Understanding and Organising the Latent Space of Autoencoders

Alasdair Newson

Télécom ParisTech
alasdair.newson@telecom-paristech.fr

6 February, 2020

Collaborators

- This work was carried out in collaboration with the following colleagues

Andrés Almansa
(Université Paris Descartes)

Saïd Ladjal
(Télécom ParisTech)

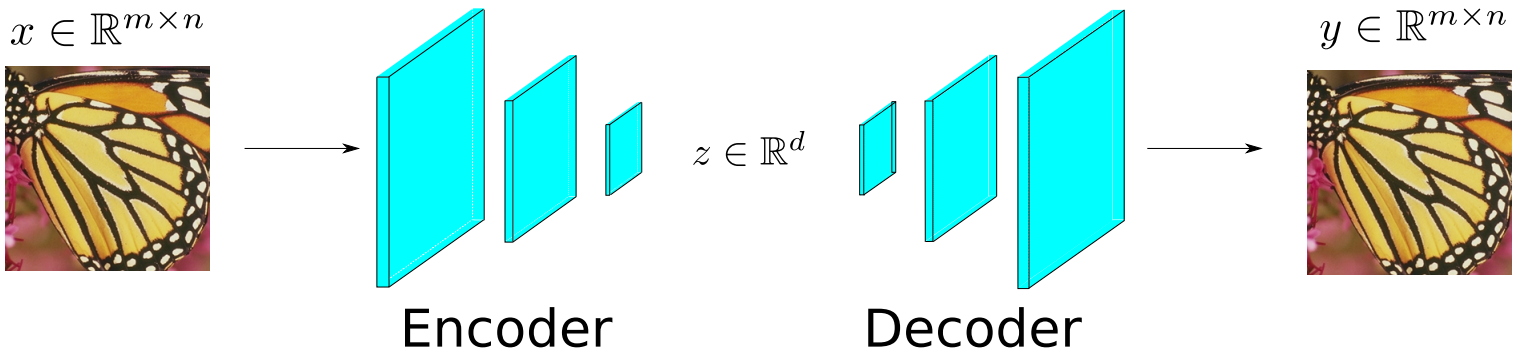
Yann Gousseau
(Télécom ParisTech)

Chi-Hieu Pham
(Télécom ParisTech)

Autoencoders - introduction

What are autoencoders ?

- Deep neural networks
  - Cascaded operations : linear transformations, convolutions, non-linearities
  - Great flexibility : approximate a large class of functions
- Autoencoder : neural network designed for **compressing** and **uncompressing** data



- The lower-dimensional space in the middle is known as the **latent space**

Autoencoders - introduction

What are autoencoders used for ?

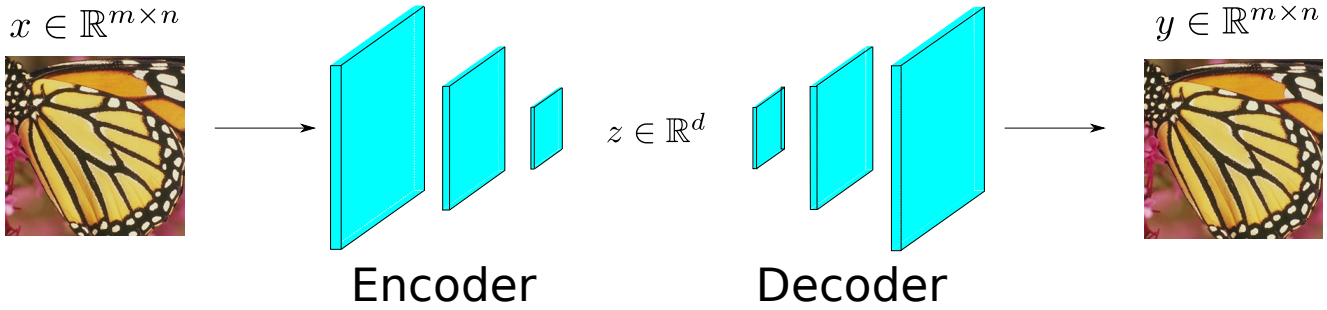
- Synthesis of high-level/abstract images
- Autoencoder-type networks which are designed for synthesis are known as **Generative Models**
  - Eg.: Variational Autoencoders and Generative Adversarial Networks (GANs)

Density estimation using Real NVP, L. Dinh, J. Sohl-Dickstein, S. Bengio, arXiv 2016

- These produce impressive results. However, autoencoder mechanisms and latent spaces are not well understood
- Goal of our work : **understand underlying mechanisms**, and create **interpretable and navigable latent spaces**

Subject of this talk

Understanding and Organising the Latent Space of Autoencoders



- Subjects of this talk
  1. Understand how autoencoders can encode/decode basic geometric attributes of images
     - Size
     - Position
  2. Propose an autoencoder algorithm which aims to separate different image attributes in the latent space
     - PCA-like autoencoder
     - Encourage ordered and decorrelated latent spaces

Summary

Autoencoding size

- We are interested in understanding how autoencoders can encode/decode shapes

Example of latent space interpolation in a generative model

- Simple example of such a shape is a disk
- How can an autoencoder encode and decode a disk ?
- We present our problem setup now

† ***Generative Visual Manipulation on the Natural Image Manifold***, *J-Y. Zhu, P. Krähenbühl, E. Schechtman, A. Efros, CVPR 2016*

Disk autoencoder : problem setup

Autoencoding size

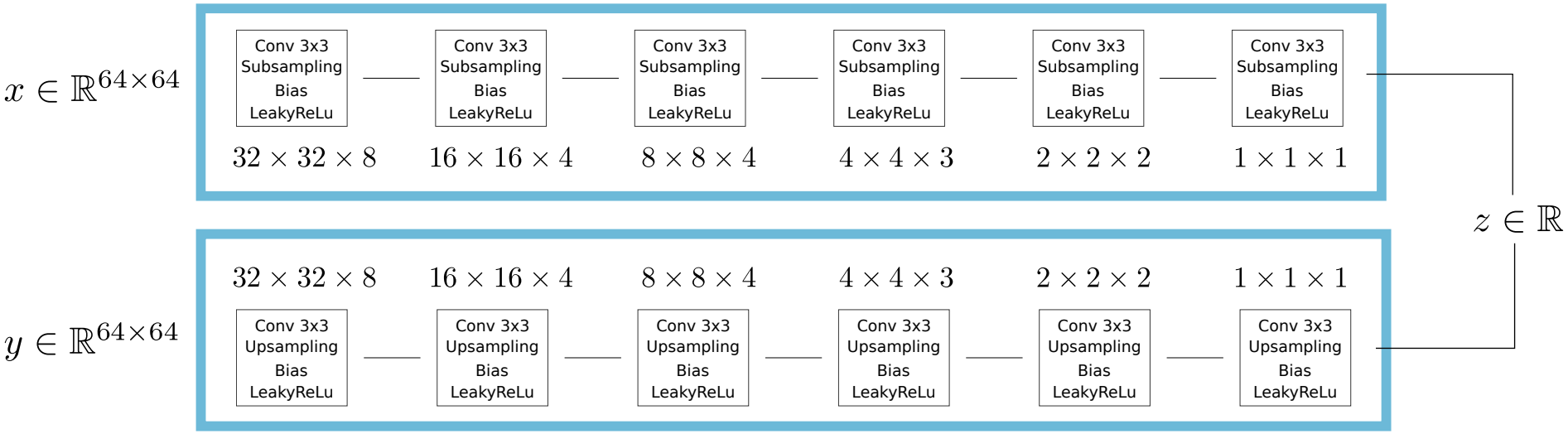
- Can AEs encode and decode a disk "optimally"; if so, how ?
- Training set : square, disk, images of size 64×64
  - Blurred slightly to avoid discrete parameterisation
- Each image contains one **centred disk of random radius r**

- Optimality, perfect reconstruction : $x = D \circ E\,(x)$, with smallest d possible $(d = 1)$
- E is the encoder, D is the decoder

Disk autoencoder : problem setup

Disk autoencoder design

$$E = E_L \circ \cdots \circ E_1$$



$$D = D_1 \circ \cdots \circ D_L$$

- Four operations : convolution, sub/up-sampling, additive biases, Leaky ReLU :

$$\phi_\alpha(t) = \begin{cases} t \text{ , if } t > 0 \\ \alpha t \text{ , if } t \leq 0 \end{cases}$$

- Number of layers determined by subsampling factor $s = \frac{1}{2}$

Disk autoencoder

Disk autoencoding training minimisation problem

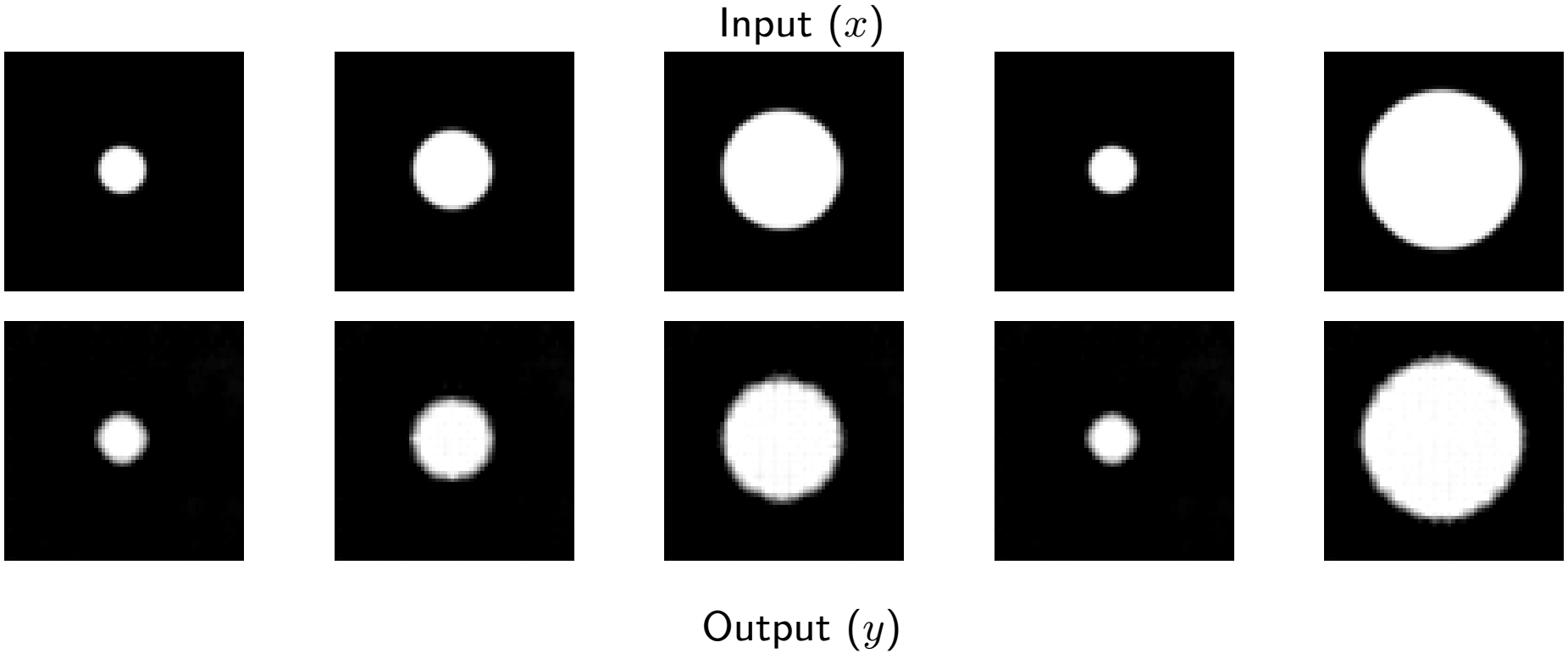
$$\hat{\Theta}_E, \hat{\Theta}_D = \underset{\Theta_E, \Theta_D}{\arg\min} \sum_{x_r} \| D \circ E\left(x_r\right) - x_r \|_2^2 \tag{1}$$

- Θ_E, Θ_D : parameters of the network (weights and biases)
- x_r : image containing disk of radius r

- NB : We do not enter into the minimisation details here (Adam optimiser)

Investigating autoencoders

First question, can we compress disks to 1 dimension ? Yes !

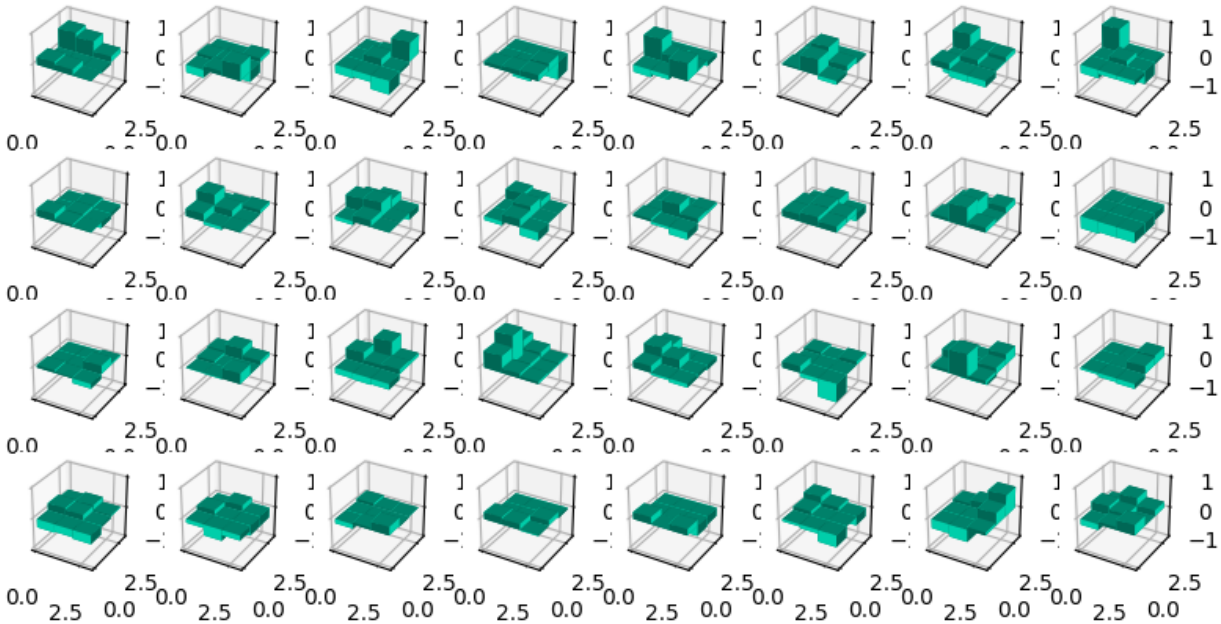


- Let us try to understand how this works

Investigating autoencoders

How does the autoencoder work in the case of disks ?

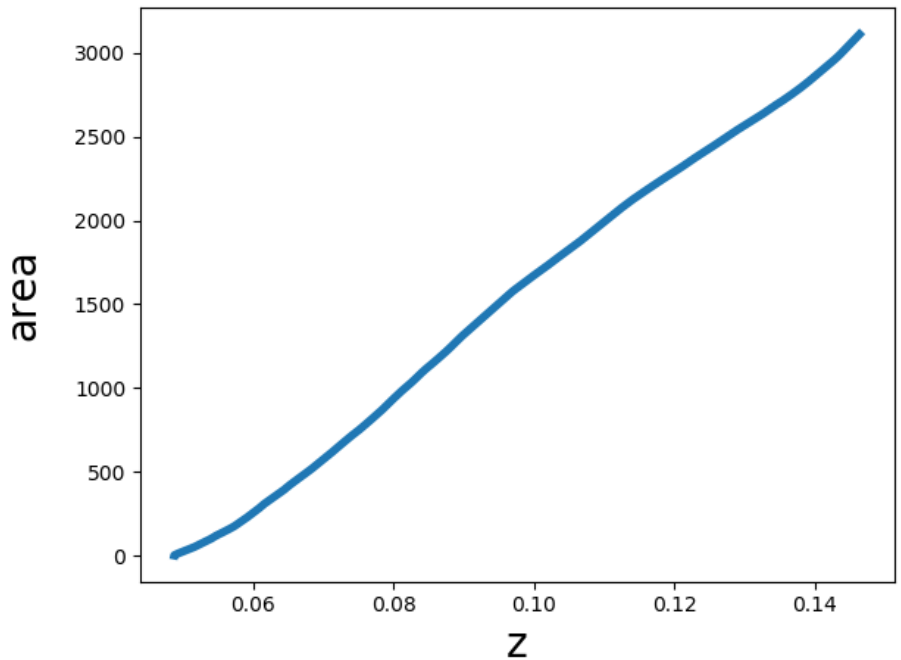
- First idea, inspect network *weights*
- Unfortunately, very difficult to interpret



Example of weights (3×3 convolutions)

Investigating autoencoders

How does the encoder work : inspect the *latent space*



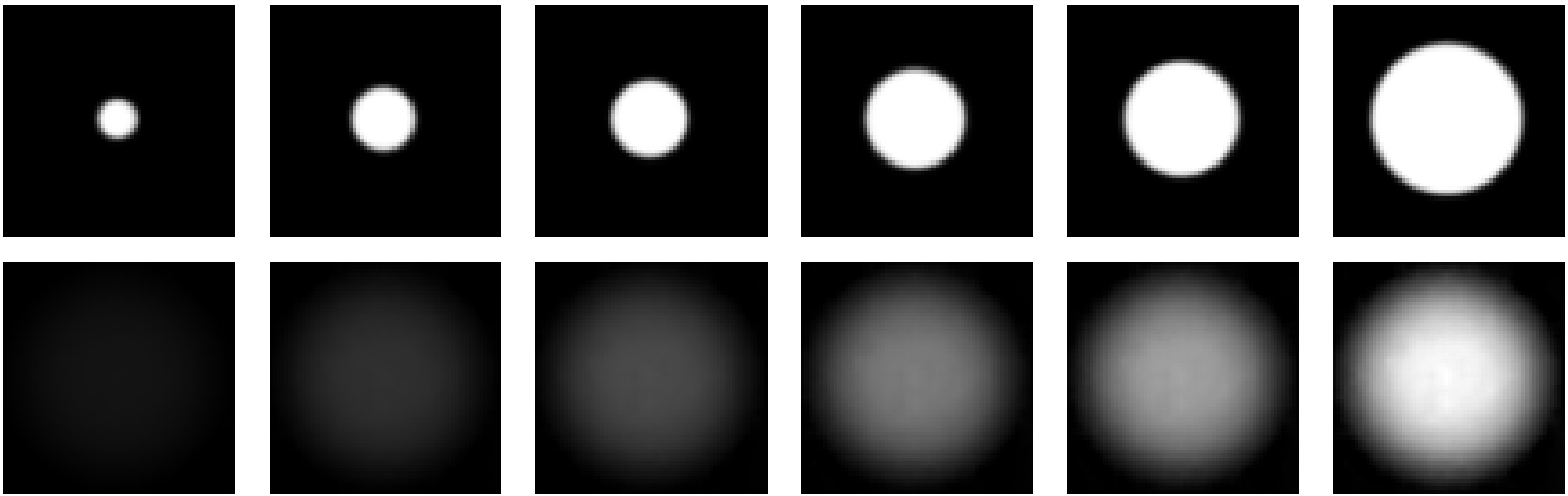
- Encoding simple to understand : averaging filter gives area of disks*
- How about decoding ?
  - Inspecting weights and biases is tricky
  - We can describe the decoding function when we **remove the biases** (ablation study)

* *In fact, one can show that the optimal encoding is indeed the area, when a contractive loss is used*

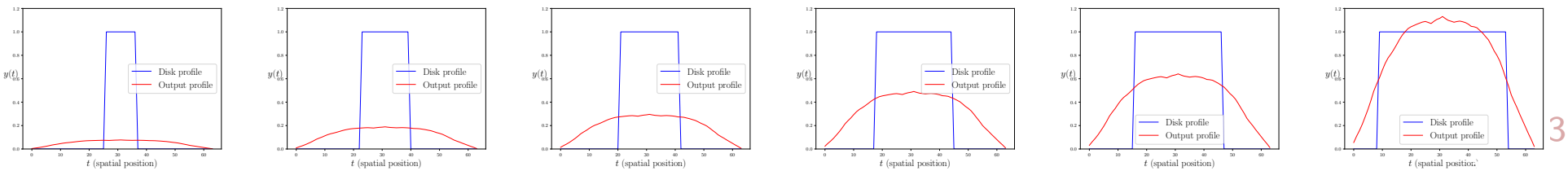
Decoding a disk

Ablation study : remove biases of the network

Input



Output



Positive Multiplicative Action of the Decoder Without Bias

Consider a decoder, without biases, with $D^{\ell+1} = \text{LeakyReLU}_\alpha\left(U(D^\ell) * w^\ell\right)$, where U is an upsampling operator. In this case, we have

$$\forall z,\, \forall \lambda \in \mathbb{R}^+,\, D(\lambda z) = \lambda D(z). \tag{2}$$

Positive Multiplicative Action of the Decoder Without Bias

Consider a decoder, without biases, with $D^{\ell+1} = \mathsf{LeakyReLU}_\alpha\left(U(D^\ell) * w^\ell\right)$, where U is an upsampling operator. In this case, we have

$$\forall z,\, \forall \lambda \in \mathbb{R}^+,\; D(\lambda z) = \lambda D(z). \tag{2}$$

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

- Output can be written $y = h(r)f$, with f learned during training
- In the case without bias, we can rewrite the training problem in a simpler form

Decoding a disk

Disk autoencoding training problem (continuous case), without biases

$$\hat{f} = \arg\max_{f} \int_0^R \langle f, \mathbb{1}_{B_r} \rangle^2 \, dr \tag{3}$$

Proof : The **continuous** training minimisation problem can be written as

$$\hat{f}, \hat{h} = \arg\min_{f,h} \int_0^R \int_\Omega \left(h(r) f(t) - \mathbb{1}_{B_r}(t) \right)^2 \, dt \, dr \tag{4}$$

Also, for a fixed f, the optimal h is given by

$$\hat{h}(r) = \frac{\langle f, \mathbb{1}_{B_r} \rangle}{\|f\|_2^2} \tag{5}$$

Decoding a disk

- We insert the optimal $\hat{h}(r)$, and choose the (arbitrary) normalisation $\|f\|_2^2 = 1$
- This gives us the final result :

$$\hat{f} = \arg\min_f \int_0^R -\langle f, \mathbb{1}_{B_r} \rangle^2 \, dr \tag{6}$$

$$= \arg\max_f \int_0^R \langle f, \mathbb{1}_{B_r} \rangle^2 \, dr. \tag{7}$$

Decoding a disk

- We insert the optimal $\hat{h}(r)$, and choose the (arbitrary) normalisation $\|f\|_2^2 = 1$
- This gives us the final result :

$$\hat{f} = \arg\min_f \int_0^R - \langle f, \mathbb{1}_{B_r} \rangle^2 \, dr \tag{6}$$

$$= \arg\max_f \int_0^R \langle f, \mathbb{1}_{B_r} \rangle^2 \, dr. \tag{7}$$

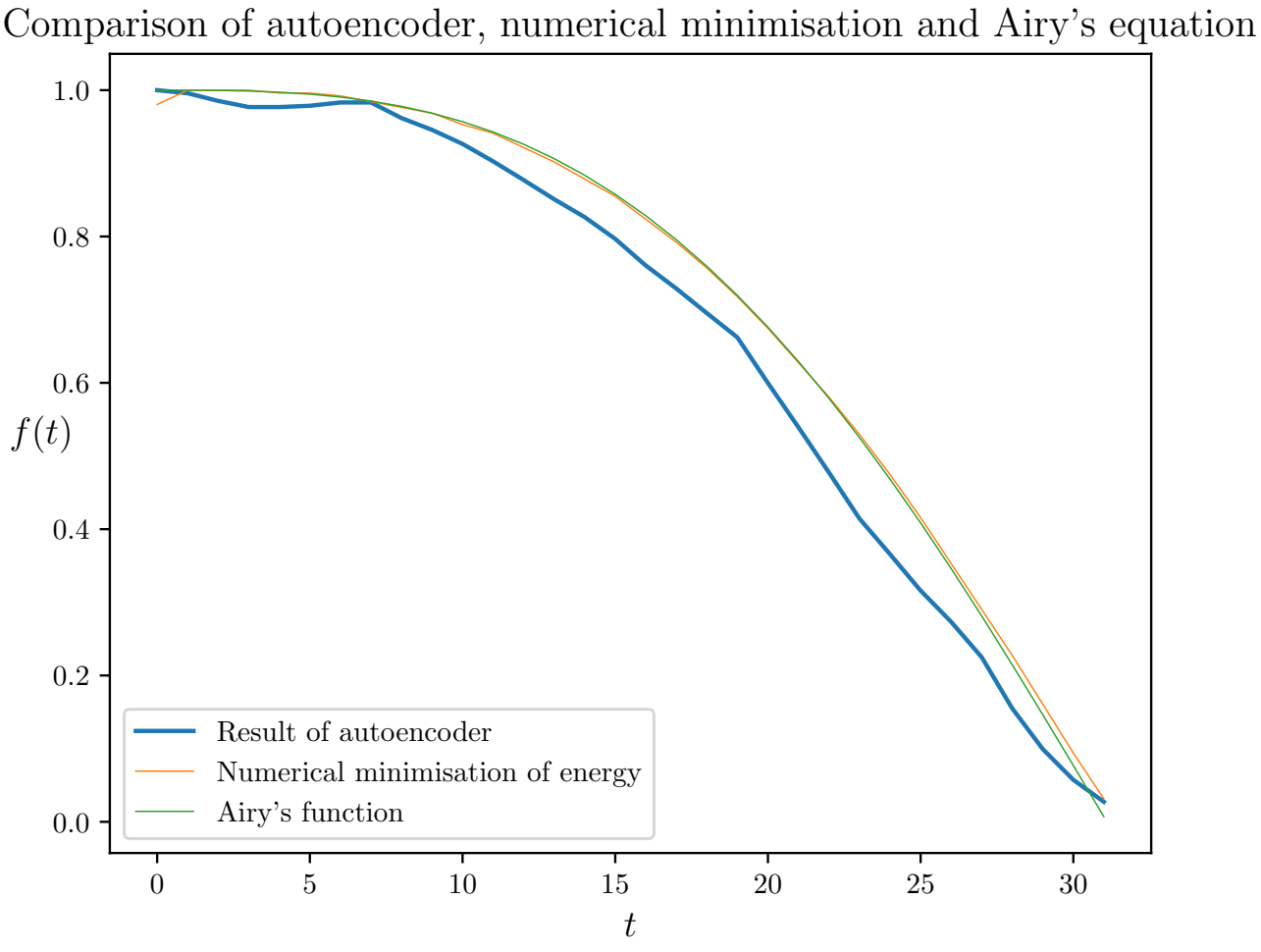
- Since the disks are **radially symmetric**, the integration can be simplified to one dimension
- The first variation of the functional in Equation (3) leads to a differential equation, Airy's equation

$$f''(\rho) = -kf(\rho)\rho, \tag{8}$$

with $f(0) = 1, f'(0) = 0$

Decoding a disk

- The functional is indeed minimised by the training procedure



Decoding a disk

Summary

- Encoder : integration (averaging filter) sufficient
- Decoder : a function learned, scaled and thresholded

- The encoder extracts the parameter of the shape (radius here)
- The decoder contains a primitive of the shape
  - Parametrisation of this shape uses latent space

Summary

- Further work : apply this to scaling of any shape
  - Useful for understanding how autoencoders process binary images

Scaled mnist data

Corpus callosum data (MRI images)

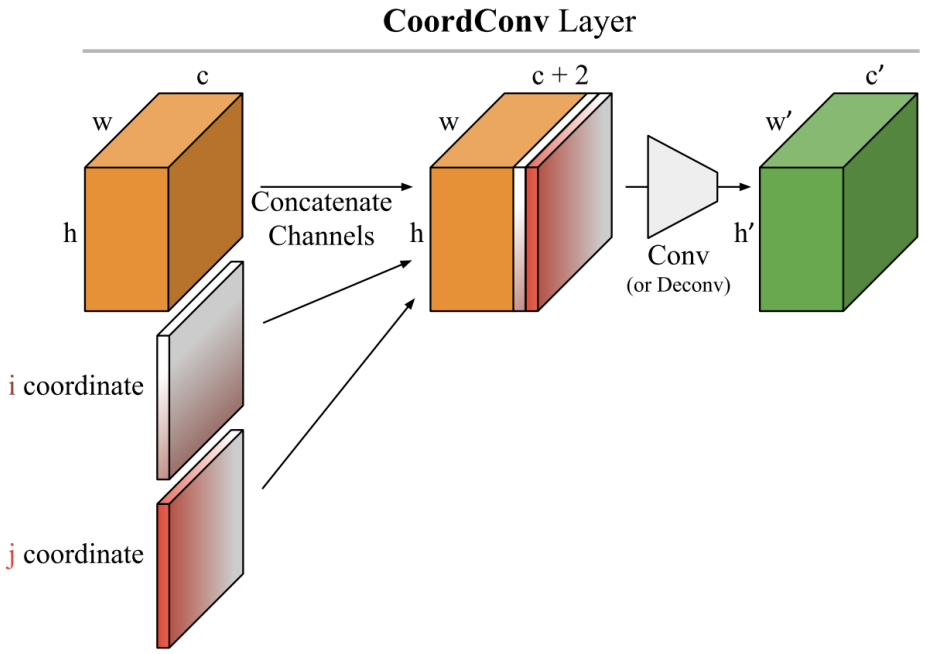
Summary

Autoencoding position

- The second characteristic we wish to extract is **position**
- In many cases, the objects in images are somewhat centred, however, not completely
- Autoencoders still need to be able to describe position

Autoencoding position

- Few workq concentrate on the positional aspect of autoencoders
- "CoordConv"*
  - Solution to position problem : explicitly add spatial information



- However, we wish to understand how an autoencoder can do this without explicit "instructions" (in an unsupervised manner)

* *R. Liu et al, An Intriguing Failing of Convolutional Neural Networks and the CoordConv Solution, NIPS, 2018.*

Autoencoding position

- We first studied the capacity of an autoencoder to **encode** position
- Consider the 1D case of a one-hot vector δ_a (a Dirac impulse), with a 1 at position a, with $a = 0, \ldots, n-1$

$$\delta_a(i) = \begin{cases} 1 & \text{if} \quad i = a \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

Autoencoding position

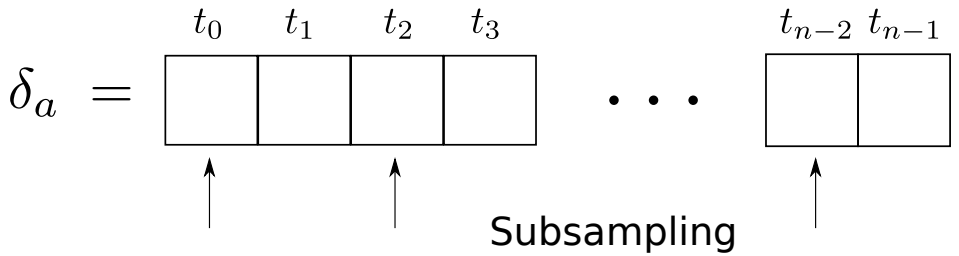
- We first studied the capacity of an autoencoder to **encode** position
- Consider the 1D case of a one-hot vector δ_a (a Dirac impulse), with a 1 at position a, with $a = 0, \ldots, n-1$

$$\delta_a(i) = \begin{cases} 1 & \text{if} \quad i = a \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

- It turns out that extracting the position a from δ_a can be achieved with a **simple filter and subsampling**, with filter φ :

$$\varphi = [1, 2, 1] \tag{10}$$

- We subsample **at even positions** :



Autoencoding position

- We denote with u^ℓ the output of layer ℓ. A "layer" is one filtering and one subsampling

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 1: Results of all possible one-hot vectors of size eight in the simple linear neural network with filter φ

Autoencoding position

Encoding position in an autoencoder

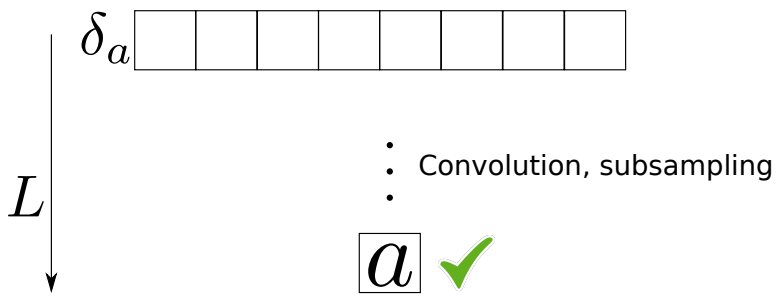
- Let E^L refer to the linear network created by a cascade of filtering and subsampling with filter φ and subsampling $\frac{1}{2}$
- The network E^L indeed **extracts the (inverted) position** of a from δ_a, $E^L(\delta_a) = 2^L - a$

Autoencoding position

Encoding position in an autoencoder

- Let E^L refer to the linear network created by a cascade of filtering and subsampling with filter φ and subsampling $\frac{1}{2}$
- The network E^L indeed **extracts the (inverted) position** of a from δ_a, $E^L(\delta_a) = 2^L - a$

- Proof : **induction argument over the number of layers**
  - Hypothesis : E^L contains L hidden layers, and extracts the position of δ_a

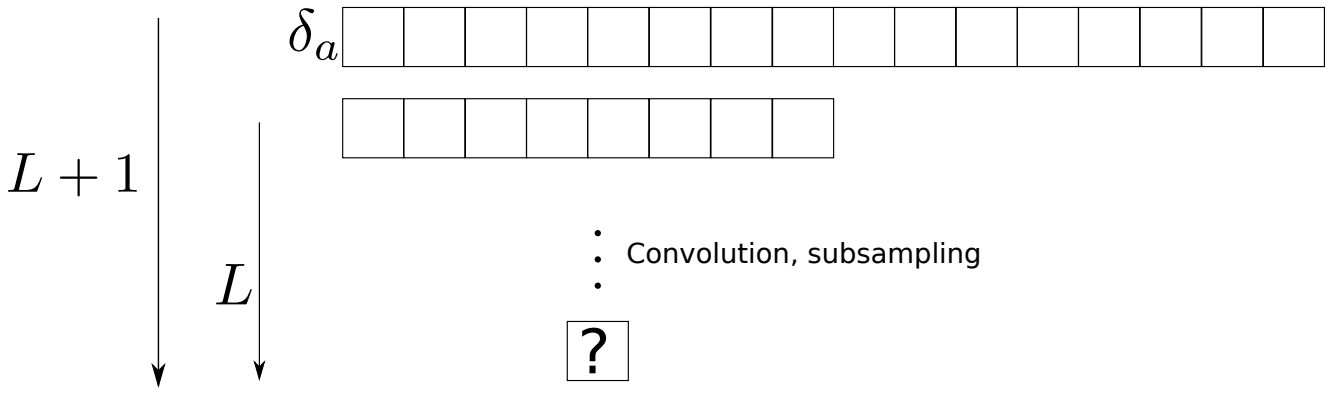


Autoencoding position

Encoding position in an autoencoder

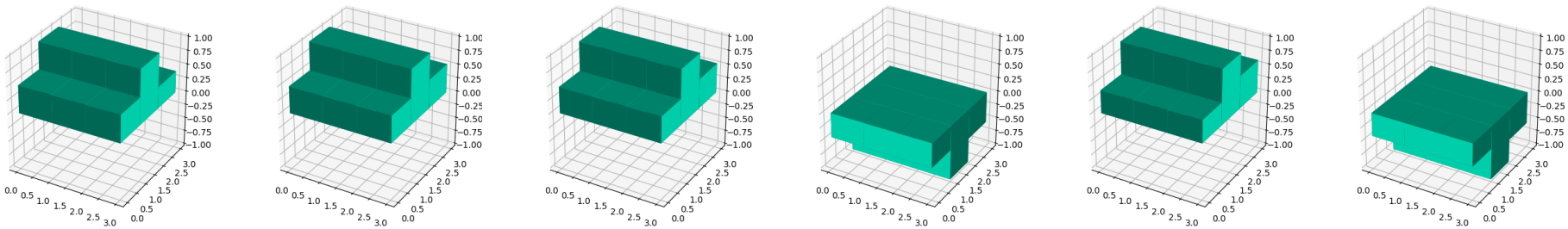
- Let E^L refer to the linear network created by a cascade of filtering and subsampling with filter φ and subsampling $\frac{1}{2}$
- The network E^L indeed **extracts the (inverted) position** of a from δ_a, $E^L(\delta_a) = 2^L - a$

- Proof : **induction argument over the number of layers**
  - Hypothesis : E^L contains L hidden layers, and extracts the position of δ_a
  - Induction : by adding a layer to E, position is still correctly extracted



Autoencoding position

- The predicted weights are indeed found during training of an encoder of position*
- The (normalised) weights are correctly predicted
  - Note, 3D representation is for easier viewing, the weights are in 1D



Experimental position encoder weights

* *The encoder was explicitly given the position as the label*

Autoencoding position

- Decoding position is more difficult to analyse, ongoing work
- Given the position a as an input, possible to train a decoder to produce δ_a, however this does not produce reliable results
  - Due to the very limited number of Dirac impulses
- Can be partly addressed by using another approximation of a Dirac, where a is now a continuous parameter

$$\delta_a(t) = \begin{cases} 1 - (x - \lfloor x \rfloor) & \text{if} \quad t = \lfloor x \rfloor \\ 1 - (\lceil x \rceil - x) & \text{if} \quad t = \lceil x \rceil \\ 0 \quad \text{otherwise} \end{cases} \tag{11}$$

Summary

- Main takeaway : encoding position is not difficult (in this simplified setting)
  - Does not even require non-linearity
- However, crucial point : **strided convolution** (conv + subsampling) is necessary
  - Max pooling makes this behaviour break down

Summary

PCA Autoencoder

- Autoencoders extract the essential information of data, and represent this in the latent space
- Latent space is often poorly understood, difficult to manipulate
  - It is not clear what each of the axes in the latent space mean
  - Components can be correlated, also known as **entanglement** (attributes are mixed up)
- A key issue for generative networks, since many works propose to **navigate in the latent space**
- **Disentanglement** appears as a natural requirement of generative models

Entangled latent space

PCA Autoencoder

- Most works use a supervised approach : they have access to the labels which need to be disentangled
  - "Fader Networks"* isolate certain attributes in the latent space

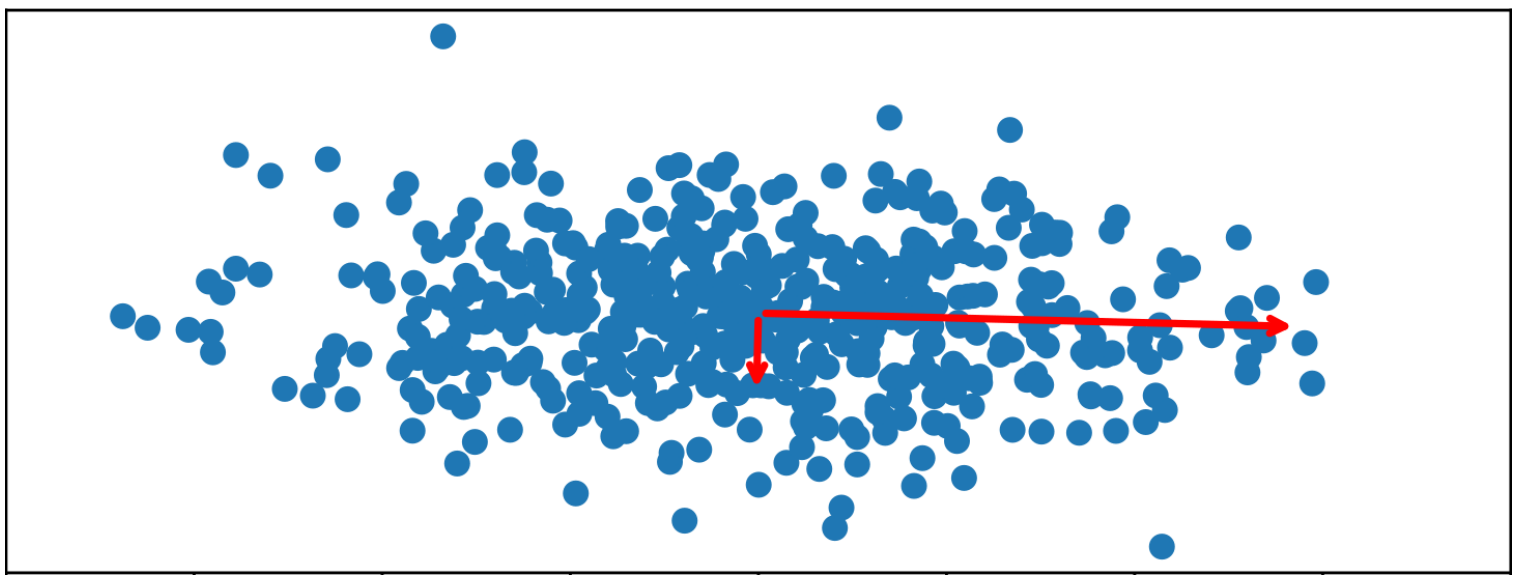
- We want an unsupervised autoencoder to **discover** independent characteristics
  - Cannot annotate geometry/colour, yet we wish to control them separately

Result of Zhu et al*, interpolation in the latent space

* *G. Lample et al, Fader networks: Manipulating images by sliding attributes, NIPS, 2017*
* *Zhu et al, Generative Visual Manipulation on the Natural Image Manifold , ECCV, 2016*

PCA Autoencoder

- We present an unsupervised algorithm to achieve these goals
- First remark : autoencoder greatly ressembles **Principal Component Analysis** (PCA)
- Major differences between the autoencoder and the PCA
  - Autoencoder is a **non-linear transformation**, the PCA is linear
  - PCA's axes are ordered in decreasing "importance", increases interpretability of the latent space
  - PCA finds statistically decorrelated components



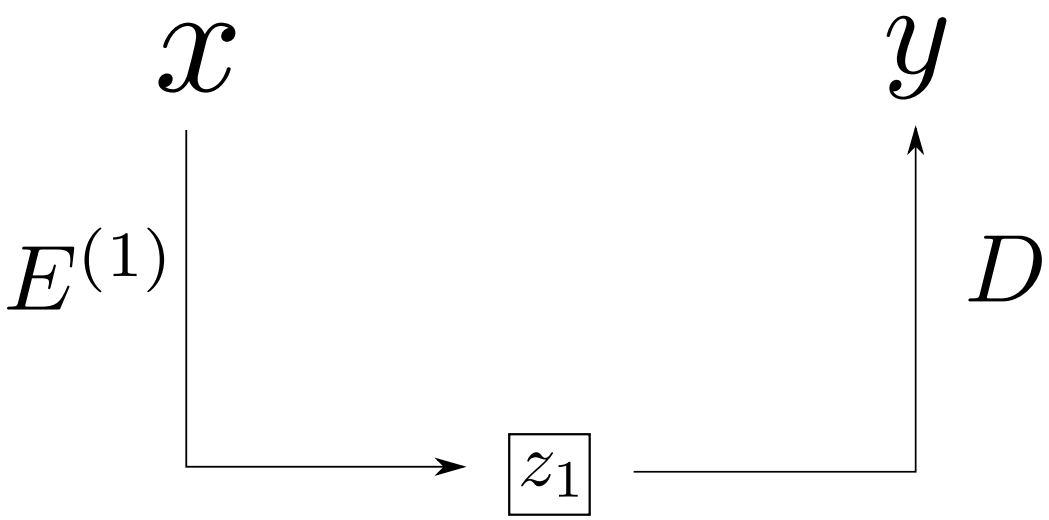
PCA Autoencoder

- To have the best of both worlds, we need to impose two criteria on the non-linear latent space
  1. Increasing importance of components
  2. Decorrelation of components
- We propose a PCA-like autoencoder which aims to achieve this, with two key choices
  1. **Progressively increasing** the latent space size to capture most important variabilities in data
  2. A **covariance loss term** to minimise correlation of latent components

PCA Autoencoder

PCA autoencoder architecture

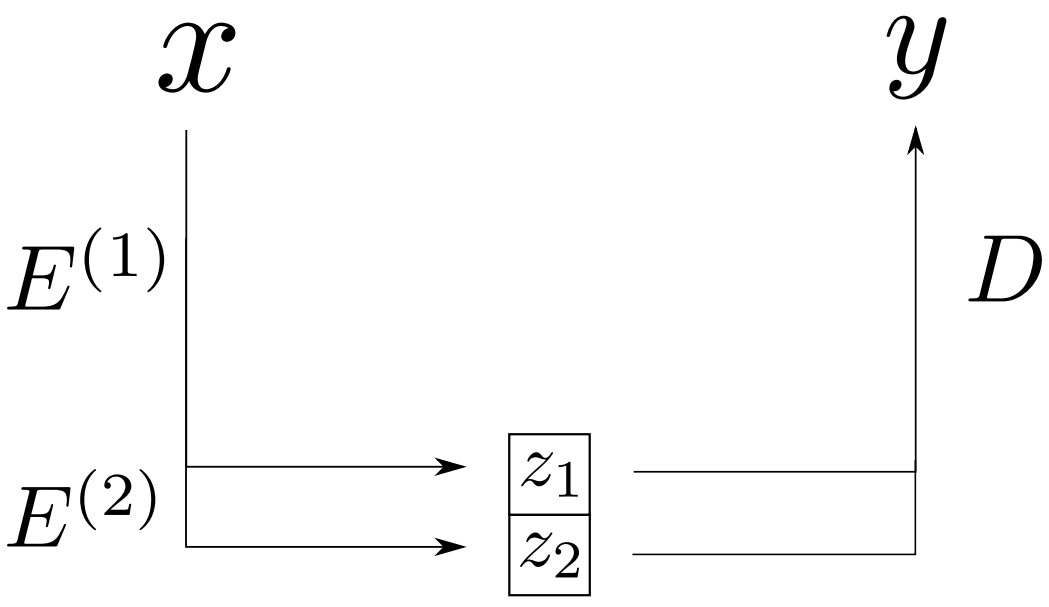
- Each encoder $E^{(i)}$ is trained, and then fixed
- At each iteration, the decoder is thrown away, and a new one is trained



PCA Autoencoder

PCA autoencoder architecture

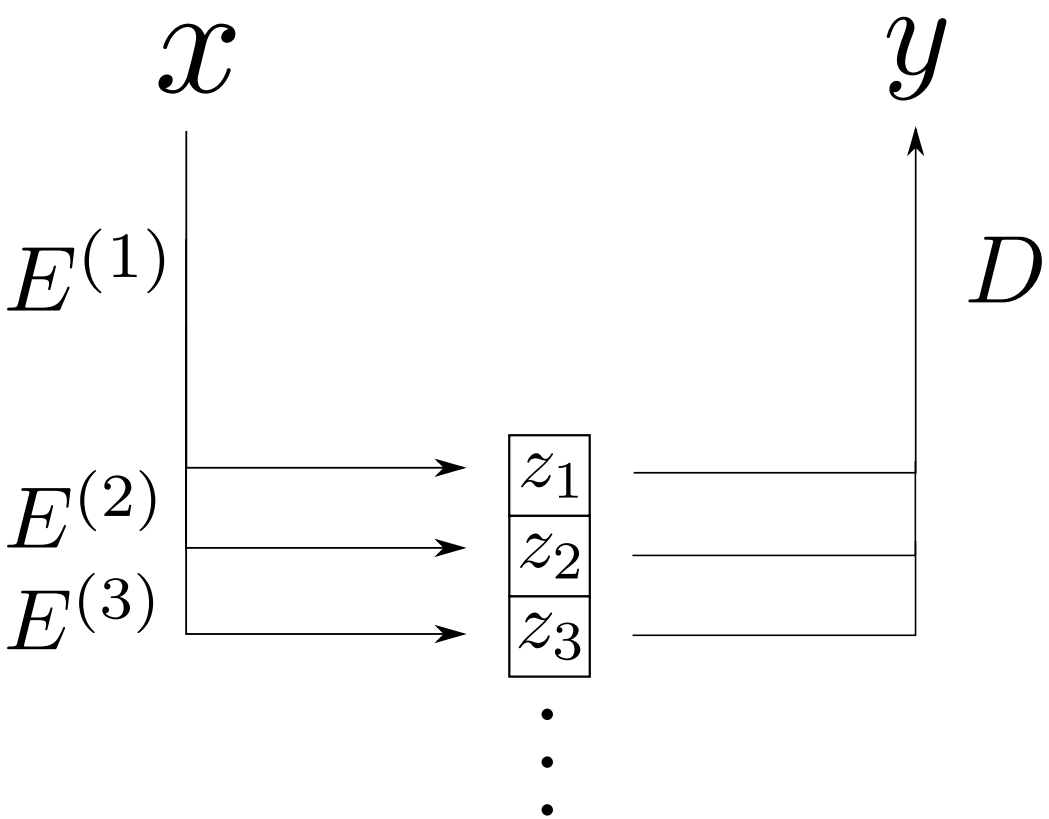
- Each encoder $E^{(i)}$ is trained, and then fixed
- At each iteration, the decoder is thrown away, and a new one is trained



PCA Autoencoder

PCA autoencoder architecture

- Each encoder $E^{(i)}$ is trained, and then fixed
- At each iteration, the decoder is thrown away, and a new one is trained



PCA Autoencoder

- We want the components of the latent space to be uncorrelated
  - Goal : improve interpretability
  - Decorrelated components likely **represent different image attributes**

PCA Autoencoder

- We want the components of the latent space to be uncorrelated
  - Goal : improve interpretability
  - Decorrelated components likely **represent different image attributes**
- We minimise the **covariance** between latent variables :
  - $\text{Cov}(z_1, z_2) = \mathbb{E}[z_1 z_2] - \mathbb{E}[z_1]\mathbb{E}[z_2]$
- If we note m the number of elements in a batch, then we can use the following estimate

$$\left(\frac{1}{m} \sum_j (z_1^{(j)} \; z_2^{(j)}) - \frac{1}{m^2} \sum_j z_1^{(j)} \sum_j z_2^{(i)} \right)^2 \tag{12}$$

- If the latent codes are **zero-mean**, this can be simplified to

$$\left(\frac{1}{m} \sum_j^m z_1^{(j)} \; z_2^{(j)} \right)^2 \tag{13}$$

PCA Autoencoder

- We impose $\mathbb{E}[z] = 0$ by adding a **Batch Normalisation layer*** just before the latent space
- Therefore, for iteration k, our covariance loss is :

$$\mathcal{L}_{\text{cov}}^{(k)}(z) = \frac{1}{k} \sum_{i=1}^{k-1} \left(\frac{1}{m} \sum_{j=1}^{m} (z_i^{(i)} \; z_k^{(i)}) \right)^2 \tag{14}$$

- The total PCA autoencoder loss is

$$\mathcal{L}^{(k)}(x) = \|x - D \circ E^{(k)}(x)\|_2^2 + \lambda \mathcal{L}_{\text{cov}}^{(k)} \left(E^{(k)}(x) \right) \tag{15}$$

- The exact architecture depends on the type of data (more or less layers etc.)

* *Without training the Batch Normalisation parameters*

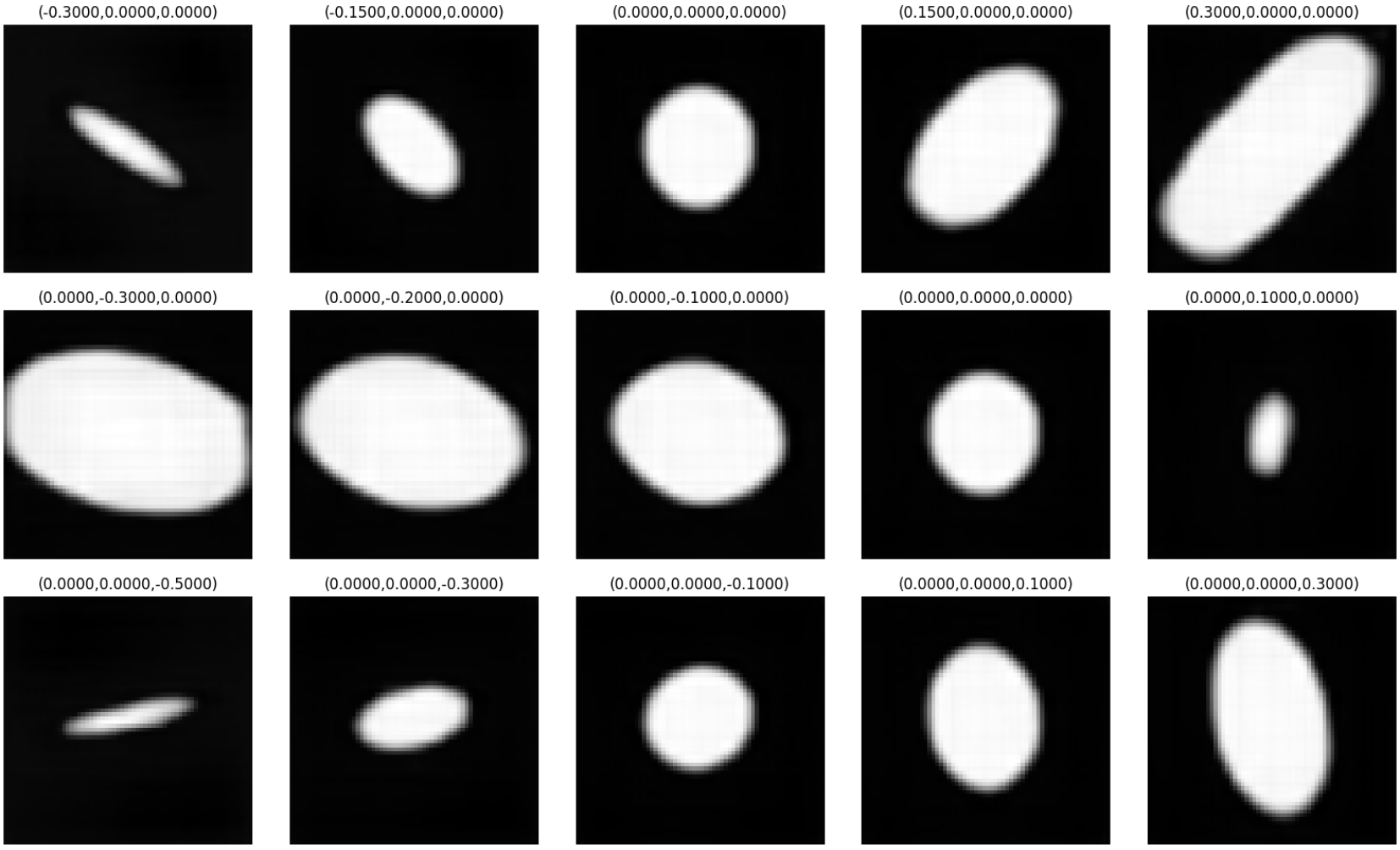
PCA Autoencoder - results

- We show some results on synthetic data, ellipses with three parameters (two axes and rotation)

Ellipse dataset example images

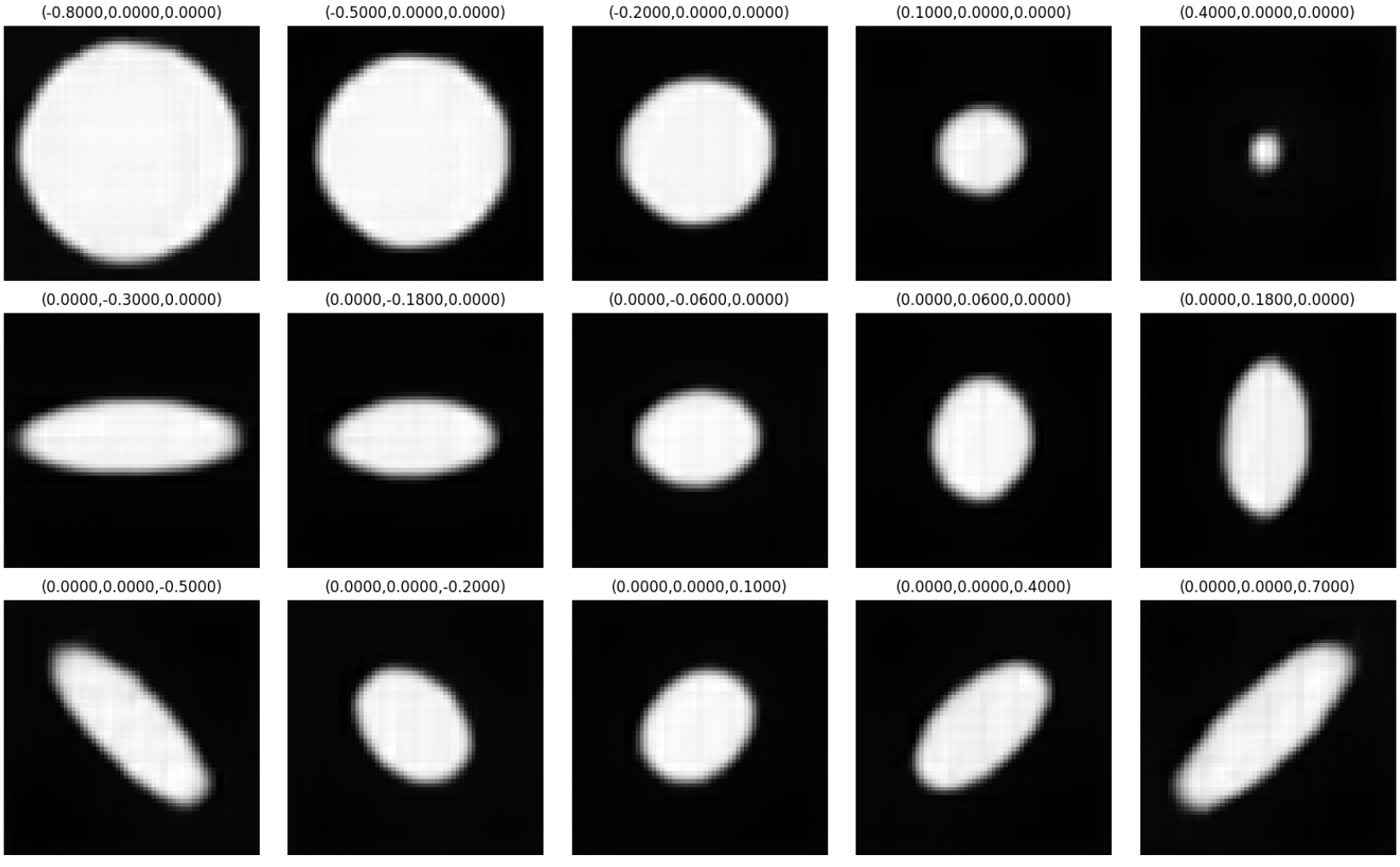
PCA Autoencoder - results

Latent space navigation - standard autoencoder



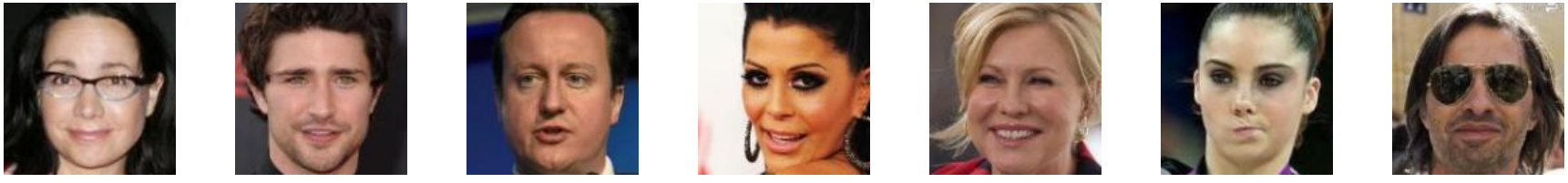
PCA Autoencoder - results

Latent space navigation - PCA autoencoder



PCA Autoencoder

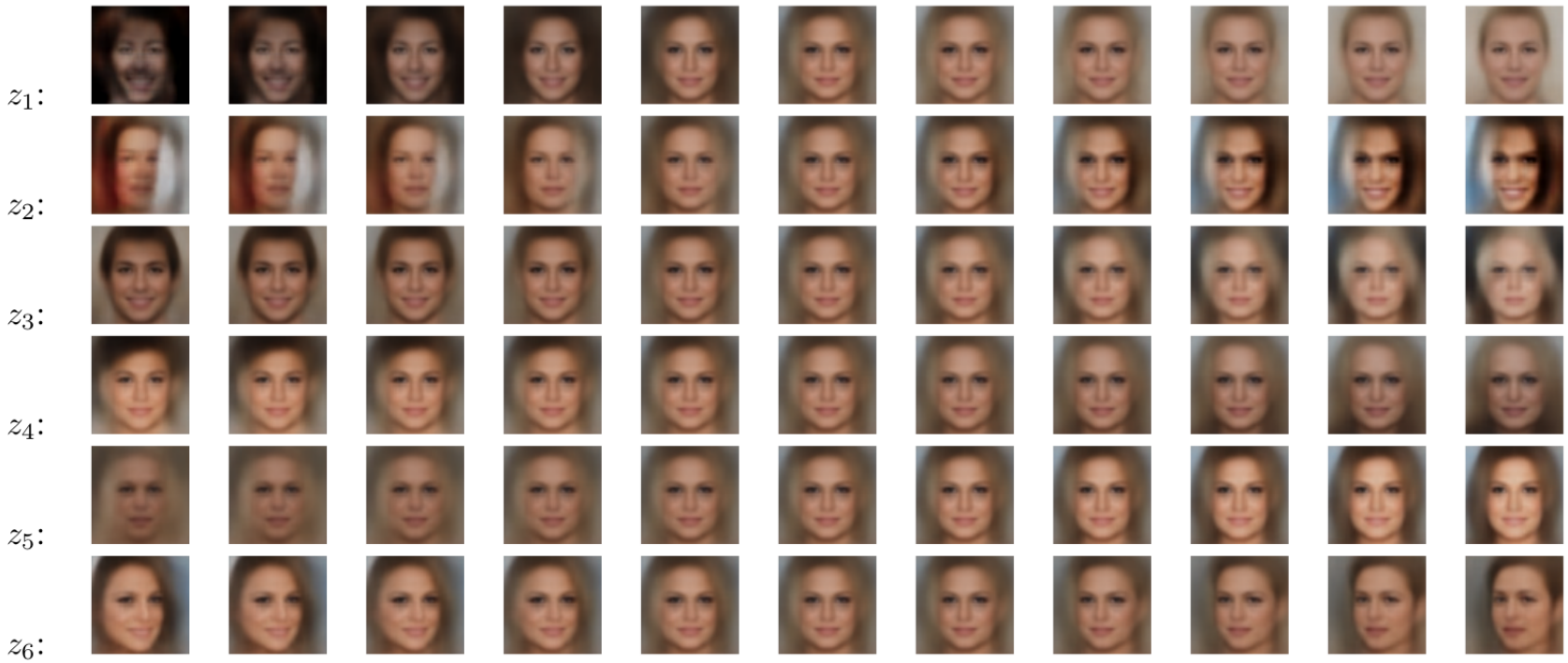
- The PCA autoencoder allows for meaningful navigation in the latent space of simple geometric shapes
- What happens if we try to apply the PCA autoencoder to more complex data ?
- Face data, from the "Celeba" dataset*
  - More than 200,000 images
  - 10,177 identities, 40 attributes (glasses, mustache etc)



* *Liu et al, Large-scale CelebFaces Attributes (CelebA) Dataset, ICCV 2015*

PCA Autoencoder

- If we apply our PCA autoencoder directly to the images of the celeb-a database, we get the following results :



- Extremely blurry : we tend to extract an average image
- Difficult to create and organise the latent space at the same time
- Solution : apply the PCA autoencoder **directly to the latent space of a pre-trained GAN**

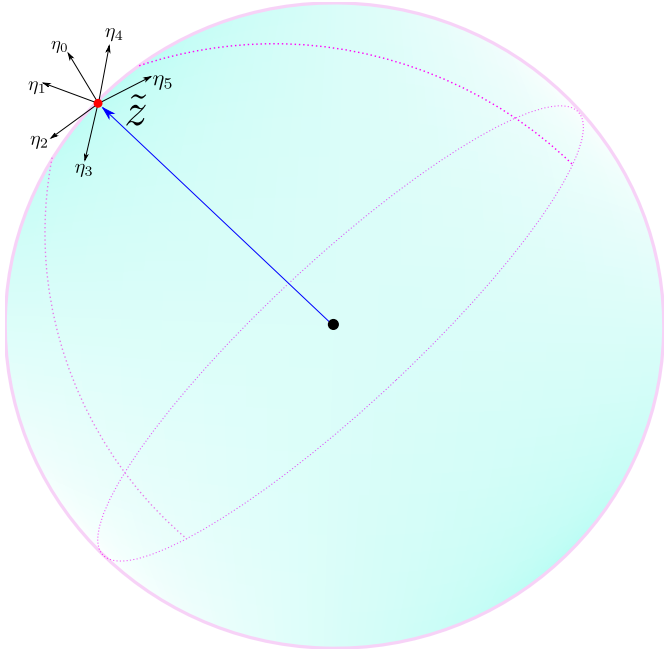
PCA Autoencoder

- A GAN is basically a decoder (called a "generator") with a probability distribution imposed on the latent space
  - Takes a random vector and decodes it as an images
- We used the powerful PGAN* (high-resolution images)

† ***Progressive growing of gans for improved quality, stability, and variation***, *Karras, T., Aila, T., Laine, S., and Lehtine, J., arXiv preprint arXiv:1710.10196, 2017*

PCA Autoencoder

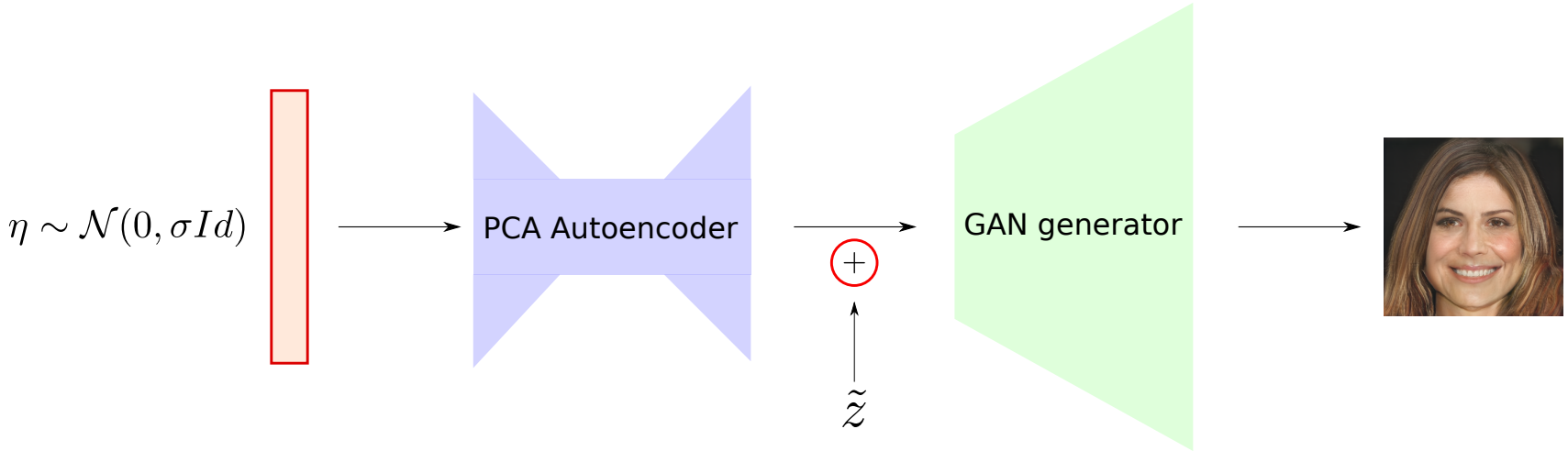
- We found the complete PGAN latent space too complicated to use directly
- Thus, we learn a **local space** around a certain, chosen, code $\tilde{z}$
  - The database consists of $\tilde{z} + \eta$, with $\eta \sim \mathcal{N}(0, \sigma Id)$



- $\tilde{z}$ is on the unit sphere, since PGAN normalises its input

PCA Autoencoder

- The PCA autoencoder applied to a pre-trained GAN latent space is trained in the following manner



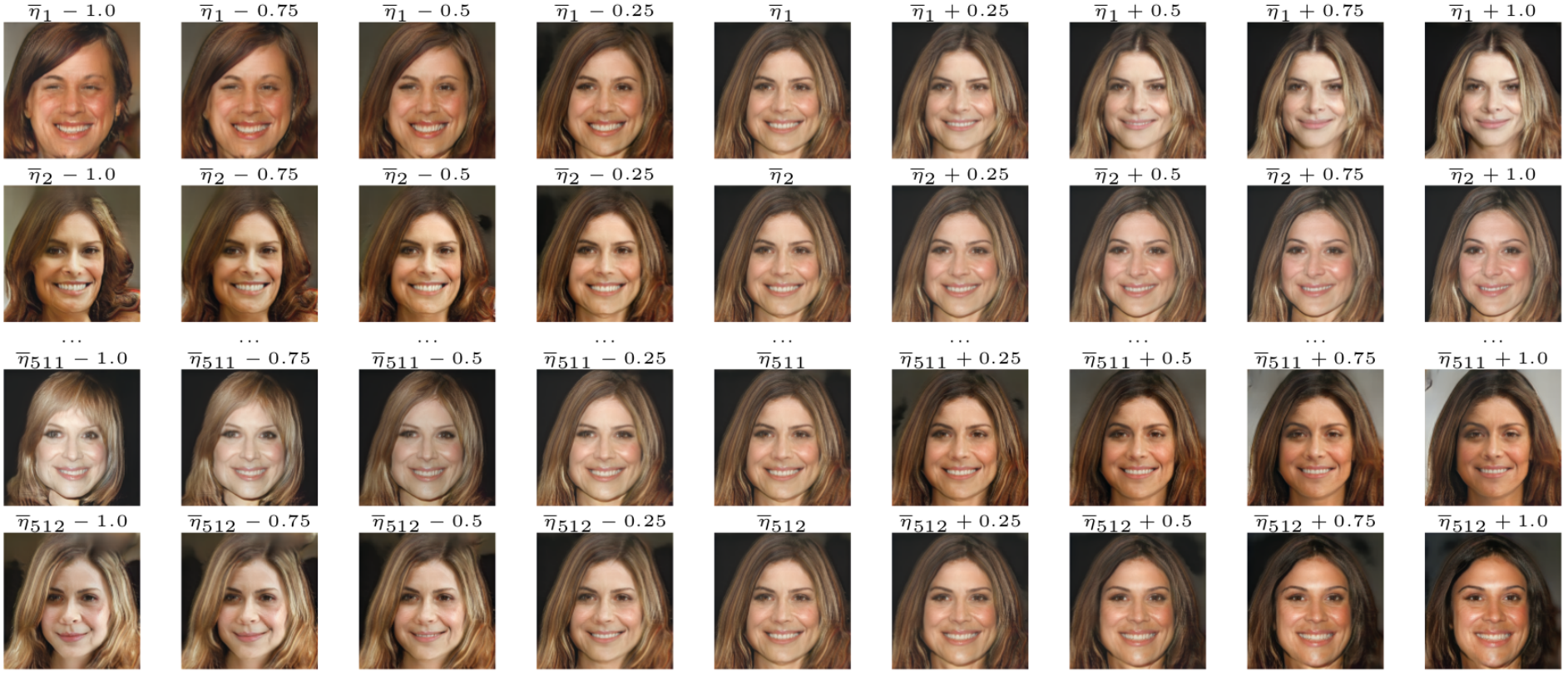
- We require that the final synthesis result be meaningful, therefore we change the loss of the PCA autoencoder to

$$\mathcal{L}(\eta) = \|G(\eta + \tilde{z}) - G(D \circ \ E(\eta) + \tilde{z})\|_2^2 \ + \ \mathcal{L}_{\text{cov}}(\eta) \tag{16}$$

- G is the GAN's generator

PCA Autoencoder

- Firstly, we look at navigation in the **original PGAN space**, locally around a certain z_0
- Facial attributes are mixed up (entangled) : hair colour, identity, smile

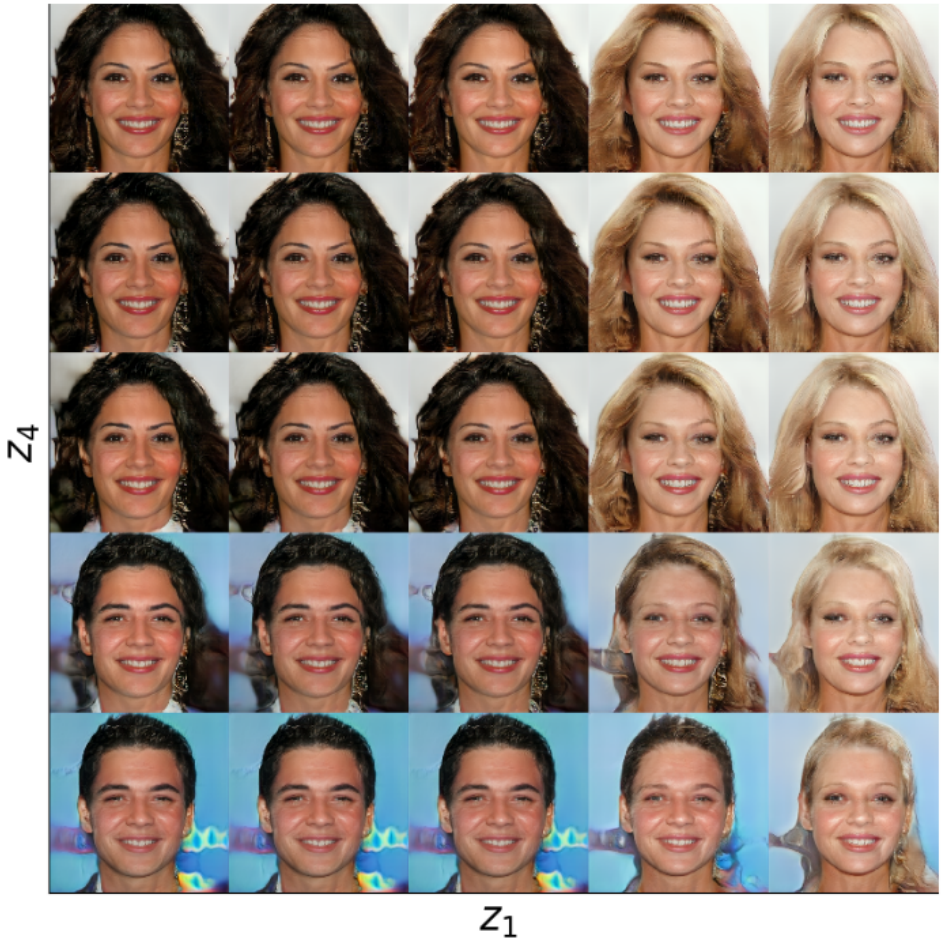


PCA Autoencoder

- Our method's results. We see that the first axis corresponds to hair colour, the second to pose, the third to identity
- Note, our method is **entirely unsupervised** : at no point does the algorithm have access to any labels

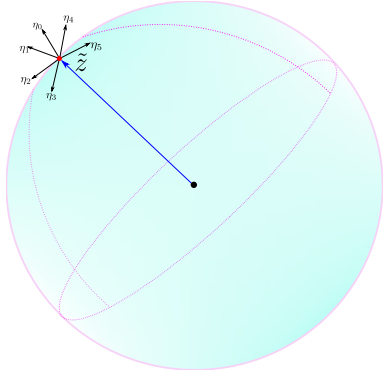
PCA Autoencoder

- Some more results : z_1 vs z_4



PCA Autoencoder

- The PCA autoencoder allows for meaningful navigation and interpretation of the latent space
- We can discover existing independent attributes in the data, **in a completely unsupervised manner**
- However, there are certain drawbacks of our approach :
  - Cases where progressively increasing the latent space might not work
    - Translation for example, autoencoder needs two coordinates together
  - Possible solution : increase the latent space size in code packets
- It is likely that the PCA autoencoder works best as a tool for **local exploration/organisation** of complex latent spaces



Conclusion

Conclusion

Summary

- We have investigated how autoencoders process simple geometric shapes
  - Size can be extracted with a simple averaging filter
  - Decoding requires the learning of a shape primitive, modulated by the latent space
- We investigated the encoding of a Dirac impulse
  - Can be done with a simple filter and subsampling
- We proposed an autoencoder methodology which encourages a latent space with two desirable properties
  - Statistical decorrelation of latent components (with covariance loss)
  - Ordering of latent components wrt to reconstruction error (progressive increase of latent space size)
- Can be applied a posteriori to organise pretrained complex latent spaces

Further work

- We would like to understand how an autoencoder can decode more complex shapes (curves etc)
- Work remains to understand the autoencoder decoding process for position
  - How does the autoencoder place an object in an image ?
- Questions remain with respect to the PCA autoencoder
  - Increase of latent space size by packets
  - Is the PCA autoencoder too local to apply to the entire space of complex generative models ?
  - Is it preferable/possible to learn first and organise afterwards ?

References of this work

- Alasdair Newson, Andrés Almansa, Yann Gousseau, Saïd Ladjal, *Processing Simple Geometric Attributes with Autoencoders*, JMIV, 2019

- Saïd Ladjal, Alasdair Newson, Chi-Hieu Pham, A PCA-like Autoencoder, *A PCA-like autoencoder*, arXiv:1904.01277, 2019

Thank you !

Position autoencoder induction proof

Initialisation

- In the case of **one hidden layer**, the property is easy to show. There are two cases:
  1. $\delta_0 = [1, 0]$: then $\varphi * \delta_0 = [2, 1] \implies E^1(\delta_0) = 2$
  2. $\delta_1 = [0, 1]$: then $\varphi * \delta_1 = [1, 2] \implies E^1(\delta_1) = 1$

Autoencoding position

Induction

- Suppose that E^L extracts the inverted position of $\delta_a \in \mathbb{R}^{2^L}$, so :

$$E^L(\delta_a) = 2^L - a \tag{17}$$

Autoencoding position

Induction

- Suppose that E^L extracts the inverted position of $\delta_a \in \mathbb{R}^{2^L}$, so :

$$E^L(\delta_a) = 2^L - a \tag{17}$$

- Furthermore :
  - The output of the network is a fixed positive linear combination of δ_a
  - Only one element of δ_a is non-zero
- Therefore, we can rewrite the output of the network as :

$$E^L(\delta_a) = \sum_{i=0}^{2^L-1} (2^L - i)\delta_a(i) \tag{18}$$

Autoencoding position

Induction

- Suppose that E^L extracts the inverted position of $\delta_a \in \mathbb{R}^{2^L}$, so :

$$E^L(\delta_a) = 2^L - a \tag{17}$$

- Furthermore :
  - The output of the network is a fixed positive linear combination of δ_a
  - Only one element of δ_a is non-zero
- Therefore, we can rewrite the output of the network as :

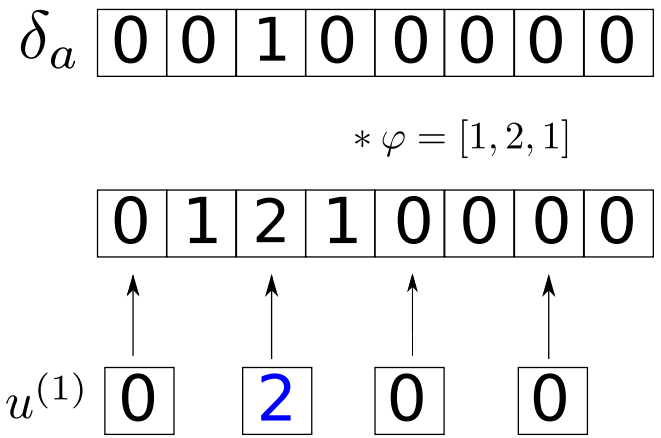
$$E^L(\delta_a) = \sum_{i=0}^{2^L-1} (2^L - i)\delta_a(i) \tag{18}$$

- Now, suppose that we add another layer to the network
- There are three cases of a to distinguish between : even, odd, or at end

Induction - case 1

1. a is an even position, so that $\exists k \in \mathbb{N}$, s.t. $a = 2k$. Thus, we have :

$$
\begin{aligned}
E^{L+1}(\delta_a) &= \sum_{i=0}^{2^L-1} (2^L - i) u^{(1)}(i) \\
&= \left(2^L - k\right) \cdot 2 \\
&= 2^{L+1} - 2k
\end{aligned}
$$



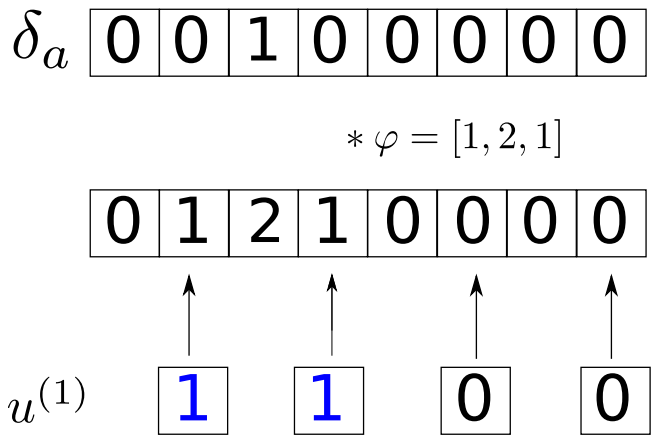
- The first case (a even) is verified

Autoencoding position

Induction

3. a is an odd position, so that $\exists k \in \mathbb{N}$, s.t. $a = 2k+1$. Thus, we have :

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



- The second case (a odd) is verified

Autoencoding position

Induction

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

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

Autoencoding position

Induction

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

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

- Conclusion : the network E, a simple filter/subsampling network, extracts the position information from a Dirac impulse
  - Also works for any $\varphi = c[1, 2, 1]$, $c \neq 0$
- This result easily generalises to 2D since the two directions can be processed independently