this is the *only* solution. Indeed there are probably many other valid representations of the disk, such as encoding the radius. However, these solutions are likely to be more complicated in terms of their filters, and we may also ask if encoding the area may be more desirable than other solutions. Another way of putting this is to find the simplest encoder network out of all the possible valid encoders. In deep learning, this is most often done using network *regularisation* [7]. The simplest regularisation technique is to minimise a norm of the filter weights, which aims to reduce the model complexity. Another regularisation approach of Rifai et al [22] is the "contractive autoencoder". This adds a penalisation of the Jacobian of the code z w.r.t the image $x : \|\nabla_x z\|_2^2$. Basically, this specifies that a small perturbation in the input image should result in a small perturbation in the code. It turns out that this regularisation leads to an encoder that indeed extracts the area of the disk. We formalise this result now.

Proposition 1 *[The contractive encoder encodes the area of a disk]*

Consider an encoder $E : \mathbb{R}^{m \times m} \rightarrow \mathbb{R}$, which has been presented with images of centred disks having radii uniformly distributed between 0 and 1 during training. Let R_{max} represent the largest radius observed in the dataset. The encoder which has *minimal contractive loss $\|\nabla_x E(x)\|_2^2$ and is non-constant is given by $E(x_r) = \gamma r^2$, where γ is a constant. In other words, the contractive encoder represents the image of a disk with its area.*

Proof : See Appendix A for the proof.

We also show in Appendix A further experimental evidence that the area is indeed extracted by the contractive encoder.

Before moving on, we highlight again that the encoder can learn any representation as long as there is an unambiguous link between each point in the data space and in the latent space. The contractive loss allows us to carry out the previous calculations, but this does not lead to the only valid representation.

4.3 Decoding a disk

A more difficult question is how does the autoencoder convert a scalar, z, to an output disk of a certain size (the decoding process) ? One approach to understanding the inner workings of autoencoders, and indeed any neural network, is to remove certain elements of the network and to see how it responds, otherwise known as an *ablation* study. We found that removing the *biases* of the autoencoder leads to very interesting observations. While the encoder is perfectly able to function without these biases (see previous section), this is not the case for the decoder. Figure 4 shows the results of this ablation. The decoder learns to spread the energy of z in the output according to a certain function g. Thus, the goal of the biases is to shift the intermediary (hidden layer) images such that a cut-off can be carried out to create a satisfactory decoding.

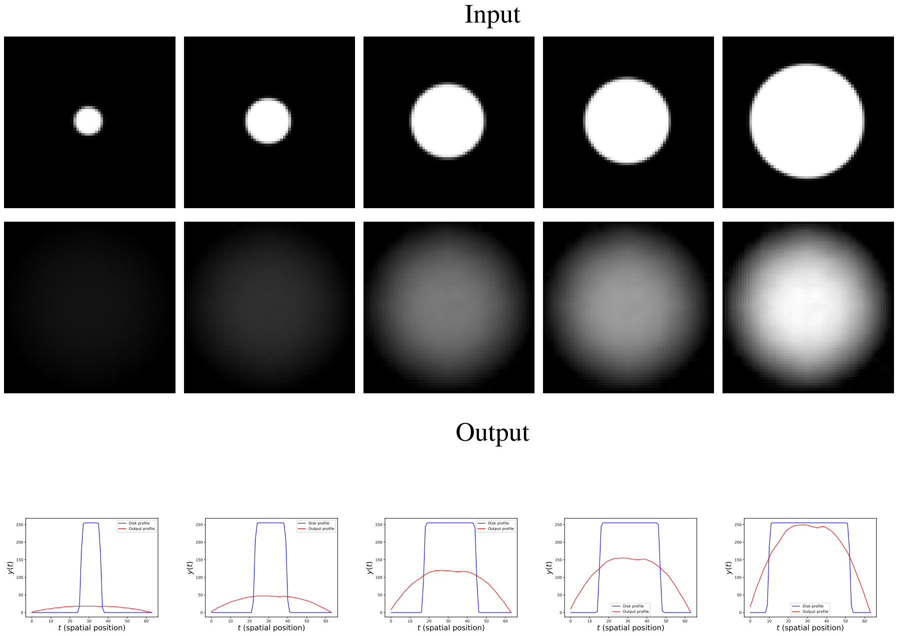
In order to analyse the inner mechanism of the decoder in more depth, we have investigated the behaviour of the decoder in this ablated case (without biases), where it is possible to describe the decoding process with great precision. In particular, we will derive an explicit form for the energy minimized by the network, for which a closed form solution can be found (see Appendix C), and we will show experimentally that the network indeed finds this solution. We first make a general observation about this configuration (without biases).

Proposition 2 *[Positive Multiplicative Action of the Decoder Without Bias]*
Consider a decoder, without biases $D(z) = D^L \circ \cdots \circ D^1(z)$, with $D^{\ell+1} = \phi_\alpha\left(U(D^\ell) * w'_{\ell_i}\right)$, where U stands for upsampling with zero-padding. In this case, the decoder acts multiplicatively on z, meaning that

$$\forall z, \forall \lambda \in \mathbb{R}^+,\ D(\lambda z) = \lambda D(z).$$

Proof : For a fixed z and for any $\lambda > 0$. We have

$$\begin{aligned} D^1(\lambda z) &= \phi_\alpha\left(U(\lambda z) * w'_\ell\right) \\ &= \max\left(\lambda(U(z) * w'_\ell), 0\right) + \alpha \min\left(\lambda(U(z) * w'_\ell), 0\right) \\ &= \lambda \max\left(U(z) * w'_\ell, 0\right) + \lambda\alpha \min\left(U(z) * w'_\ell, 0\right) \\ &= \lambda\phi_\alpha\left(U(z) * w'_\ell\right) = \lambda D^1(z). \end{aligned} \tag{4}$$



1D Profile, representing the cross-sections of the images above (input in blue, output in red)

Fig. 4: **Autoencoding of disks when the autoencoder is trained with no bias.** The first two rows are the input and output of disks when no bias is included in the network. The third row represents the 1D cross-section of these radially symmetric images. The autoencoder learns a function f which is multiplied by a constant scalar, $h(r)$, for each radius. This behaviour is formalised in Equation (5).

This reasoning can be applied successively to each layer up to the output y. When the code z is one dimensional, the decoder can be summarized as two linear functions, one for positive codes and a second one for the negative codes. However, in all our experiments, the autoencoder without bias has chosen to use only one possible sign for the code, resulting in a linear decoder. □

The profiles in Figure 4 suggest that a single function is learned, and that this function is multiplied by a factor depending on the radius. In light of Proposition 2, this means that the decoder has chosen a fixed sign for the code and that the decoder is linear. This can be expressed as

$$D(E(\mathbb{1}_{B_r}))(t) = h(r)f(t), \tag{5}$$

where t is a spatial variable and $r \in (0, \frac{m}{2}]$ is the radius of the disk. This is checked experimentally in Figure 13 in Appendix C. In this case, we can write the optimisation problem of the decoder as

$$\hat{f}, \hat{h} = \underset{f,h}{\operatorname{argmin}} \int_0^R \int_\Omega \left(h(r)f(t) - \mathbb{1}_{B_r}(t)\right)^2 dt\, dr, \tag{6}$$

where R is the maximum radius observed in the training set, $\Omega = [0, m-1] \times [0, m-1]$ is the image domain, and B_r is the disk of radius r. Note that we have expressed the minimisation problem for continuous functions f. In this case, we have the following proposition.

Proposition 3 (Decoding Energy for an autoencoder without Biases) *The decoding training problem of the autoencoder without biases has an optimal solution $\hat{f}$ that is radially symmetric and maximises the following energy:*

$$J(f) := \int_0^R \left(\int_0^r f(\rho)\, \rho \, d\rho \right)^2 dr, \tag{7}$$

under the (arbitrary) normalization $\|f\|_2^2 = 1$.

Proof : When f is fixed, the optimal h for Equation (6) is given by

$$\hat{h}(r) = \frac{\langle f, \mathbb{1}_{B_r} \rangle}{\|f\|_2^2}, \tag{8}$$

where $\langle f, \mathbb{1}_{B_r} \rangle = \int_\Omega f(t) \mathbb{1}_{B_r}(t)\ dt$. After replacing this in Equation (6), we find that

$$\hat{f} = \underset{f}{\operatorname{argmin}} \int_0^R -\frac{\langle f, \mathbb{1}_{B_r} \rangle^2}{\|f\|^2} dr = \underset{f}{\operatorname{argmin}} \int_0^R -\langle f, \mathbb{1}_{B_r} \rangle_2^2 \, dr, \tag{9}$$

where we have chosen the arbitrary normalisation $\|f\|_2^2 = 1$. The form of the last equation shows that the optimal solution is obviously radially symmetric[2]. Therefore, after a change of variables, the energy maximised by the decoder can be written as

$$\int_0^R \left(\int_0^r f(\rho)\, \rho \, d\rho \right)^2 dr =: J(f), \tag{10}$$

such that $\|f\|_2^2 = 1$. □

In Appendix C, we compare the numerical solution of this problem with the actual profile learned by the network, yielding a very close match. This result is enlightening, since it shows that the training process has achieved the optimal solution, in spite of the fact that the loss is non convex.

4.4 Generalisation and regularisation

As we have recalled in Section 2, many works have recently investigated the generative capacity of autoencoders or GANs. Nevertheless, it is not clear that these architectures truly invent or generalize some visual content. A simpler question is : to what extent is the network able to generalise in the case of the simple geometric

[2] If not, then consider its mean on every circle, which decreases the L^2 norm of f while maintaining the scalar product with any disk. We then can increase back the energy by dividing by this smaller L^2 norm according to $\|f\|_2 = 1$.

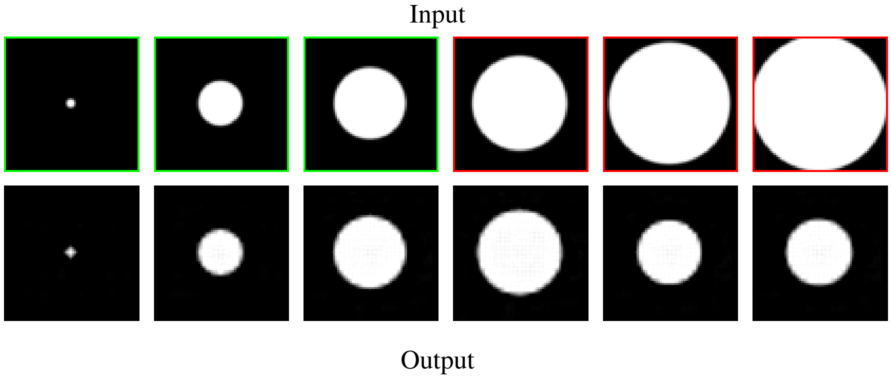


Fig. 5: **Autoencoding of disks with a database with limited radii.** The autoencoder is not able to extrapolate further than the largest observed radius. The images with a green border represent disks whose radii have been observed during training, while those in red have not been observed.

notion of size ? In this section, we address this issue in our restricted but interpretable case.

For this, we study the behaviour of our autoencoder when examples are removed from the training dataset. In Figure 5, we show the autoencoder result when the disks with radii above a certain threshold R are removed. The radii of the left three images (with a green border) are present in the training database, whereas the radii of the right three (red border) have not been observed. It is clear that the network lacks the capacity to extrapolate further than the radius R. Indeed, the autoencoder seems to project these disks onto smaller, observed, disks, rather than learning the abstraction of a disk.

Again by removing the biases from the network, we may explain why the autoencoder fails to extrapolate when a maximum radius R is imposed. In Appendix D, we show experimental evidence that in this situation, the autoencoder learns a function f whose support is restricted by the value of R, leading to the autoencoder's failure. However, a fair criticism of the previous experiment is simply that the network (and deep learning in general) is not designed to work on data which lie outside of the domain observed in the training data set. Nevertheless, it is reasonable to expect the network to be robust to holes *inside* the domain. Therefore, we have also analysed the behaviour of the autoencoder when we removed training datapoints whose disks' radii lie within a certain range, between 11 and 18 pixels (out of a total of 32). We then attempt to reconstruct these points in the test data. Figure 6 shows the results of this experiment failure. Once again, in the unknown regions the network is unable to recreate the input disks. Several explanations in the deep learning literature of this phenomenon, such as a high curvature of the underlying data manifold [7] (see page 521, or end of Section 14.6), noisy data or high intrinsic dimensionality of the data [3]. In our setting, *none of these explanations is sufficient*. Thus we conclude

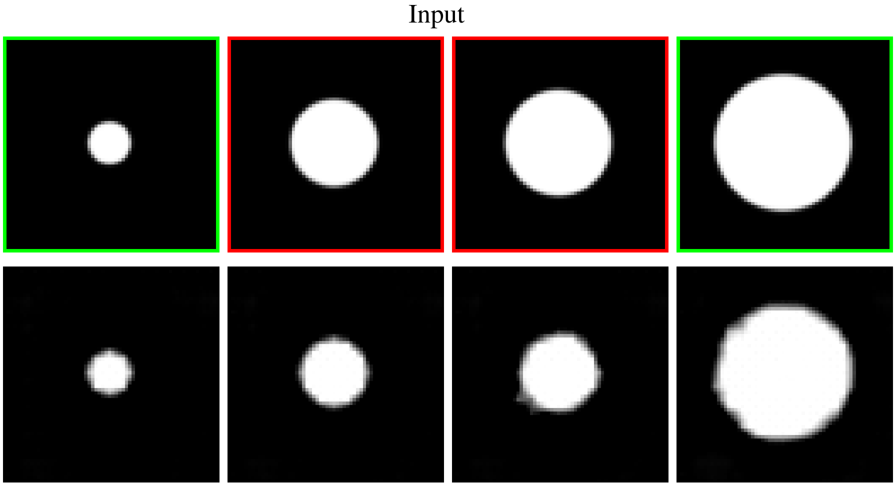


Fig. 6: **Input and output of our network when autoencoding examples of disks when the database contains a "hole".** Disks of radii between 11 and 18 pixels (out of 32) were not observed in the database. In green, the disks whose radii have been observed in the database, in red those which have not.

that, even in the simple setting of disks, the classic autoencoder cannot generalise correctly when a database contains holes.

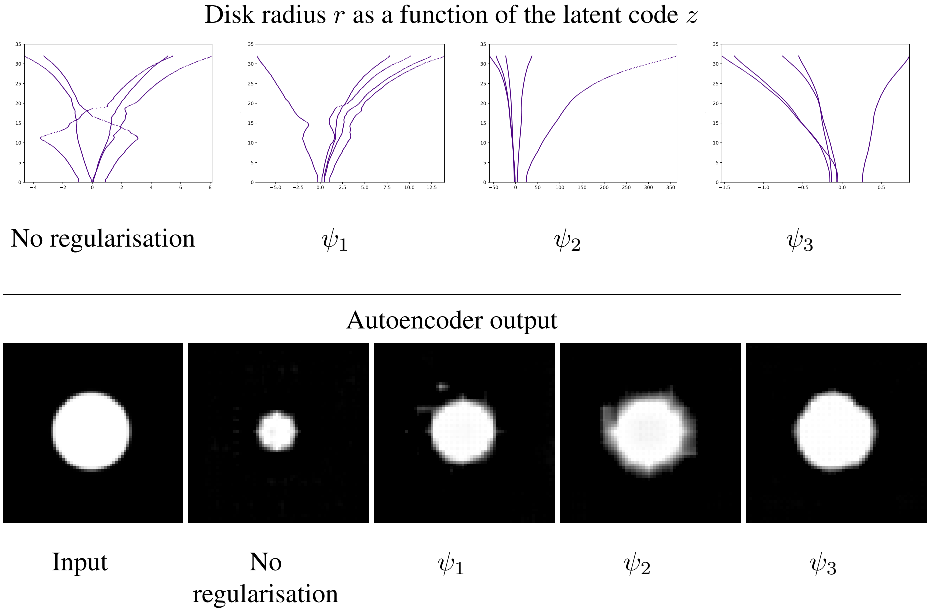
Consequently, this effect is clearly due to the gap between two different formulations of the loss of an autoencoder :

$$\mathcal{L}_1 = \mathbb{E}_{x \sim p_x} \|x - D(E(x))\|^2 \tag{11}$$

$$\mathcal{L}_2 = \mathbb{E}_{x \in \text{dataset}} \|x - D(E(x))\|^2. \tag{12}$$

The latter supposes that the dataset faithfully reflects the distribution p_x of images and is the empirical loss actually used in most of the literature. In our setting we are able to faithfully sample the true distribution p_x and study what happens when a certain part of the distribution is not well observed.

This behaviour is potentially problematic for applications which deal with more complex natural images, lying on a high-dimensional manifold, as these are very likely to contain such holes. We have therefore carried out the same experiments using the recent DCGAN approach of [20]. The visual results of their algorithm are displayed in Appendix E. We trained their network using a code size of $d = 1$ in order to ensure fair comparisons. The network fails to correctly autoencode the disks belonging to the unobserved region. Indeed, *GAN-type networks may not be very good at generalising data*, since their goal is to find a way to map the observed data to some predefined distribution, therefore there is no way to modify the latent space itself. This shows that the generalisation problem is likely to be ubiquitous, and indeed observed in more sophisticated networks, designed to learn natural images manifolds, even in the simple case of disks. We therefore believe that this issue deserves



Disk radius r as a function of the latent code z

No regularisation ψ_1 ψ_2 ψ_3

Autoencoder output

Input No regularisation ψ_1 ψ_2 ψ_3

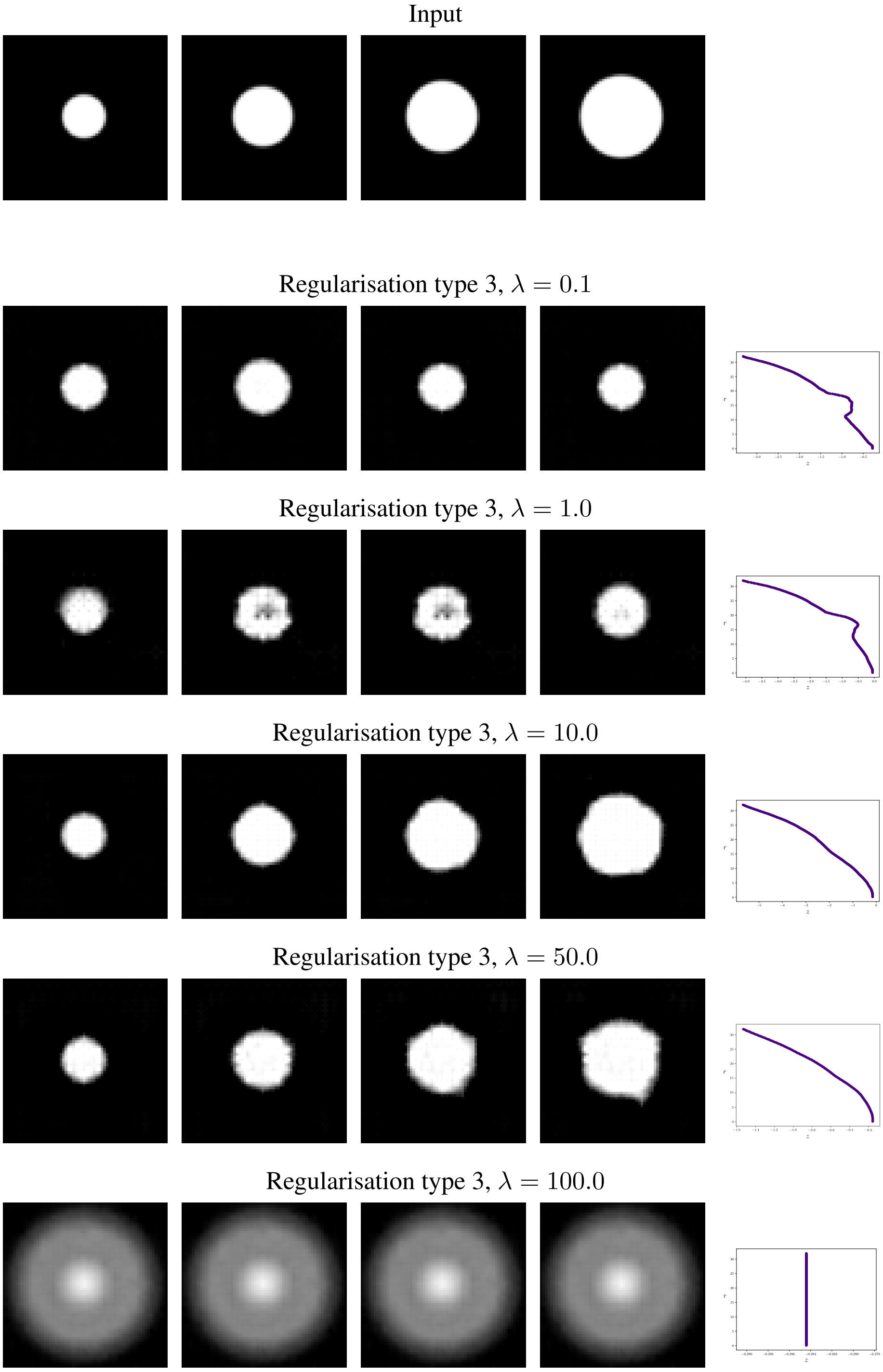
Fig. 7: **Result of different types of regularisation on autoencoding in an "unknown region" of the training database.** We have encoded/decoded a disk which was not observed in the training dataset. We show the results of four experiments: no regularisation, ℓ_2 regularisation in the latent space (ψ_1), ℓ_2 weight penalisation of the encoder and decoder (ψ_2) and ℓ_2 weight penalisation of the encoder only (ψ_3). In order to highlight the instability of the autoencoder without regularisation, we have carried out the same experiment five times, and shown the resulting latent spaces for each experiment. The latent spaces produced by a regularised autoencoder, and in particular types 2-3, are consistently smoother than the unregularised version, which can produce incoherent latent spaces, and thus incorrect outputs.

careful attention. Actually this experiment suggets that the capacity to generate new and simple geometrical shapes could be taken as a minimal requirement for a given architecture.

In order to address the problem, we now investigate several regularisation techniques whose goal is to aid the generalisation capacity of neural networks.

4.4.1 Regularisation

We would like to impose some structure on the latent space in order to interpolate correctly in the case of missing datapoints. This is often achieved via some sort of regularisation. This regularisation can come in many forms, such as imposing a certain distribution in the latent space, as in variational autoencoders [13], or by encouraging z to be sparse, as in sparse auto-encoders [21, 18]. In the present case, the former is not particularly useful, since a probabilistic approach will not encourage the latent



Input

Regularisation type 3, $\lambda = 0.1$

Regularisation type 3, $\lambda = 1.0$

Regularisation type 3, $\lambda = 10.0$

Regularisation type 3, $\lambda = 50.0$

Regularisation type 3, $\lambda = 100.0$

Fig. 8: **Effect of encoder regularisation on the generalisation capacity of the network**. Regularisation of the network with a varying value of λ, using the regularisation ψ_3 (encoder regularisation) described in Section 4.4

space to correctly interpolate. The latter regularisation does not apply, since we already have $d = 1$. Another commonly used approach is to impose an ℓ_2 penalisation of the weights of the filters in the network. The idea behind this bears some similarity to sparse regularisation; we wish for the latent space to be as simple as possible, and therefore hope to avoid over-fitting.

We have implemented several regularisation techniques on our network. Firstly, we attempt a simple regularisation of the latent space by requiring a locality-preservation property as suggested in [10,1,16], namely that the ℓ_2 distance between two images (x, x') be maintained in the latent space. This is done by randomly selecting a neighbour of each element in the training batch. Secondly, we regularise the weights of the encoder and/or the decoder (also known as weight decay).

Our training attempts to minimise the sum of the data term, $\|x - D(E(x))\|_2^2$, and a regularisation term $\lambda\psi(x, \theta)$, which can take one of the following forms:

- Type 1 : $\psi_1(x, x') = (\|x - x'\|_2^2 - \|E(x) - E(x')\|_2^2)^2$
- Type 2 : $\psi_2(\Theta_E, \Theta_D) = \sum_{\ell=1}^{L} \|w_{\cdot,\ell}\|_2^2 + \|w'_{\cdot,\ell}\|_2^2$
- Type 3 : $\psi_3(\Theta_E) = \sum_{\ell=1}^{L} \|w_{\cdot,\ell}\|_2^2$.

In Section 4.2, we used the contractive loss to derive Proposition 1 which showed that this loss encouraged the encoder to extract the area. We have not used this regularisation here, however, since it gives quite similar results to the Type 3 regularisation. This is coherent with the results from Rifai et al [22], who showed a formal link between the contractive regularisation and weight regularisation in the case of one hidden layer. Since the weight regularisation is a more practical alternative to the contractive regularisation, we have only experimented with the former here. Finally, we note that, given the very strong bottleneck of our architecture, the dropout regularisation technique does not make much sense here.

Figure 7 shows the results of these experiments. First of all, we observe that ψ_1 does not work satisfactorily. One interpretation of this is that the manifold in the training data is “discontinuous”, and therefore there are no close neighbours for the disks on the edge of the unobserved region. The second type of regularisation, minimising the ℓ_2 norm of the encoder and decoder weights, produces a latent space which appears smooth, however the final result is not of great quality. Finally, we observe that regularising the weights of the encoder (ψ_3) works particularly well, and that the resulting manifold is smooth and correctly represents the area of the disks. Consequently, this asymmetrical regularisation approach is to be encouraged in other applications of autoencoders. We show further results of this regularisation approach in Figure 8, when the regularisation parameter is varied. We see that increasing this parameter smooths the latent space, until λ becomes too great and the training fails.

At this point, we take the opportunity to note that the clear, marked effects seen with the different regularisation approaches are consistently observed in different training runs. This is due in large part to the controlled, simple setting of autoencoding with disks. Indeed, many other more sophisticated networks, especially GANs, are known to be very difficult to train [23], leading to unstable results or poor reproducibility. We hope that our approach can be of use to more high-level applications, and possibly serve as a sanity check to which these complicated networks should be submitted. Indeed, it is reasonable to assume that such networks should be able to perform well in simple situations before moving onto complicated data.

5 Encoding position in an autoencoder

We now move on to the analysis of our second geometric property : position. For this, we ask the following question : is it possible to encode the position of a simple one-hot vector (a discretised Dirac in other words) to a scalar, and if so, how ? A similar situation was investigated concurrently to our work by Liu et al. [17], who studied a network which projected images of randomly positioned squares to a position (a vector in $\mathbb{R}^2$), and then back again to the pixel space, with as small a loss as possible. Their opinion was that this was not possible, at least to a satisfactory degree, by training neural networks, which lead them to propose the CoordConv network layer.

In the following, we hand-craft a simple neural network which can achieve this in the forward direction : from a one-hot vector to the position. To simplify, we will analyse the 1-D case, that is to say the input lives in a one dimensional space.

Firstly, let us define some notation. We denote $x \in \mathbb{R}^n$ the input to the network, where n is the input dimension. We shall denote with $u^{(\ell)}$ the output of the ℓ-th layer of the neural network. We shall denote with φ the filter of our network. We shall consider the following hand-crafted filter :

$$\varphi = [1, 2, 1] \,. \tag{13}$$

Let us also suppose that subsampling factor is $s = 2$, and that it takes place at every even position (0, 2, 4 etc). We denote with $\mathcal{S}$ the subsampling operator. We do not use any non-linearities or biases in the network. Finally, we denote with E the whole linear neural network.

5.1 Some concrete examples

As a simple example, let use consider an input vector $x = [1, 0, 0, 0]$. After the first filtering and subsampling step, we have $u^{(1)} = [2, 0]$, and then $u^{(2)} = 4$. Similarly, if $x = [0, 0, 0, 1]$, then $u^{(2)} = 1$. Thus, with these two simple operations, it seems we can extract the non-zero position of a one-hot vector.

To take another example of the result of these operations, let us take a look at a similar result in $n = 8$. In Table 2, we can see the results for every possible 1-hot vector. Indeed, the network seems to extract the position of the one-hot vector.

5.2 Position encoding in the general case

Now, if we take the general case, $x \in \mathbb{R}^n$ with $n = 2^L$ and where L is the total number of layers, then the output of each layer $u^{(\ell)}$ can be written in terms of the convolution with the previous layer :

x	$[1,0,0,0,0,0,0,0]$	$[0,1,0,0,0,0,0,0]$	$[0,0,1,0,0,0,0,0]$	$[0,0,0,1,0,0,0,0]$
$u^{(1)}$	$[2,0,0,0]$	$[1,1,0,0]$	$[0,2,0,0]$	$[0,1,1,0]$
$u^{(2)}$	$[4,0]$	$[3,1]$	$[2,2]$	$[1,3]$
$u^{(3)}$	$[8]$	$[7]$	$[6]$	$[5]$
x	$[0,0,0,0,1,0,0,0]$	$[0,0,0,0,0,1,0,0]$	$[0,0,0,0,0,0,1,0]$	$[0,0,0,0,0,0,0,1]$
$u^{(1)}$	$[0,0,2,0]$	$[0,0,1,1]$	$[0,0,0,2]$	$[0,0,0,1]$
$u^{(2)}$	$[0,4]$	$[0,3]$	$[0,2]$	$[0,1]$
$u^{(3)}$	$[4]$	$[3]$	$[2]$	$[1]$

Table 2: Results of all possible one-hot vectors of size eight in the simple linear neural network described in Section 5

$$u^{(\ell)}(t) = \sum_{i \in \mathcal{A}} \varphi(i) u^{(\ell-1)}(st - i), \tag{14}$$

where $\mathcal{A}$ is defined as the support of the filter φ. In our case, $\mathcal{A} = \{-1, 0, 1\}$. Using an induction argument, we can show that the network E indeed extracts the position of the one-hot input vector. More precisely, as we have seen in Section 5.1, the network extracts the position in an inverted order, that is to say $n - a$, if a is the postion of the non-zero element of x and if we number the elements of x from x_0 to x_{2^L-1}.

Proposition 4 (The linear neural network E extracts the position of a Dirac input) *Consider the neural network E described earlier in this section, and a one-hot input vector $x \in \mathbb{R}^n$, with $n = 2^L$ and where $(x_i), i \in [0, \dots, n-1]$ denotes the i^{th} element of x. If a is the position of the non-zero element of x, then $E(x) = n - a$. In other words, the network E extracts the (inverted) position of the non-zero element.*

Proof :

We prove this by induction over the number of layers in the network.

One hidden layer This is easy to verify for a network with one hidden layer. Indeed, if the input $x \in \mathbb{R}^2$ contains a 1 at the first (0^{th}) position, then the network output is $2*1 = 2$. If x contains a 1 at the second position, then the network output is $1*1 = 1$. Thus, the property is true for the case of one hidden layer.

L hidden layers Let us suppose that the network contains L hidden layers, and extracts the non-zero position in reverse order, that is to say $u^{(L)} = 2^L - a$, where a is the non-zero position in x. Since the output of the network is a positive linear combination of the input vector with fixed coefficients, and the property holds for any a, we can rewrite the output as

$$E(x) = \sum_{i=0}^{2^L-1} (2^L - i)x_i. \tag{15}$$

Now let us suppose that we add a layer above the input layer, so that the network now has $L+1$ hidden layers and the input x now belongs to $\mathbb{R}^{2^{L+1}}$, and the previous x is now $u^{(1)}$. We can determine the output of the network using Equation (15). There are three cases to distinguish between.

Suppose first that a is an even position, so that $\exists k \in \mathbb{N}, a = 2k$. Thus, using Equation (15), we have that

$$\begin{aligned} E(x) &= \sum_{i=0}^{2^L-1} (2^L - i)u^{(1)}(i) \\ &= (2^L - k).2 \\ &= 2^{(L+1)} - 2k. \end{aligned} \tag{16}$$

Thus, we find that the network extracts the correct "inverted-order" position, with $a = 2k$.

Let us suppose now that $a = 2k+1$. In this case, we have

$$\begin{aligned} E(x) &= (2^L - k).1 + (2^L - (k+1)).1 \\ &= 2^{(L+1)} - (2k+1). \end{aligned} \tag{17}$$

Again, the network correctly identifies the position $a = 2k+1$.

Finally, there is a special case, where $a = 2^{(L+1)} - 1 = 2k+1$, with $k = 2^L - 1$ (at the end of the vector x). In this case, we have

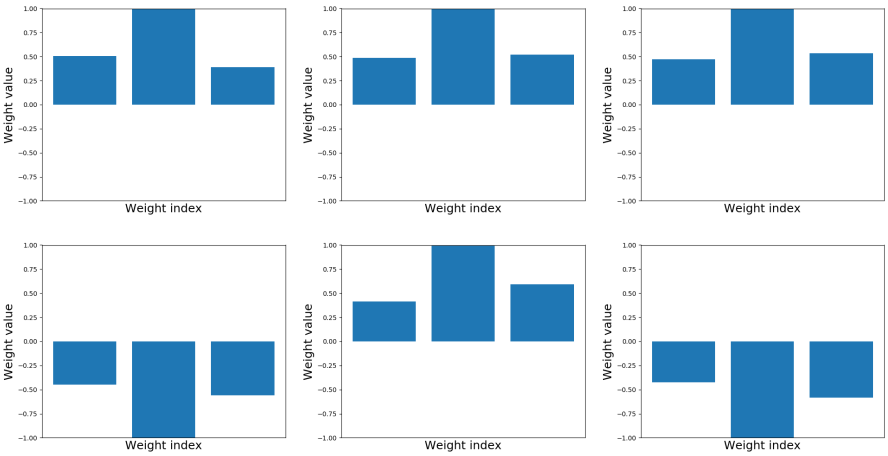
$$\begin{aligned} E(x) &= (2^L - k).1 \\ &= 2^L - (2^L - 1) \\ &= 1. \end{aligned} \tag{18}$$

Thus, in the extreme case of $a = 2^{(L+1)} - 1$, E still extracts the inverted-order position. Thus, we have proved that the network E extracts the position k of the non-zero element of a one-hot input vector. □

Furthermore, obviously any variant $b\varphi$, with $b \in \mathbb{R}^*$ also extracts the position multiplied by b^L, since the encoder described in Proposition 4 is a linear function. Finally, we note that our proof relies on the fact that the subsampling factor is $s = 2$. While this proof only directly applies to the example of one-hot pixels, it can provide a useful rule-of-thumb for designing networks which need to deal with position.

5.3 Experimental results

We now present experimental evidence that training a neural network with the above encoder leads to the previously exhibited hand-crafted weights in practice. Please



Weights for each layer of the encoder network E

Fig. 9: **Weights of position encoder network**. We show the weights found by the encoder network E, trained to extract the position. These weights agree with our theoretical prediction in Section 5.2.

note that our goal here is to confirm that the weights which we have constructed are indeed correct. Therefore, in this experiment, we have imposed two main restrictions. Firstly, we construct our encoder to have one filter per layer. We do not allow for many filters, since they would become uninterpretable due to the various possible combinations of these filters. Secondly, we train the encoder to predict the position a of the one-hot vector. The loss function is therefore simply the mean squared error loss between the predicted position and the true position. Indeed, while we have described a mechanism whereby a convolutional network can extract the position, we have no guarantee that this is the *only* solution. Therefore we use this restricted experimental setting in order to improve interpretability.

In Figure 9, we show the weights found by stochastic gradient descent training. They fit the handcrafted weights in Equation (13) remarkably well. At every layer, we have the handcrafted weights, multiplied by $+1$ or -1. This obviously makes no difference to the final result of the network, since it can flip the sign at any point before the latent code.

5.4 Decoding position

We now show that it is also possible to perform this inverse operation, in other words, starting from a position z, output a 1d signal which approximates a delta at position z. To do this, we use a triangular approximation of the Dirac delta. For a Dirac posi-

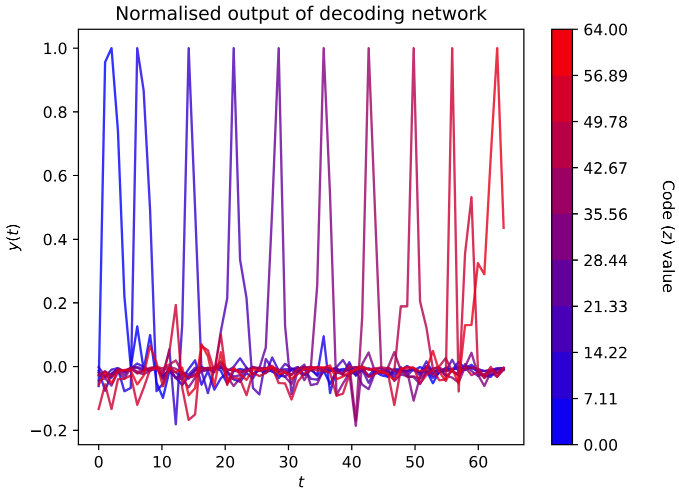


Fig. 10: **Normalised output of decoding of a postion to a 1D Dirac**. We show the decoding of increasing values of z. We have normalised each output y, to highlight the position of the Dirac.

tioned at $a \in [0, n]$, this approximation is :

$$y_a(t) = \begin{cases} 1 - |t - a|, \; if \; |t - a| < 1 \\ 0, \; \text{otherwise} \end{cases} , \tag{19}$$

It is important to note that the continuous sampling of the parameter space (the position of the Dirac) and subsequent discretisation is crucial to obtaining successful decoding (as it was in the case of disks). Indeed, we also tried to use the approach described in the case of the encoder, that is to say that the Dirac is a one-hot vector at the position a, similar to the experiments described in the "CoordConv" network [17]. In this case, the database is limited, and the decoding is not successful. In particular, interpolating between known datapoints is quite unstable. Sampling a continuous parameter a and choosing an appropriate discretisation solves this problem.

The decoding network was chosen in a similar manner to the case of disks 4.3, with 1D convolutions of size 3, biases and leaky ReLU non-linearities. The filter depths chosen were the same as in the case of disks (see Table 1), with an output signal size of $n = 64$. The results of the decoding can be seen in Figure 10.

In this Section, we have described a hand-crafted filter which, when coupled with subsampling, can achieve perfect encoding of the position of a Dirac input signal. We show that a network with an appropriate architecture indeed finds this filter during training. Secondly, we have shown experimentally that decoding is also possible as long as the latent space is sampled in a continuous manner and the corresponding signals are appropriately discretised. This highlights the necessity of correctly sampling the input data.

6 Conclusion and future work

We have investigated in detail the specific mechanisms which allow autoencoders to encode and decode two fundamental image properties : size and position. The first property is studied via the specific case of binary images containing disks. We first showed that the architecture we proposed was indeed capable of projecting to and from a latent space of size 1. We have shown that the encoder works by integrating over disk, and so the code z represents the area of the disk. In the case where the autoencoder is trained with no bias, the decoder learns a single function which is multiplied by a scalar that is dependent on the size of the disk. Furthermore, we have shown that the optimal function is indeed learned by our network during training. This indicates that the decoder works by multiplying and thresholding this function to produce a final binary image of a disk. We have also illustrated certain limitations of the autoencoder with respect to generalisation when datapoints are missing in the training set. This is potentially problematic for higher-level applications, whose data have higher intrinsic dimensionality and therefore are more likely to include such holes. We identify a regularisation approach which is able to overcome this problem particularly well. This regularisation is asymmetrical as it consists of regularizing the encoder while leaving more freedom to the decoder.

Secondly, we have analysed how an autoencoder is able to process position in input data. We do this by studying the case of vectors containing Dirac delta functions (or “one-hot vectors”). We identify a hand-crafted convolutional filter and prove that by using convolutions with this filter and subsampling operations, an encoding network is able to perfectly encode the position of the Dirac delta function. Furthermore, we show experimentally that this filter is indeed learned by an encoding network during training. Finally, we show that a decoding network is able to decode a scalar position and produce the desired Dirac delta function.

We believe that it is important to study generative networks in simple cases in order to properly understand how they work, so that, *in fine*, we can propose architectures that are able to produce increasingly high-level and complex images in a reliable manner and with fine control over the results (for example interpolating in the latent space). An important future goal is to extend the theoretical analyses obtained to increasingly complex visual objects, in order to understand whether the same mechanisms remain in place. We have experimented with other simple geometric objects such as squares and ellipses, with similar results in an optimal code size. Another question is how the decoder works with the biases included. This requires a careful study of the different non-linearity activations as the radius increases. Finally, we are obviously interested in how these networks process other fundamental image properties, such as rotation or colour. Some recent interesting work on increasing independence in the latent codes’ elements [14] could be useful in this respect.

Acknowledgements This work was funded by the Agence Nationale de la Recherche (ANR), in the MIRIAM project

References

1. Alain, G., Bengio, Y.: What regularized auto-encoders learn from the data-generating distribution. The Journal of Machine Learning Research **15**(1), 3563–3593 (2014)
2. Ballé, J., Laparra, V., Simoncelli, E.P.: End-to-end optimized image compression. arXiv preprint arXiv:1611.01704 (2016)
3. Bengio, Y., Monperrus, M.: Non-local manifold tangent learning. Advances in Neural Information Processing Systems (2005)
4. Bourlard, H., Kamp, Y.: Auto-association by multilayer perceptrons and singular value decomposition. Biological cybernetics **59**(4), 291–294 (1988)
5. Choi, Y., Choi, M., Kim, M., Ha, J.W., Kim, S., Choo, J.: Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. arXiv preprint **1711** (2017)
6. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (2011)
7. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. MIT press (2016)
8. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in neural information processing systems, pp. 2672–2680 (2014)
9. Ha, D., Eck, D.: A neural representation of sketch drawings. arXiv preprint arXiv:1704.03477 (2017)
10. Hadsell, R., Chopra, S., LeCun, Y.: Dimensionality reduction by learning an invariant mapping. In: IEEE Conference on Computer Vision and Pattern Recognition (2006). DOI 10.1109/CVPR.2006.100
11. Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. arXiv preprint (2017)
12. Karras, T., Aila, T., Laine, S., Lehtinen, J.: Progressive growing of gans for improved quality, stability, and variation. arXiv preprint arXiv:1710.10196 (2017)
13. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. In: International Conference on Learning Representations (2014)
14. Lample, G., Zeghidour, N., Usunier, N., Bordes, A., Denoyer, L., et al.: Fader networks: Manipulating images by sliding attributes. In: Advances in Neural Information Processing Systems, pp. 5967–5976 (2017)
15. LeCun, Y.: Learning processes in an asymmetric threshold network. Ph.D. thesis, Paris VI (1987)
16. Liao, Y., Wang, Y., Liu, Y.: Graph regularized auto-encoders for image representation. IEEE Transactions on Image Processing **26**(6), 2839–2852 (2017)
17. Liu, R., Lehman, J., Molino, P., Such, F.P., Frank, E., Sergeev, A., Yosinski, J.: An intriguing failing of convolutional neural networks and the coordconv solution. arXiv preprint arXiv:1807.03247 (2018)
18. Makhzani, A., Frey, B.: K-sparse autoencoders. arXiv preprint arXiv:1312.5663 (2013)
19. Metz, L., Poole, B., Pfau, D., Sohl-Dickstein, J.: Unrolled generative adversarial networks. arXiv preprint arXiv:1611.02163 (2016)
20. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434 (2015)
21. Ranzato, M., Boureau, Y., LeCun, Y.: Sparse feature learning for deep belief networks. In: Conference on Neural Information Processing Systems (2007)
22. Rifai, S., Vincent, P., Muller, X., Glorot, X., Bengio, Y.: Contractive auto-encoders: Explicit invariance during feature extraction. In: Proceedings of the 28th international conference on machine learning (2011)
23. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X.: Improved techniques for training gans. In: Advances in Neural Information Processing Systems, pp. 2234–2242 (2016)
24. Upchurch, P., Gardner, J.R., Pleiss, G., Pless, R., Snavely, N., Bala, K., Weinberger, K.Q.: Deep feature interpolation for image content changes. In: CVPR, vol. 1, p. 3 (2017)
25. Zhu, J.Y., Krähenbühl, P., Shechtman, E., Efros, A.A.: Generative visual manipulation on the natural image manifold. In: European Conference on Computer Vision (2016)

Appendix A Contractive encoders learn the area of disks

We study the encoder part of an autoencoder that takes an image and outputs a one-dimensional feature. We show that, with a simple constraint that the output of the encoder is not constant and in the absence of any other loss than the contractive loss, the feature is merely the area of the disk presented to the encoder.

We refer to the input image as x_r, where r is the radius of the disk present in the image (one disk for each image, and each disk centred). In this simple setting we will seek to find a function $z : L^2(\Omega) \to \mathbb{R}$ that stands for the encoder E, where Ω is the support of the images. . The loss associated with a contractive auto-encoder [22] is

$$\mathcal{L}(z) = \sum_{r=0}^{R_{max}} \|\nabla z(x_r)\|^2, \tag{20}$$

where $\nabla z \in L^2(\Omega)$ stands for the gradient of the latent z with respect to the input image, when the input image is x_r (it is an image). R_{max} is the maximum radius observed in the dataset, which we normalise to 1. Although the parameter of a loss is typically the set of parameters θ of a network and is usually written $\mathcal{L}(\theta)$, here we minimize among all possible encoders z simulating an infinite capacity of the the encoder hence the notation $\mathcal{L}(z)$.

We can take a continuous proxy for this loss and write

$$L(z) = \int_0^1 \|\nabla z(x_r)\|_2^2 dr, \tag{21}$$

Note the integration against the simple measure dr reflects the fact that the distribution of the radii is uniform. In anticipation of the derivations ahead we suppose that the encoder function is smooth and that the edges of the shapes are also smooth. We will investigate what happens when the shapes become infinitely sharp after. We can express this by

$$x_r(p_x, p_y) = \varphi\left(\frac{\sqrt{p_x^2 + p_y^2} - r}{\sigma}\right) = \varphi_\sigma\left(\sqrt{p_x^2 + p_y^2} - r\right), \tag{22}$$

where $p = (p_x, p_y)$ is a position, φ is some smooth real function that is equal to 1 before -1, 0 after 1 (think of a simplified tanh function) and σ is a scaling factor. When σ goes to zero we will be in the case of sharp edges. Other smooth representations of a disk are possible, for example $x_r(p_x, p_y) = (\mathbb{1}_{B_r} * g_\sigma)(p_x, p_y)$, where $\mathbb{1}_{B_r}$ is the indicator function of the ball of radius r, as used in our experiments, and when σ goes to zero we are back to sharp edges again. We will stick to the representation in Equation (22) since it simplifies our calculations further on, in particular in Section A.1.

To avoid trivial cases, we also require our encoder not to be constant.[3] Once scaled, this constraint can be written

[3] This obviously cannot happen in the case of a full autoencoder, but we must impose it when studying the encoder only.

$$1 = z(x_1) - z(x_0) = \int_0^1 \frac{\partial z}{\partial r} dr = \int_0^1 < \nabla z | \frac{\partial x_r}{\partial r} > dr, \tag{23}$$

the last equality being the chain rule. Let us denote

$$h_r(p) := \frac{\partial x_r}{\partial r}(p). \tag{24}$$

Now our problem boils down to

$$\begin{array}{ll} \text{Minimise :} & \int_0^1 \|\nabla z(x_r)\|^2 dr \\ \text{Under the constraint :} & \int_0^1 \langle \nabla z(x_r) | h_r \rangle \, dr = 1 \end{array} \tag{25}$$

The minimization being performed among all possible z functions that are smooth enough to have a gradient with respect to its input x.

For a fixed r, among all $\nabla z(x_r)$ satisfying

$$\langle \nabla z(x_r) | h_r \rangle = C(r)$$

for some constant $C(r)$, the one with minimal $\|\nabla z(x_r)\|$ is of the form $c(r)h_r$. To see this, write $\nabla z(x_r) = \beta h_r + h_r^{\perp}$, a decomposition of $\nabla z(x_r)$ on $\text{Vect}(h_r)$ and its orthogonal space (in $L^2(\Omega)$). Hence, we can decrease the quantity to minimise in Equation (25) without changing the constraint by projecting $\nabla z(x_r)$ on $\text{Vect}(h_r)$. Thus, we can make the assumption that our solution z is such that

$$\nabla z(x_r) = c(r) h_r, \tag{26}$$

and we are reduced to finding a single function c that satisfies:

$$\begin{array}{ll} \text{Minimize :} & \int_0^1 c(r)^2 H_2(r) dr \\ \text{Under the constraint :} & \int_0^1 c(r) H_2(r) dr = 1, \end{array} \tag{27}$$

where

$$H_2(r) = \int\!\!\int h_r(p_x, p_y)^2 dp_x dp_y \tag{28}$$

Let us consider a small perturbation of the solution, $c + \epsilon\delta$, for some smooth function δ which satisfies $\int_0^1 \delta(r) H_2(r) dr = 0$ (the derivative of the constraint). Then, we have

$$\frac{d}{d\epsilon}\left(\int_0^1 (c(r) + \epsilon\delta(r))^2 H_2(r) dr \right) = \int_0^1 \left(2c(r)\delta(r) + 2\epsilon\delta(r)^2\right) H_2(r) dr. \tag{29}$$

If we take the limit when $\epsilon \rightarrow 0$, we have the condition

$$\int_0^1 c(r)\delta(r) H_2(r) = 0 \tag{30}$$

The solution of the system (27) is $c(r) = C$ for C some constant, since the only function $c(r)$ that satisfies Equation (30) for any valid increment δ is a constant one.

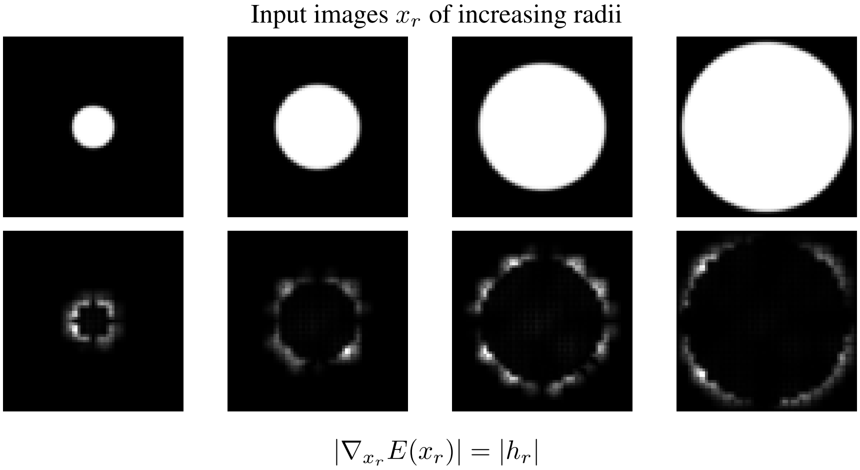


Fig. 11: **Absolute value of the gradient of the code z with respect to x_r.** We verify that in the case of a contractive encoder, the gradient of the code z of the disk image x_r, with respect to the image itself, is indeed concentrated on a circle of radius r. This behaviour is important to show that the contractive encoder indeed extracts the area of the disk.

Indeed, we have the two conditions $\delta \in \mathrm{Vect}(H_2)^\perp$ and $\langle cH_2, \delta \rangle = 0$. This means that $cH_2 \in \left(\mathrm{Vect}(H_2)^\perp\right)^\perp = \mathrm{Vect}(H_2)$.

Finally, when σ, the edge width goes to zero the function h_r tends to be concentrated on a circle of radius r (see next section A.1) and a value that is almost constant over the range of r. Roughly speaking this gives

$$H_2(r) = 2\pi r \alpha. \tag{31}$$

For the sake of completeness, we have verified experimentally that the function h_r is indeed concentrated on a circle of radius r. These results can be seen in Figure 11.

Finally, by integrating, we have

$$z(r) = \int_0^r \frac{dz}{d\rho} d\rho = \int_0^r c(\rho) H_2(\rho) d\rho = \gamma r^2, \tag{32}$$

where γ is some constant.

A.1 Infinitely thin edges

Here we show our claim when the edge width goes to 0, $z(r)$ is indeed proportional to the disk area (Equation (31)). We do this with the model described in Equation (22).

$$x_r(p_x, p_y) = \varphi \left(\frac{\sqrt{p_x^2 + p_y^2} - r}{\sigma} \right) = \varphi_\sigma \left(\sqrt{p_x^2 + p_y^2} - r \right), \tag{33}$$

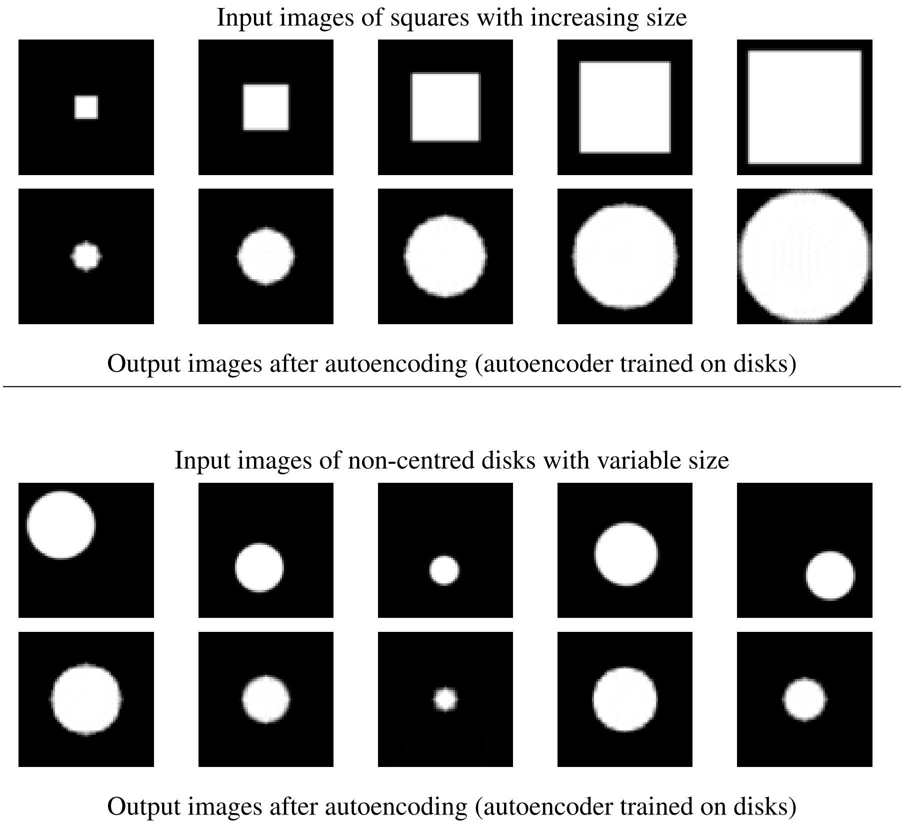

Input images of squares with increasing size
Output images after autoencoding (autoencoder trained on disks)
Input images of non-centred disks with variable size
Output images after autoencoding (autoencoder trained on disks)

Fig. 12: **Output of an autoencoder trained on disks, applied to squares and non-centred disks during testing**. The autoencoder indeed extracts the area of the object, regardless of its shape or position. Since it was trained on disks, it outputs the disks with a similar area to the objects observed during testing. Note that the autoencoder does not extract position, since it was trained on centred disks.

where φ is some smooth function that is equal to 1 before -1, 0 after 1.

In this case we have

$$\frac{\partial x_r}{\partial r}(p_x, p_y) = -\varphi'_\sigma(\sqrt{p_x^2 + p_y^2} - r). \tag{34}$$

The support of φ'_σ is $[-\sigma, \sigma]$. This function is radial and we are interested in computing (28), which gives

$$H_2(r) = \int_{r-\sigma}^{r+\sigma} 2\pi u \left(\varphi'_\sigma(u-r)\right)^2 du \tag{35}$$

with the variable u being $\sqrt{p_x^2 + p_y^2}$.

For $r \geq \sigma$ we have the following simple inequalities

$$2\pi(r-\sigma)\sigma^2 C \leq H_2(r) \leq 2\pi(r+\sigma)\sigma^2 C \tag{36}$$

where $C = \int \varphi'^2(t)dt$. This confirms the behavior of $H_2(r)$ as being merely proportional to r.

More precisely we obtain (by integration as in (32))

$$\gamma(r^2 - \sigma r) \leq z(r) \leq \gamma(r^2 - \sigma r), \tag{37}$$

which is the announced behavior for z. □

A.2 Experimental results

To further test this behaviour experimentally, we have used our contractive autoencoder trained on disks, and applied it to a test set of images with squares and non-centred disks. In Figure 12, it can be seen that the encoder indeed extracts the area of these objects, and then outputs the disk with the closest area (since it has been trained on a disk database). This further confirms that the encoder is indeed extracting the area.

Appendix B Creating the disk dataset

We wish to create a dataset which contains images of centred disks. Since the autoencoder must project each image to a continuous scalar, it makes sense to generate the disks with a continous parameter r, and that the disks also be "continuous" in some sense (each different value of r should produce a different disk. For this, as we mentioned in Section 4.1, we create the training images x_r as

$$x_r = g_\sigma * \mathbb{1}_{\mathbb{B}_r}, \tag{38}$$

where $\mathbb{1}_{\mathbb{B}_r}$ is the indicator function of the ball of radius r, and g_σ is a Gaussian kernel with variance σ. In practical terms, we carry this out using a Monte Carlo simulation to approximate the result of the convolution of an indicator function with a disk. Indeed, let $\xi_{i,i=1\dots N}$ be a sequence of independently and identically distributed (iid) random variables, with $\xi_i \sim \mathcal{N}(0,\sigma)$. Each pixel at position t is evaluated as

$$x_r(t) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}_{B_r}(\xi_i). \tag{39}$$

According to the law of large numbers, this tends to the exact value of $g_\sigma * \mathbb{1}_{\mathbb{B}_r}$, and gives a method of producing a continuous dataset.

While other approaches are available (evaluating the convolution in the Fourier domain, for example), this is simple to implement and generalises to any shape which we can parametrise. We also note that the large majority of deep learning synthesis papers suppose that the data lie on some manifold, but this hypothesis is never checked. In our case, we explicitly sample the data in a smooth space.

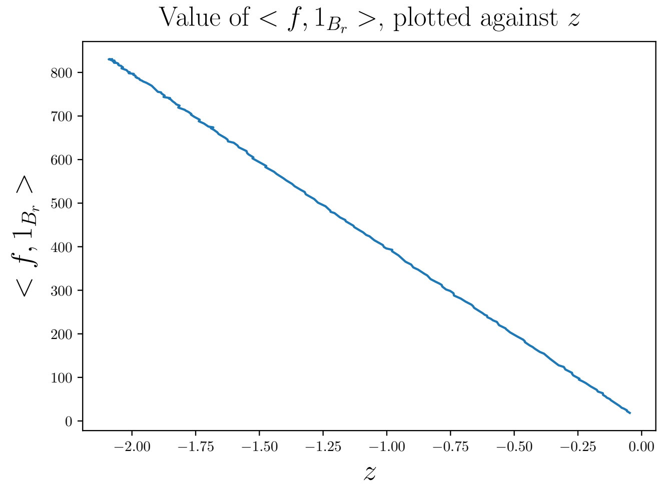


Fig. 13: **Verification of the theoretical derivations that use the hypothesis that $y(t, r) = h(r)f(t)$ for decoding, in the case where the autoencoder contains no bias.**. We have plotted z against the theoretically optimal value of h ($C \langle f, \mathbb{1}_{B_r} \rangle$, where C is some constant accounting for the arbitrary normalization of f). This experimental sanity check confirms our theoretical derivations.

Appendix C Decoding of a disk (network with no biases)

During the training of the autoencoder for the case of disks (with no bias in the autoencoder), the objective of the decoder is to convert a scalar into the image of a disk with the ℓ_2 distance as a metric. Given the profiles of the output of the autoencoder, we have made the hypothesis that the decoder approximates a disk of radius r with a function $y(t; r) := D(E(\mathbb{1}_{B_r})) = h(r)f(t)$, where f is a continuous function. We show that this is true experimentally in Figure 13 by determining f experimentally by taking the average of all output profiles, and then comparing our code z against its theoretically optimal value $\langle f, \mathbb{1}_{B_r} \rangle$. We see that they are the same up to a multiplicative constant C.

We now compare the numerical optimisation of the energy in Equation (7) using a gradient descent approach with the profile obtained by the autoencoder without biases. The resulting comparison can be seen in Figure 14. One can also derive a closed form solution of Equation (7) by means of the Euler-Lagrange equation and see that the optimal f for Equation (7) is the solution of the differential equation $y'' = -kty$ with initial state $(y, y') = (1, 0)$, where k is a free positive constant that accommodates for the position of the first zero of y. This gives a closed form of the f in terms of Airy functions.

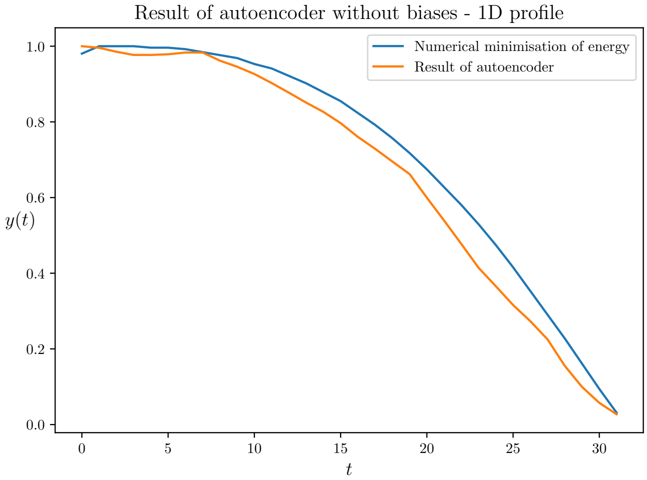


Fig. 14: **Comparison of the empirical function f of the autoencoder without biases with the numerical minimisation of Equation** (7)**.** We have determined the empirical function f of the autoencoder and compared it with the minimisation of Equation (7). The resulting profiles are similar, showing that the autoencoder indeed succeeds in minimising this energy.

Appendix D Autoencoding disks with a database with a limited observed radius (network with no biases)

In Figure 15, we see the grey-levels of the input/output of an autoencoder trained (without biases) on a restricted database, that is to say a database whose disks have a maximum radius R which is smaller than the image width. We have used $R = 18$ for these experiments. We see that the decoder learns a useful function f which only extends to this maximum radius. Beyond this radius, another function is used corresponding to the other sign of codes (see proposition 2) that is not tuned.

Appendix E Autoencoding disks with a DCGAN [20]

In Figure 16, we show the autoencoding results of the DCGAN network of Radford et al. We trained their network with a code size of $d = 1$. As can be seen, the DCGAN learns to force the training data to a predefined distribution, which cannot be modified during training (contrary to the autoencoder). Thus the network fails to correctly autoencode disks in the missing radius region which has not been observed in the training database.

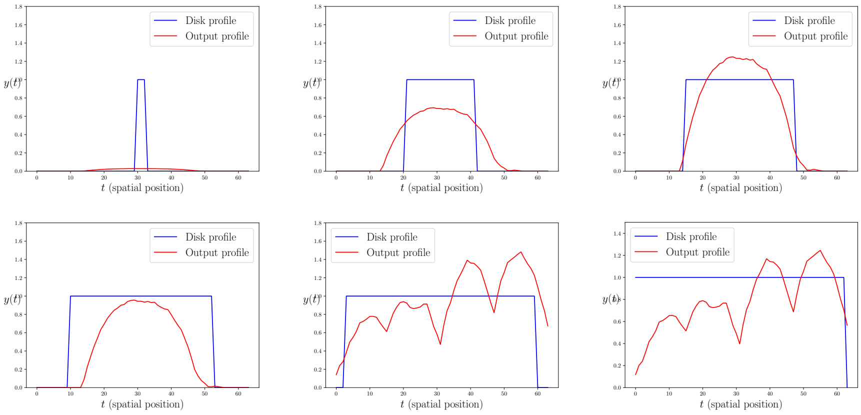


Fig. 15: **Profile of the encoding/decoding of centred disks, with a restricted database**. The decoder learns a profile f which only extends to the largest observed radius $R = 18$. Beyond this radius, another profile is learned that has is obviously not tuned to any data.

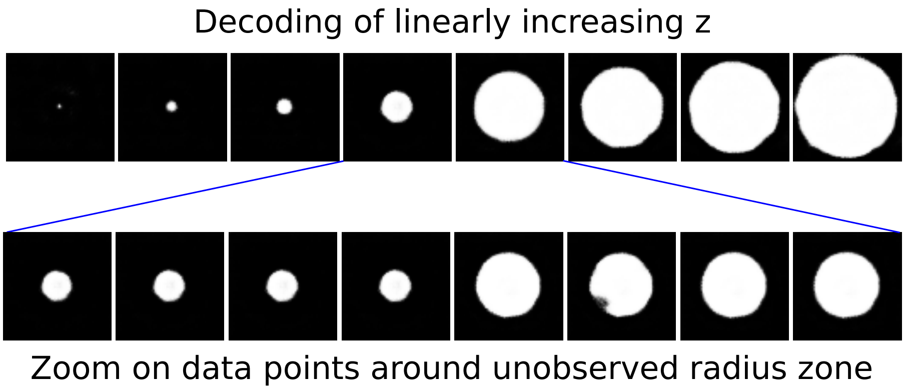

Decoding of linearly increasing z
Zoom on data points around unobserved radius zone

Fig. 16: **Output of the DCGAN of Radford et al.[20] ("IGAN") for disks when the database is missing disks of certain radii (11-18 pixels).** We can see that the DCGAN is not capable of reconstructing the disks which were not obeserved in the training dataset. This is a clear problem for generalisation. in the second we zoom on the datapoints around the radius zone which is unobserved in the training dataset.