Understanding Geometric Attributes with Autoencoders

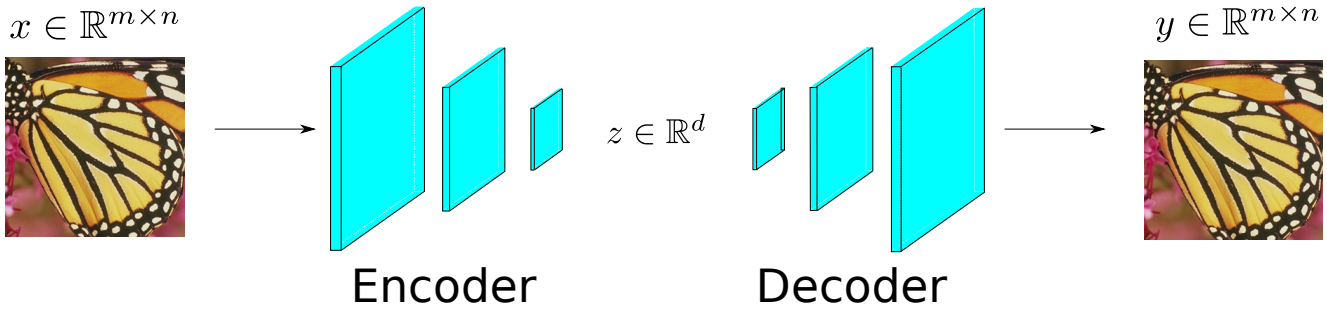
Alasdair Newson

Télécom ParisTech
alasdair.newson@telecom-paristech.fr

April 3, 2019

Subject of this talk

Understanding geometric attributes of images with Autoencoders



- Subjects of this talk
  1. Understand how **autoencoders** can encode/decode basic geometric attributes
     - Size
     - Position
  2. Propose an autoencoder algorithm which effectively separates different image attributes in the latent space
     - PCA-like autoencoder
     - Encourage meaningful interpolation and navigation of the latent space

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

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

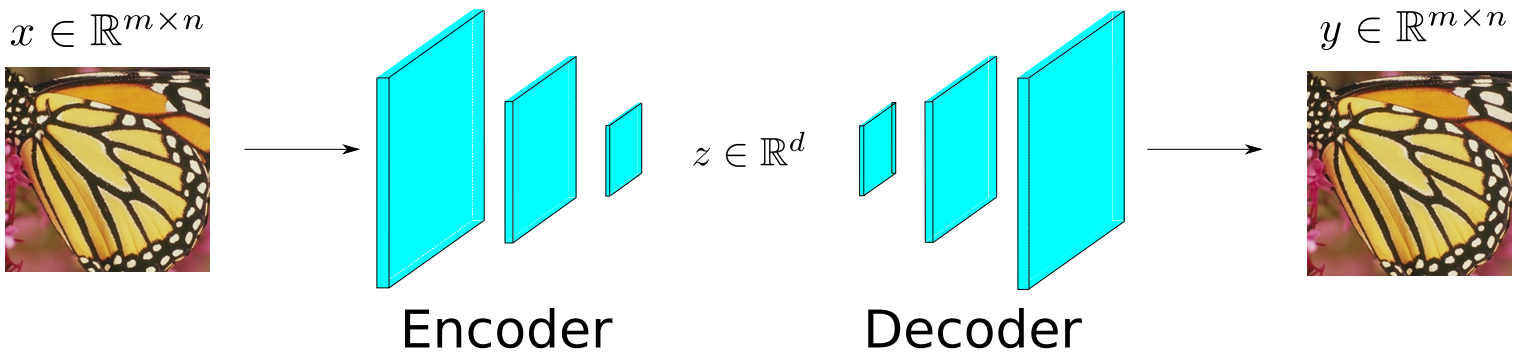
(ongoing) Goal(s) of this work ?

- Describe the mechanisms autoencoders use to encode/decode simple geometric shapes
- Propose an autoencoder architecture/algorithm where the latent space is interpretable
  - **Meaningful interpolation, navigation** of latent space

Autoencoders, introduction

What are autoencoders ?

- Autoencoder (AE) : neural network which **compresses** (encoding) and **decompresses** (decoding) some input information



- Often uses convolution and subsampling/upsampling
- Underlying goal : **learn the data manifold/space**

What are autoencoders used for ?

- Synthesis of high-level/abstract images

Synthesis examples from "Real NVP"†

- Produce impressive results, however, autoencoder mechanisms not necessarily understandood
- Our work attempts to understand underlying mechanisms, and create interpretable latent spaces

† ***Density estimation using Real NVP***, *L. Dinh, J. Sohl-Dickstein, S. Bengio, arXiv:1605.08803 2016*

Summary

Disk autoencoder : problem setup

Autoencoding size

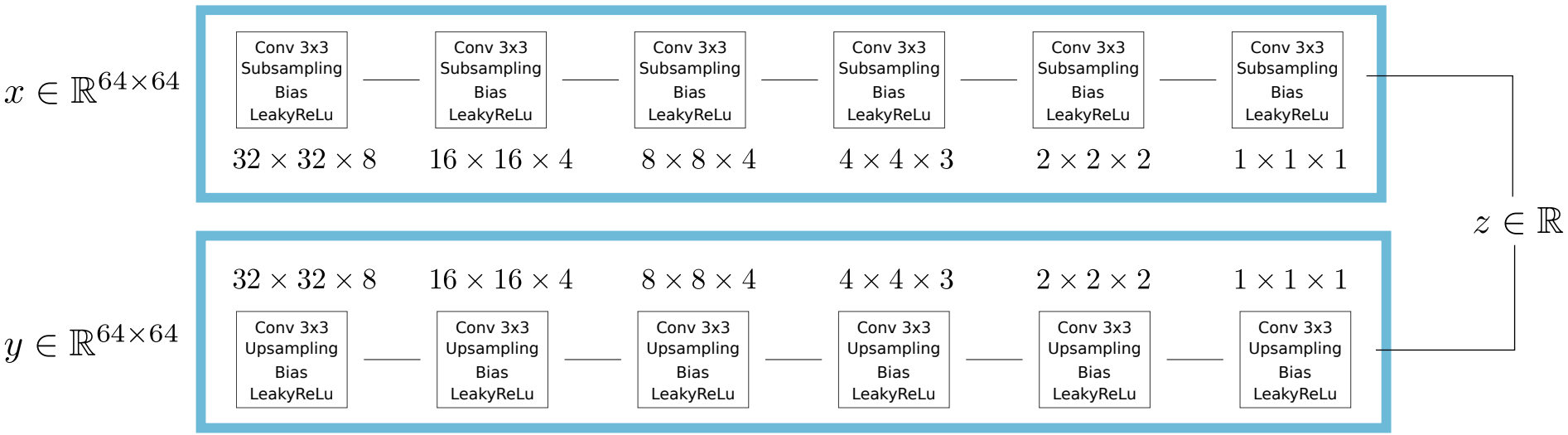
- Can AEs encode and decode a disk "optimally"; if so, how ?
- Training set : square, disk, images of size 64×64
  - Blurred slightly to avoid discrete parameterisation
- Each image contains one *centred* disk of random radius r

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

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

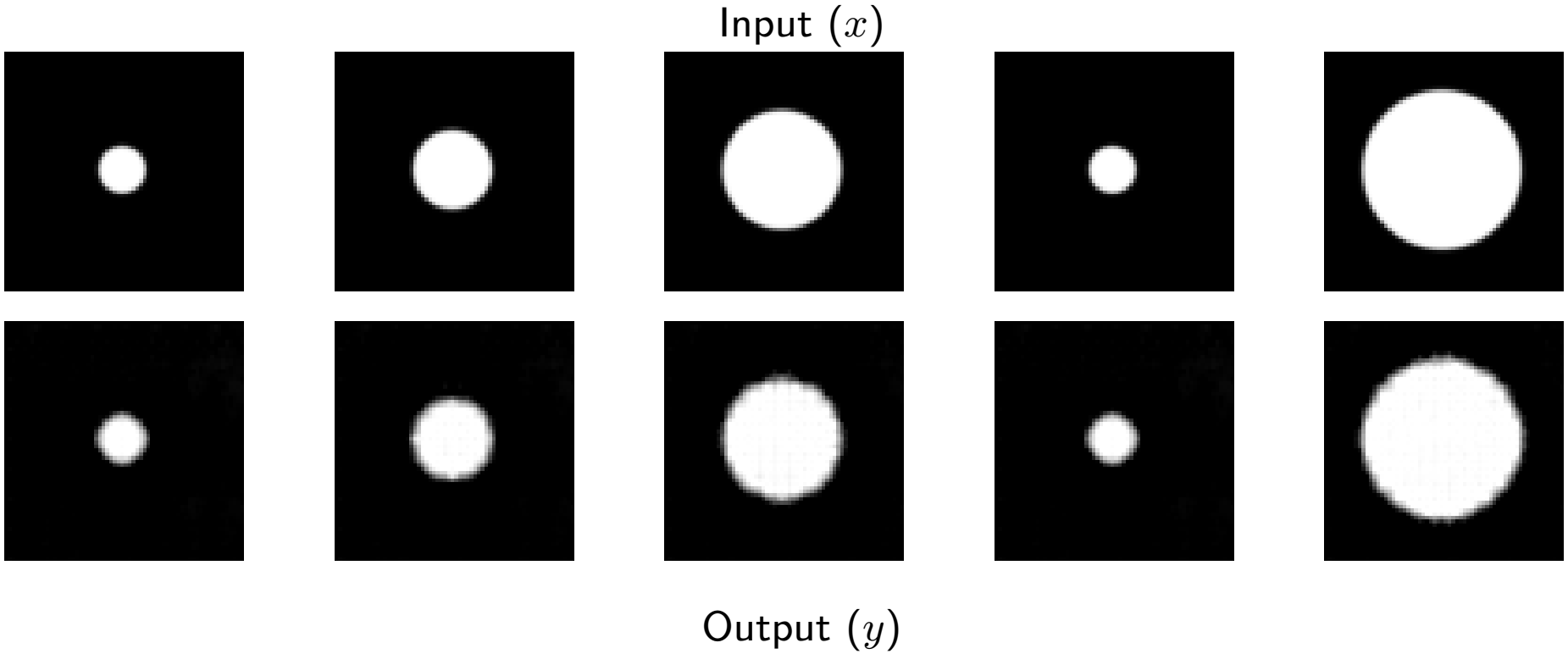
$$\hat{\Theta}_E, \hat{\Theta}_D = \underset{\Theta_E, \Theta_D}{\arg\min} \sum_{x_r} \| D \circ E\left(x_r\right) - x_r \|_2^2 \tag{1}$$

- 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 !

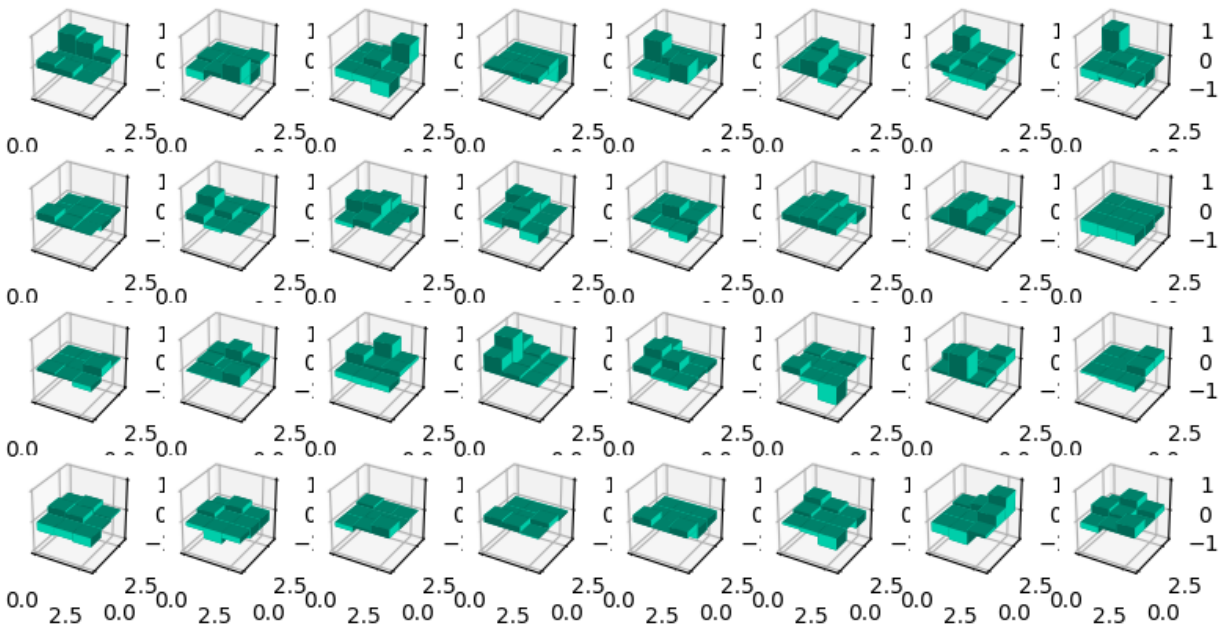


- Let us try to understand how this works

Investigating autoencoders

How does the autoencoder work in the case of disks ?

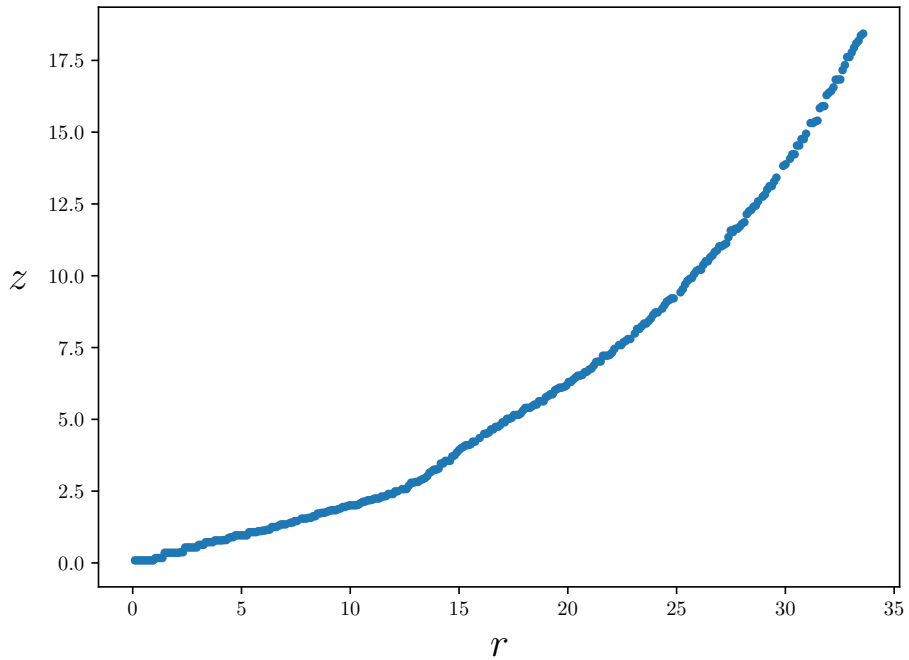
- First idea, inspect network *weights*;
- Unfortunately, very difficult to interpret;



Example of weights (3×3 convolutions)

Investigating autoencoders

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

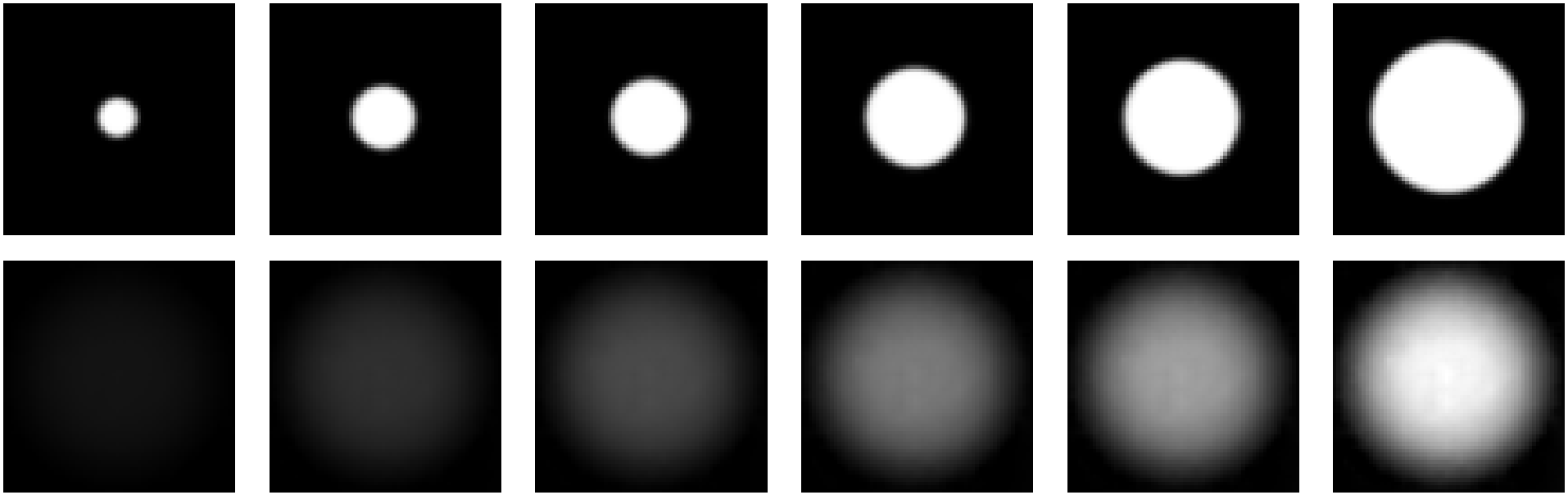


- Encoding relatively simple to understand : averaging filter
- How about decoding ?
  - Inspecting weights and biases is tricky
  - We can describe the decoding function when we **remove the biases** (ablation study)

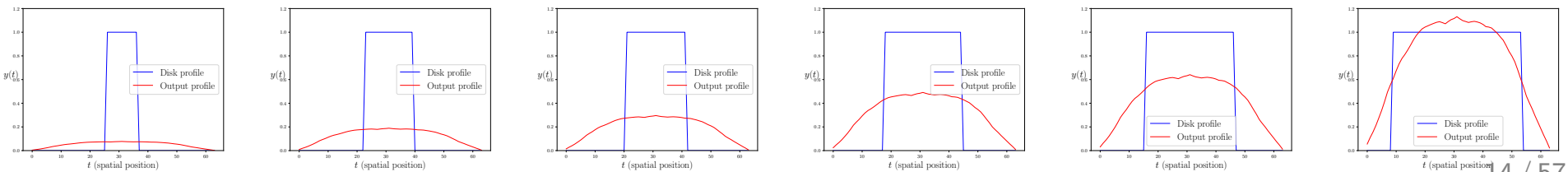
Decoding a disk

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

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

Investigating autoencoders

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 $\|f\|_2^2 = 1$
- Since the disks are **radially symmetric**, the integration can be simplified to one dimension
- This gives us the final result :

$$\hat{f} = \arg\min_f \int_0^R -\langle f, \mathbb{1}_{B_r} \rangle_2^2 \, dr \tag{6}$$

$$= \arg\max_f \int_0^R \langle f, \mathbb{1}_{B_r} \rangle_2^2 \, dr. \tag{7}$$

Decoding a disk

- We insert the optimal $\hat{h}(r)$, and choose the (arbitrary) normalisation $\|f\|_2^2 = 1$
- Since the disks are **radially symmetric**, the integration can be simplified to one dimension
- This gives us the final result :

$$\hat{f} = \arg\min_f \int_0^R -\langle f, \mathbb{1}_{B_r} \rangle_2^2 \, dr \tag{6}$$

$$= \arg\max_f \int_0^R \langle f, \mathbb{1}_{B_r} \rangle_2^2 \, dr. \tag{7}$$

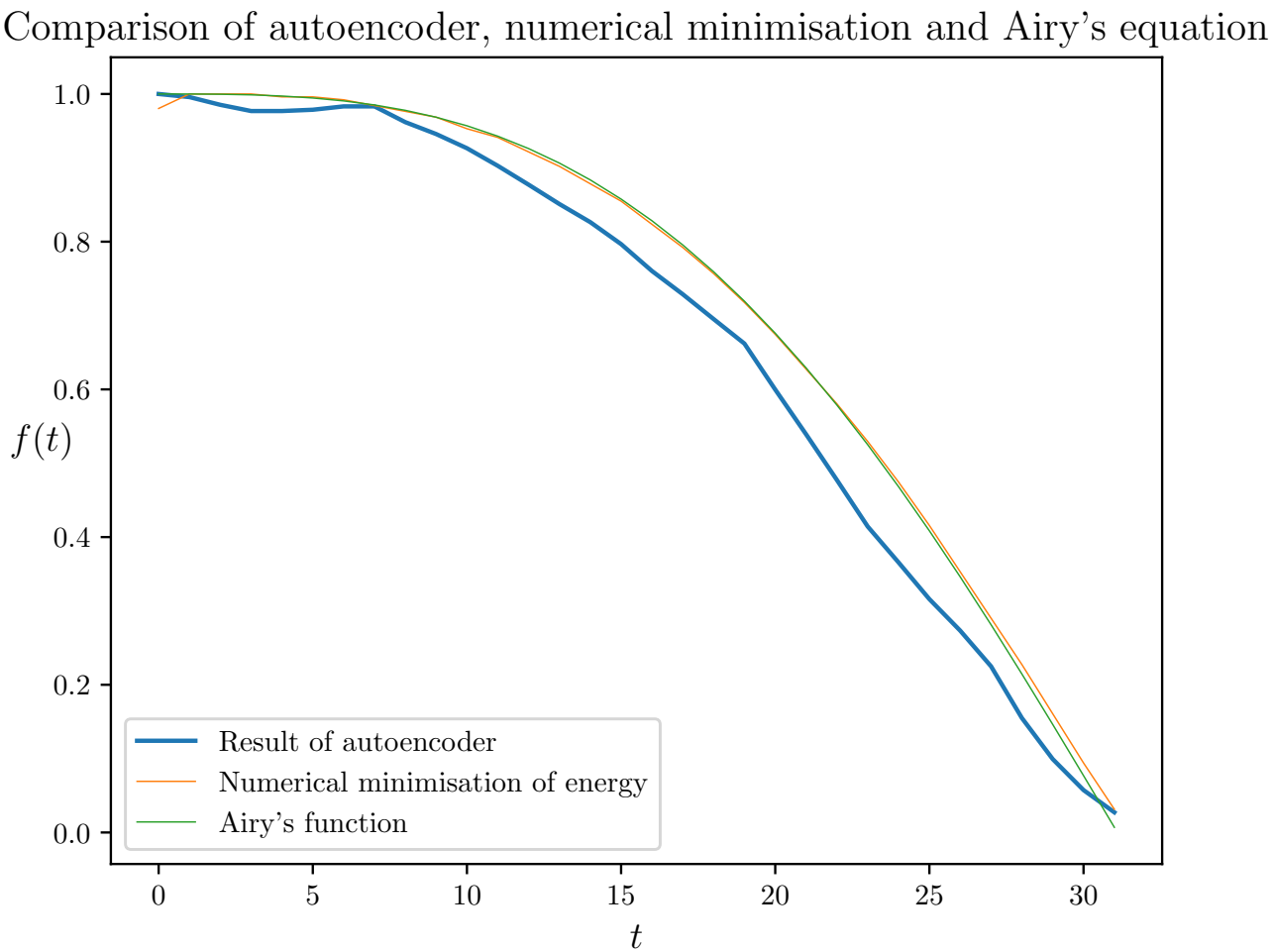
The first variation of the functional in Equation (3) leads to a differential equation, Airy's equation.

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

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

Decoding a disk

- The functional is indeed minimised by the training procedure;



Decoding a disk

Summary of disk encoder/decoder

- Encoder : integration (averaging filter) sufficient
- Decoder : a function learned, scaled and thresholded
- Further work : apply to **general scaling**

Scaled mnist data

Decoding a disk

Further questions

- What happens when samples are missing from the database ?

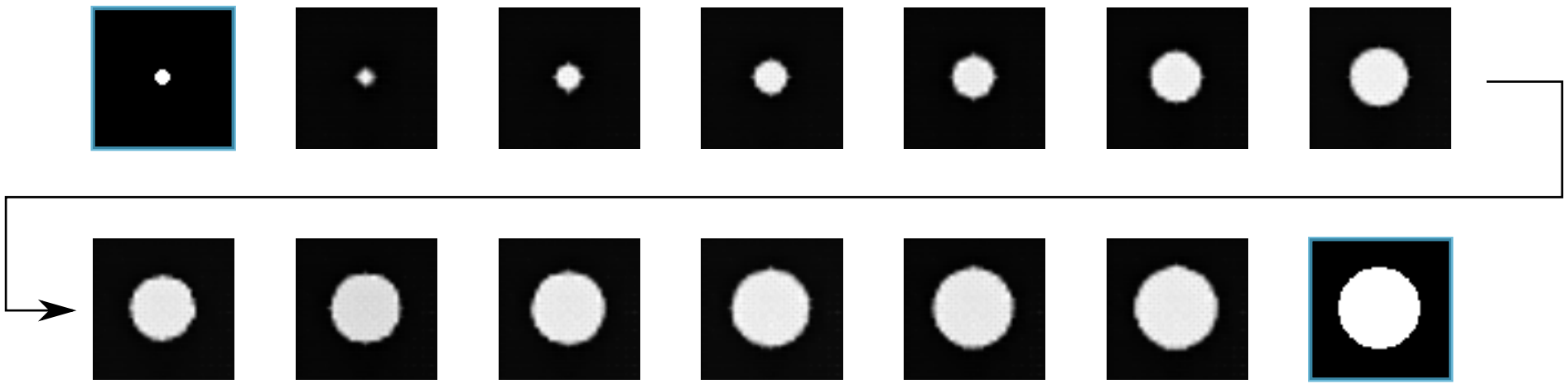
Image synthesis results of "Real NVP"[†]

- Is it possible to **interpolate** in the latent space ?

[†] ***Density estimation using Real NVP***, *L. Dinh, J. Sohl-Dickstein, S. Bengio, arXiv:1605.08803 2016*

Investigating autoencoders

Interpolation of disks in the learned space



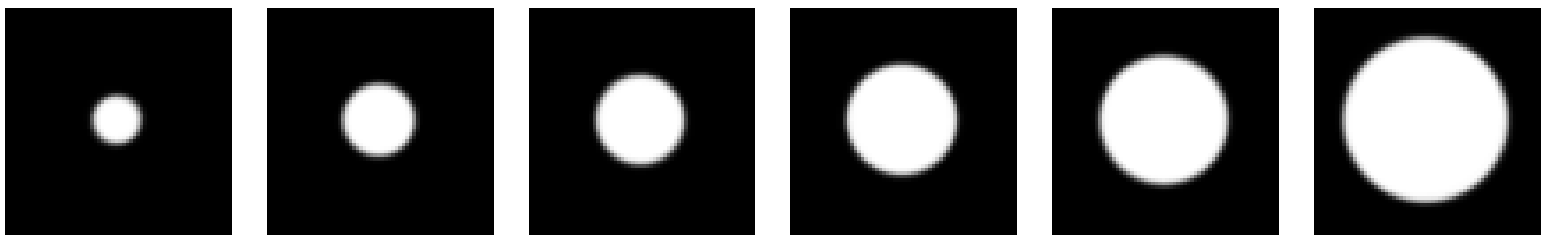
Effect of linearly increasing z

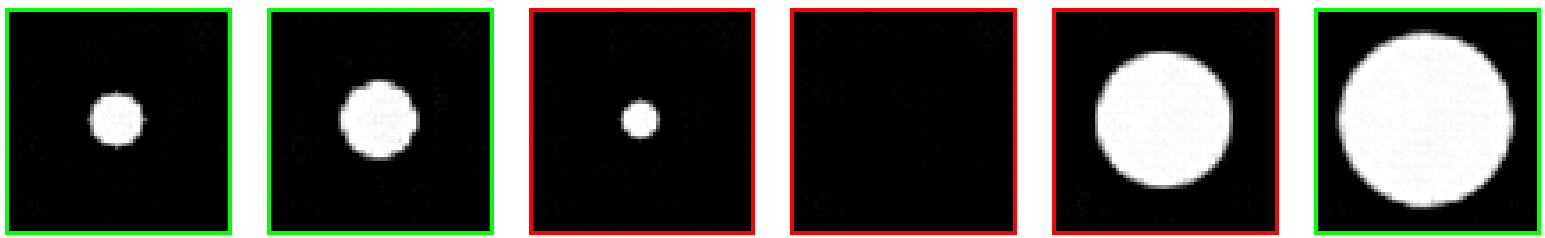
- Interpolation in the latent space is meaningful here
- What about **interpolating inside** unobserved regions in data set ?

Interpolating disks

- We trained our AE with missing radii of 11-18 pixels

Input

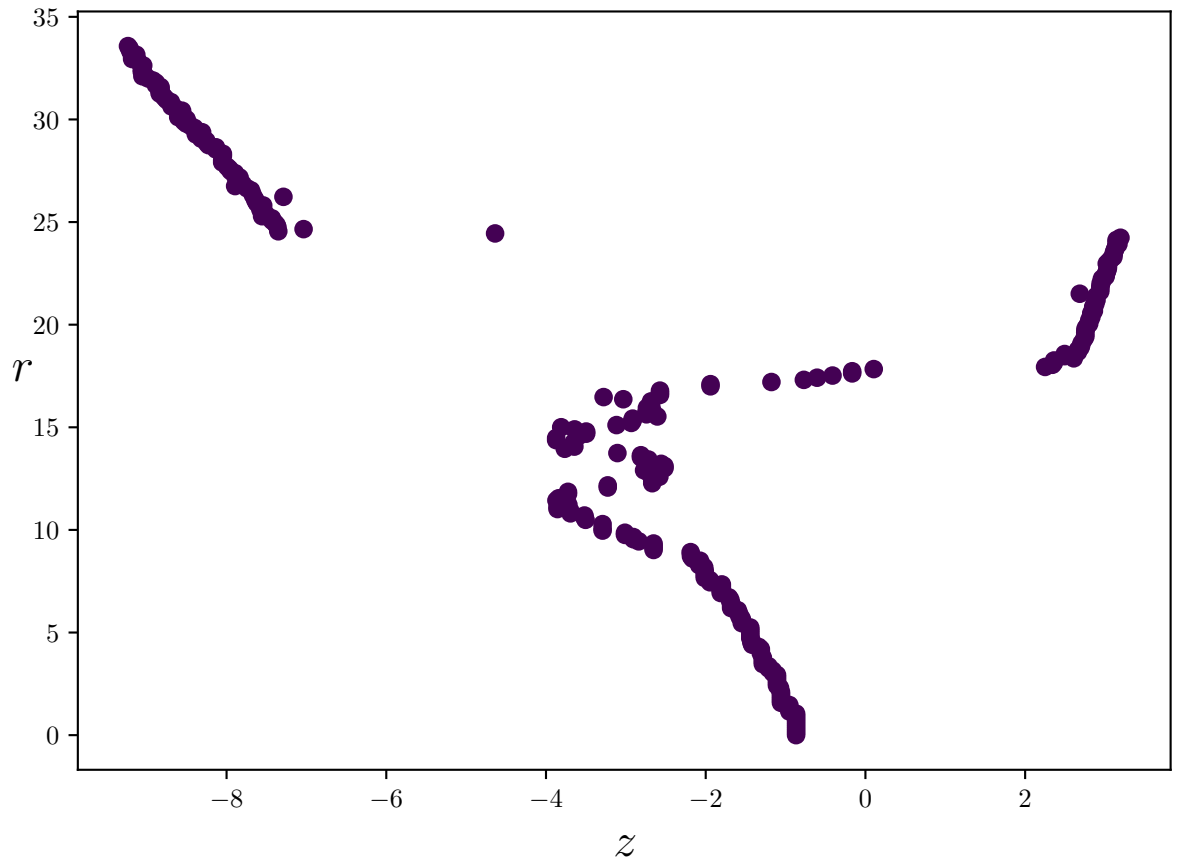




Output

Interpolation

What is this due to ? Inspect latent space



- How can this be remedied ? Regularisation of latent space

Interpolation

Various regularisation approaches available

- Maintaining norm between objects in latent space
- Denoising AEs etc.
- Regularising weights

Interpolation

Various regularisation approaches available

- Maintaining norm between objects in latent space
- Denoising AEs etc.
- Regularising weights

ℓ_2-regularisation in latent space (type 1)

$$(\|x - x'\|_2^2 - \|E(x) - E(x')\|_2^2)^2$$

Denoising autoencoder (type 2)

$$\sum_{\ell=1}^{L} \|D(E(x + \eta)) - x\|_2^2$$

Weight regularisation, of encoder (type 3)

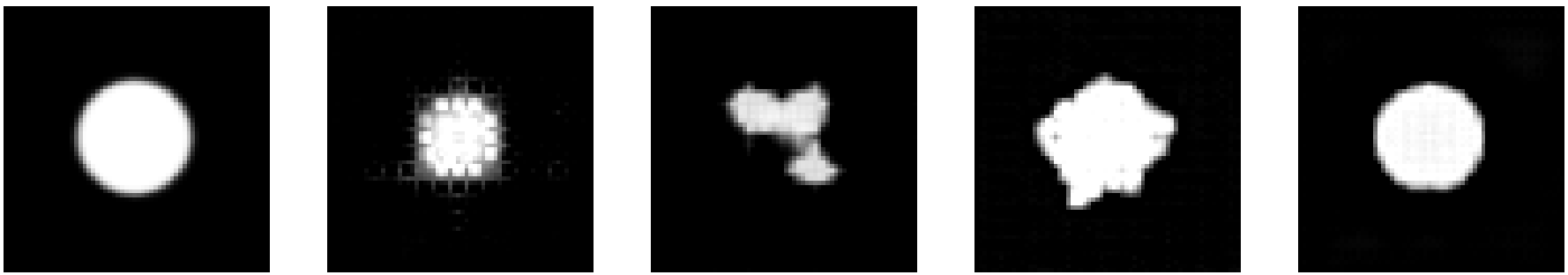
$$\sum_{\ell=1}^{L} \|W^\ell\|_2^2$$

Interpolation

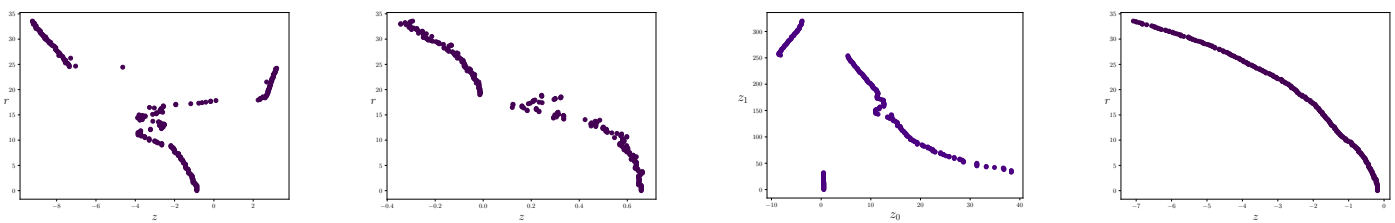
Interpolating unknown radii

Input

Output



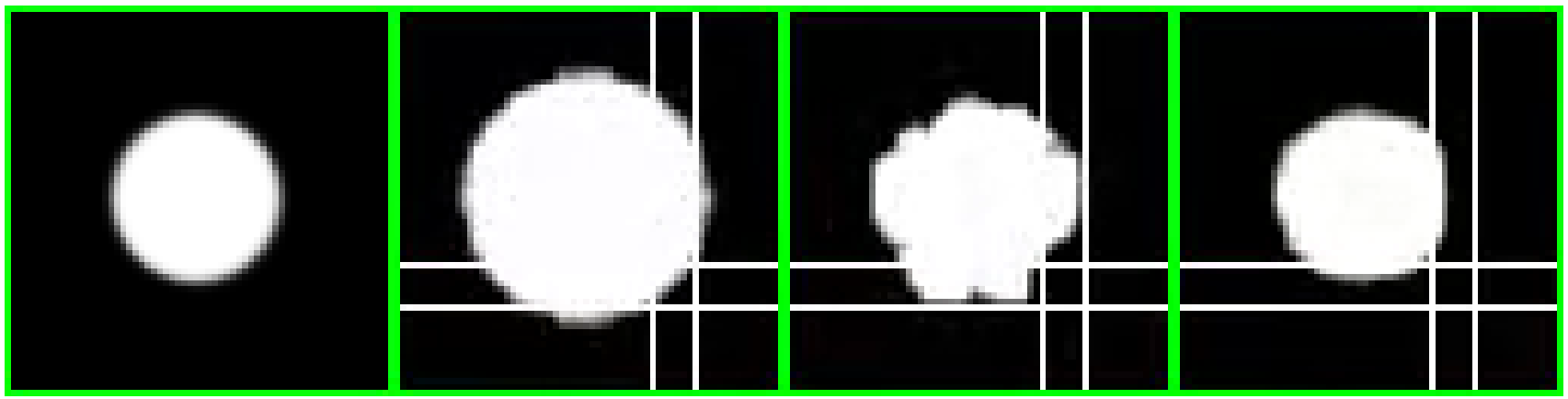
Learned manifold



No reg. Neighbours Denoising Encoder

Interpolating disks

Interpolating unknown radii



Input Neighbours Denoising Encoder

- Regularisation is crucial for correct generalisation, even in simple cases
- Regularisation of the latent space, via the **encoder**
  - Decoder can be left without regularisation

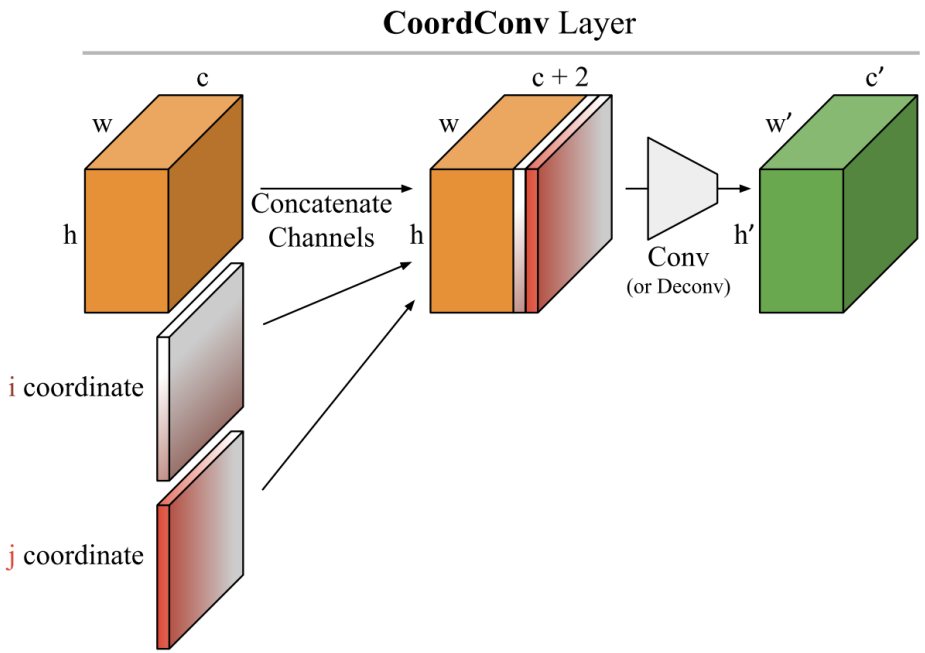
Summary

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 work concentrates 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"

* *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

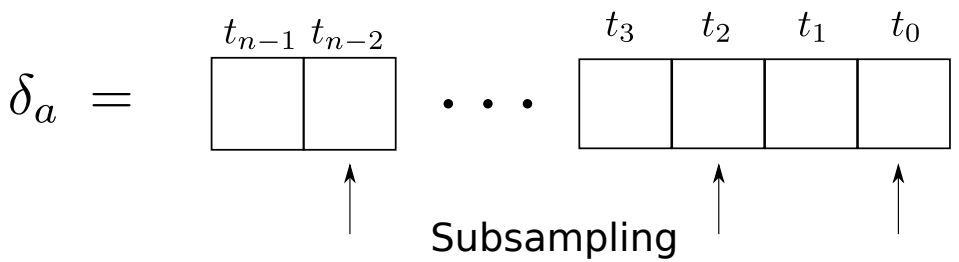
- We first studied the capacity of an autoencoder to **encode** position
- Consider the 1D case of a one-hot vector δ_a (a Dirac impulse), with a 1 at position a, with $a = 0, \ldots, n-1$

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

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

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

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

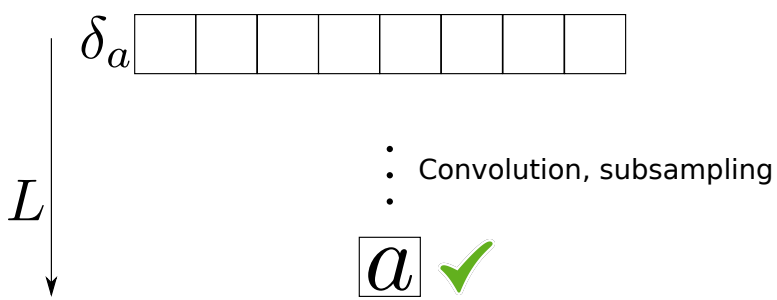
- Let E^L refer to the linear network created by a cascade of filtering and subsampling with filter φ and subsampling $\frac{1}{2}$
- The network E^L indeed **extracts the (inverted) position** of a from δ_a, $E^L(\delta_a) = 2^L - a$

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

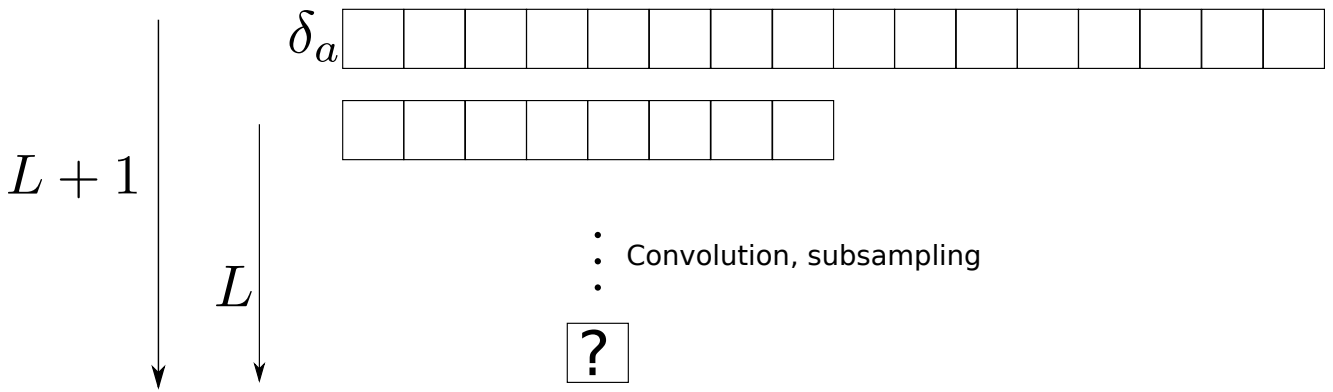


Autoencoding position

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



Initialisation

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

Autoencoding position

Induction

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

$$E^L(\delta_a) = 2^L - a \tag{11}$$

Autoencoding position

Induction

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

$$E^L(\delta_a) = 2^L - a \tag{11}$$

- 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 :

$$E^L(\delta_a) = \sum_{i=0}^{2^L-1} (2^L - i)\delta_a(i) \tag{12}$$

Autoencoding position

Induction

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

$$E^L(\delta_a) = 2^L - a \tag{11}$$

- 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 :

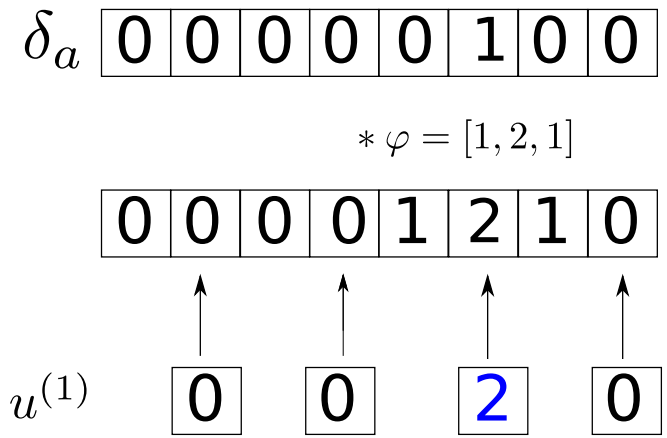
$$E^L(\delta_a) = \sum_{i=0}^{2^L-1} (2^L - i)\delta_a(i) \tag{12}$$

- 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 :

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

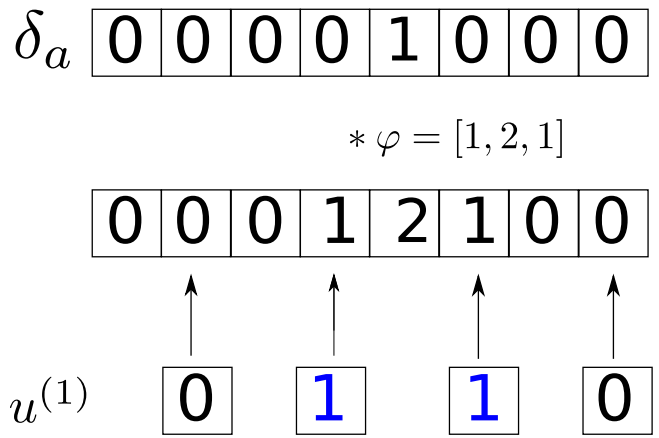


- The first case (a even) is verified

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

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)

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

Induction

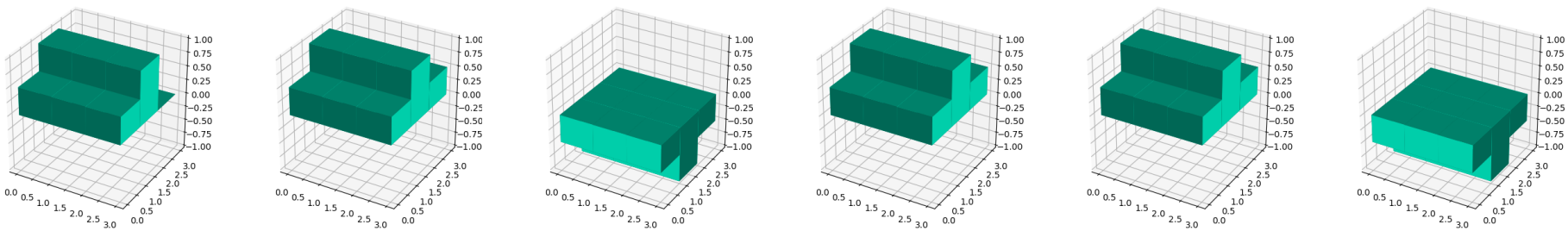
4. Finally, there is a special case where $a = 2^{L+1} - 1 = 2k + 1$, with $k = 2^L - 1$ (the 1 is placed at the end of the vector)

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

- 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

Autoencoding position

- The predicted weights are indeed found during training of an encoder of position
- Here are the (normalised) weights :



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

$$\delta_a(t) = \begin{cases} 1 - (x - \lfloor x \rfloor) & \text{if} \quad t = \lfloor x \rfloor \\ 1 - (\lceil x \rceil - x) & \text{if} \quad t = \lceil x \rceil \\ 0 \quad \text{otherwise} \end{cases} \tag{13}$$

- However, this modifies the framework of our encoder analysis. A complete analysis thus remains for further work

Summary

PCA Autoencoder

- Autoencoders extract the essential information of data, and represent this in the latent space
- However, latent space is often poorly understood
  - It is not clear what each of the axes in the latent space mean
  - Components can be **correlated** (we want independence)

* *G. Lample et al, Fader networks: Manipulating images by sliding attributes, NIPS, 2017*

PCA Autoencoder

- Autoencoders extract the essential information of data, and represent this in the latent space
- However, latent space is often poorly understood
  - It is not clear what each of the axes in the latent space mean
  - Components can be **correlated** (we want independence)
- A key issue for generative networks : many works propose to **interpolate samples in the latent space**
- Some works try to isolate certain attributes in the latent space, eg. "Fader Networks"*

* *G. Lample et al, Fader networks: Manipulating images by sliding attributes, NIPS, 2017*

PCA Autoencoder

- However, this approach requires annotated data
  - Autoencoder is non-supervised
- Ideally, we want an autoencoder which **separates different image characteristics in latent space** (disentanglement)
  - We want meaningful latent space interpolation and manipluation

[*] *Zhu et al, Generative Visual Manipulation on the Natural Image Manifold , ECCV, 2016*

PCA Autoencoder

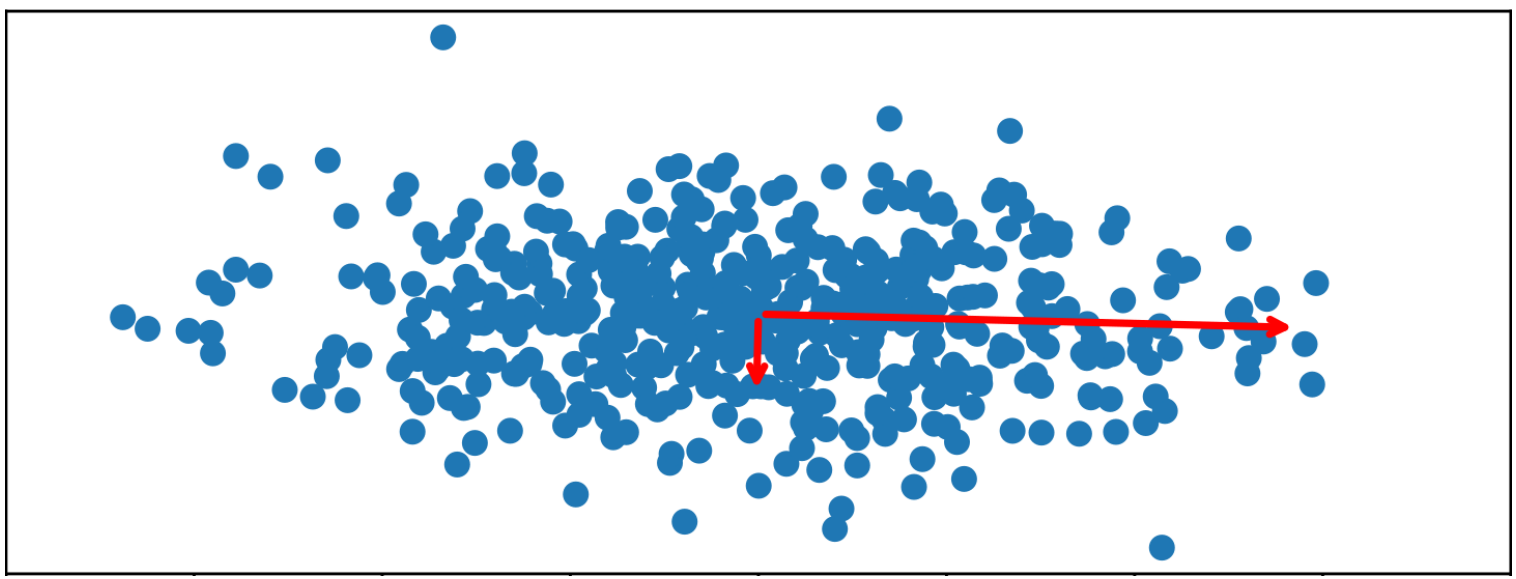
- However, this approach requires annotated data
  - Autoencoder is non-supervised
- Ideally, we want an autoencoder which **separates different image characteristics in latent space** (disentanglement)
  - We want meaningful latent space interpolation and manipluation
- We want navigation in the latent space to correspond to modifying image attributes (geometry, colour etc)

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

* *Zhu et al, Generative Visual Manipulation on the Natural Image Manifold , ECCV, 2016*

PCA Autoencoder

- You have probably noticed that the autoencoder bears much ressemblence to PCA
- There are two major differences between the autoencoder and the PCA
  - The autoencoder is a *non-linear transformation*, whereas the PCA is a linear one
  - The PCA's axes are ordered in decreasing "importance". This increases interpretability of the latent space



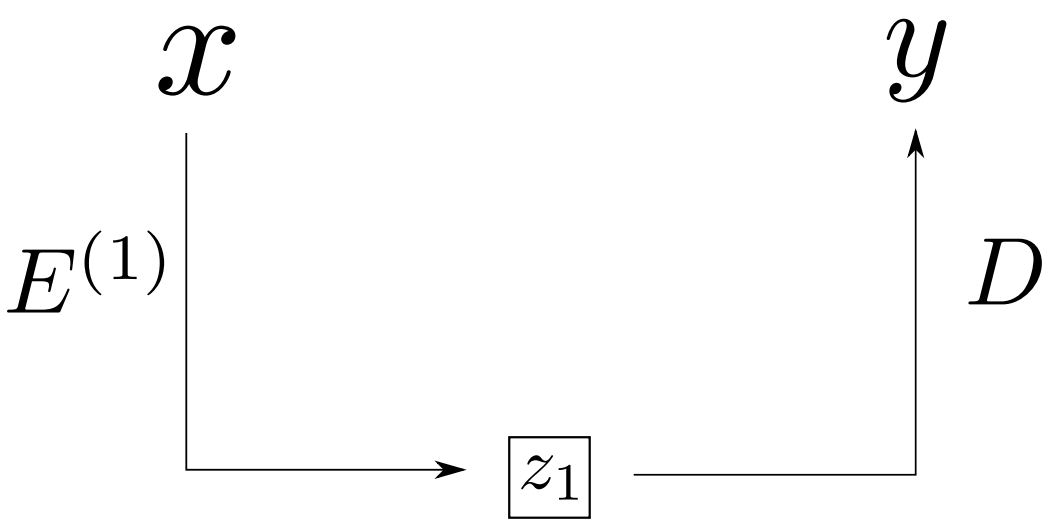
PCA Autoencoder

- Ideally, we would like to impose two criteria on the latent space :
  1. Increasing importance of components
  2. Independence of components
- We propose a PCA-like autoencoder which aims to achieve this, through two means :
  1. **Progressively increasing** the latent space size to capture most important variabilities in data
  2. A **covariance loss term** to ensure independence of latent components

PCA Autoencoder

PCA autoencoder architecture

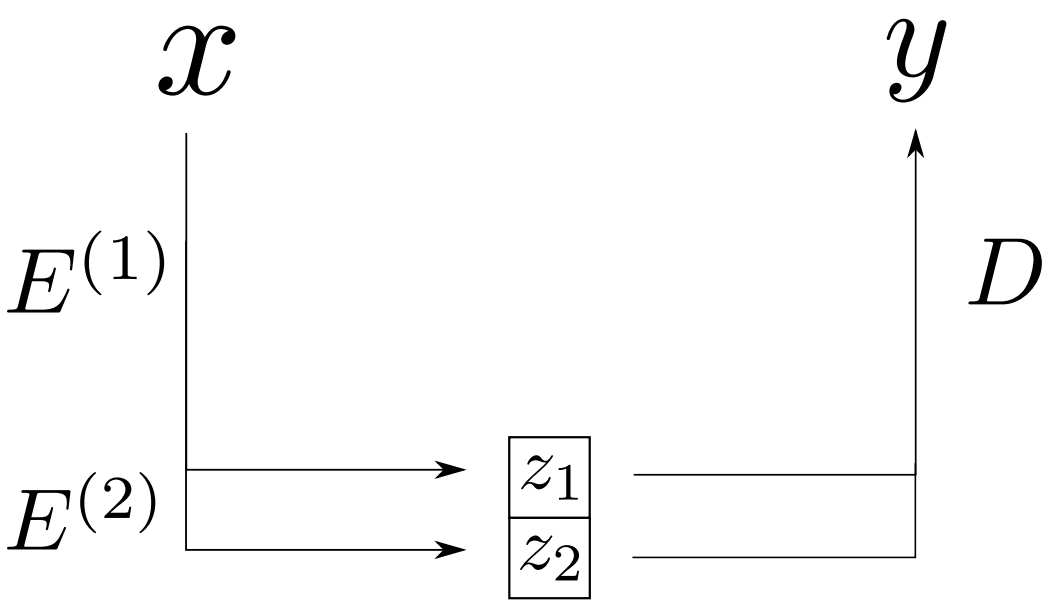
- Each encoder $E^{(i)}$ is trained, and then fixed
- At each iteration, the decoder is thrown away, and a new one is trained



PCA Autoencoder

PCA autoencoder architecture

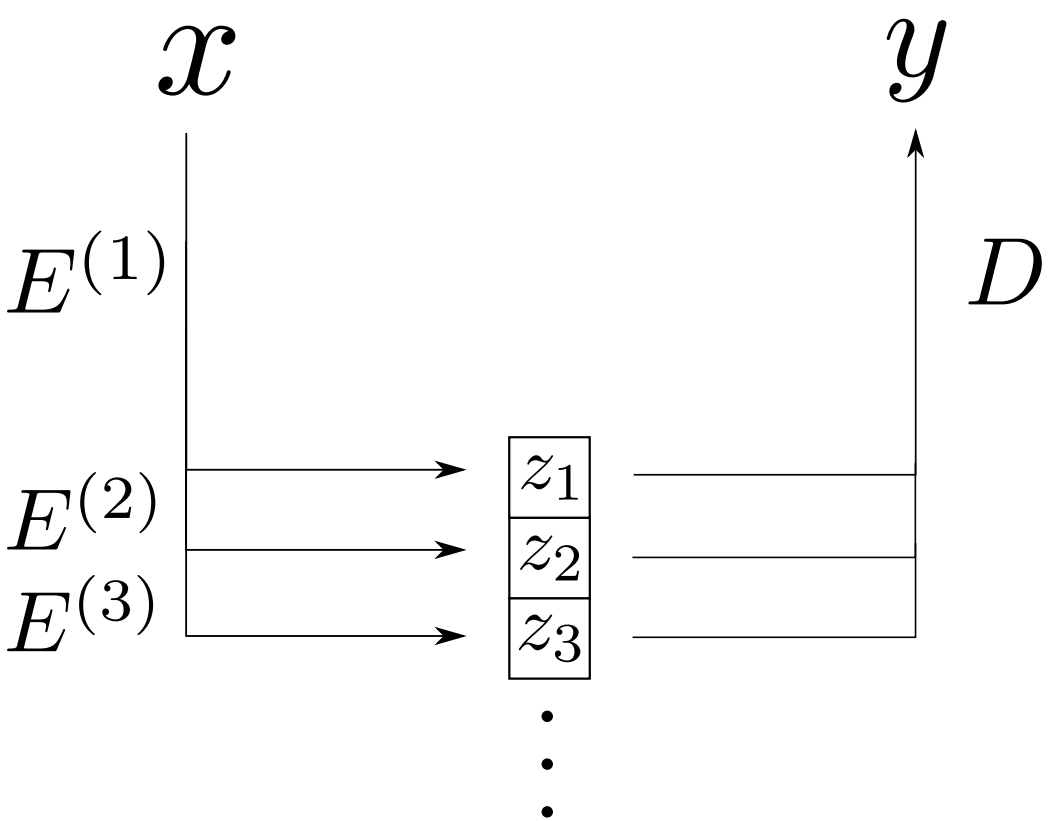
- Each encoder $E^{(i)}$ is trained, and then fixed
- At each iteration, the decoder is thrown away, and a new one is trained



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 independent
  - The goal is to improve interpretability
  - If components are independent, they likely represent different image attributes

PCA Autoencoder

- We want the components of the latent space to be independent
  - The goal is to improve interpretability
  - If components are independent, they likely represent different image attributes
- We minimise the **covariance** between latent variables :
  - $\text{Cov}(z_1, z_2) = \mathbb{E}[z_1 z_2] - \mathbb{E}[z_1]\mathbb{E}[z_2]$

$$\mathcal{L}_{\text{cov}}(z) = \left(\frac{1}{m} \sum_j (z_0^{(j)}\ z_1^{(j)}) - \frac{1}{m^2} \sum_j z_0^{(j)} \sum_j z_1^{(i)} \right)^2 \tag{14}$$

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

$$\mathcal{L}_{\text{cov}}(z) = \left(\frac{1}{m} \sum_j^m z_0^{(j)}\ z_1^{(j)} \right)^2 \tag{15}$$

PCA Autoencoder

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

$$\mathcal{L}_{\text{cov}}^{(k)}(z) = \frac{1}{k} \sum_{i=1}^{k-1} \frac{1}{m} \sum_{j=1}^{m} (z_i^{(i)} \; z_k^{(i)}) \tag{16}$$

- The total PCA autoencoder loss is

$$\mathcal{L}^{(k)}(x) = \|x - D \circ E^{(k)}(x)\|_2^2 + \lambda \mathcal{L}_{\text{cov}}^{(k)} \left(E^{(k)}(x) \right) \tag{17}$$

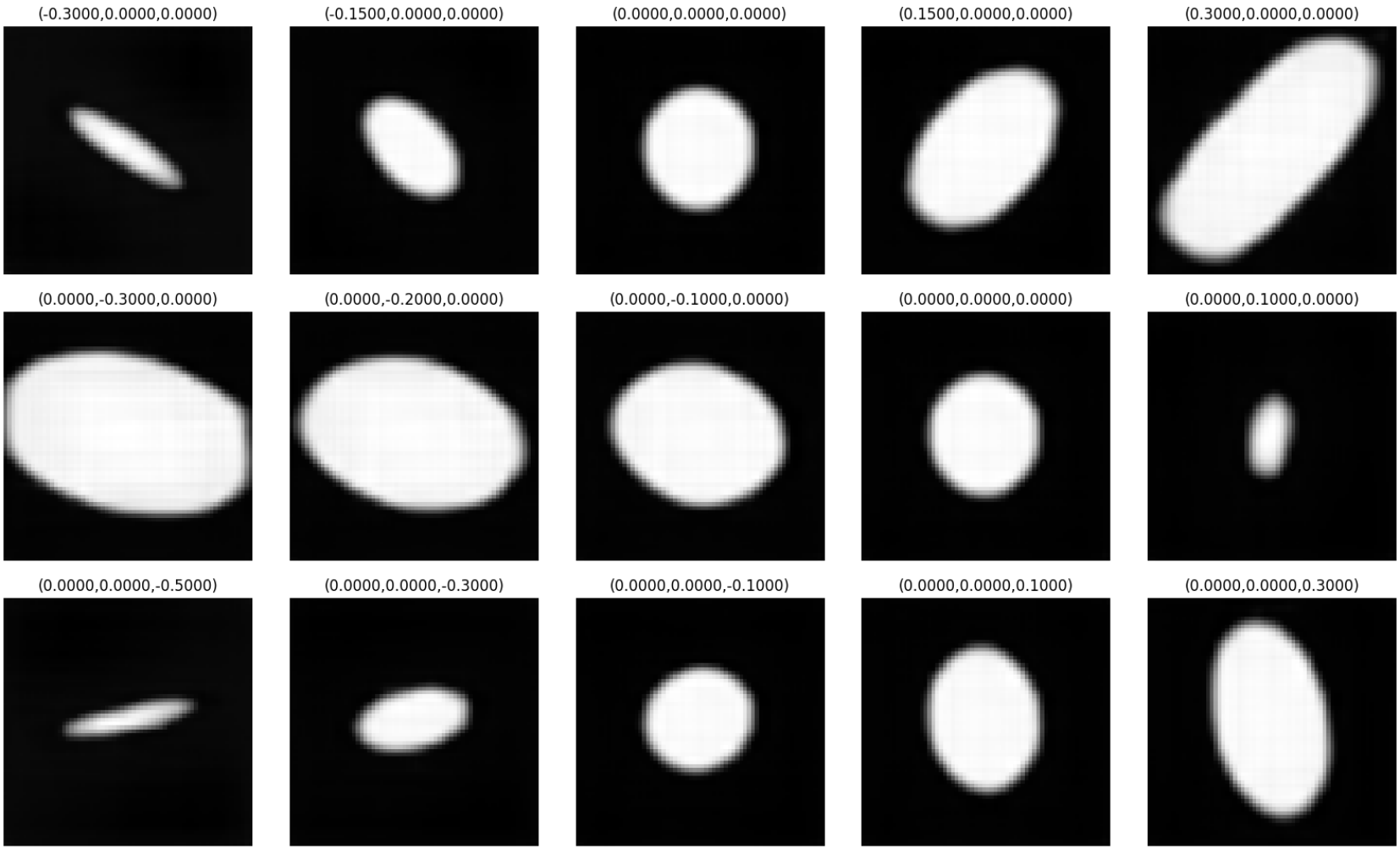
PCA Autoencoder - results

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

Ellipse dataset example images

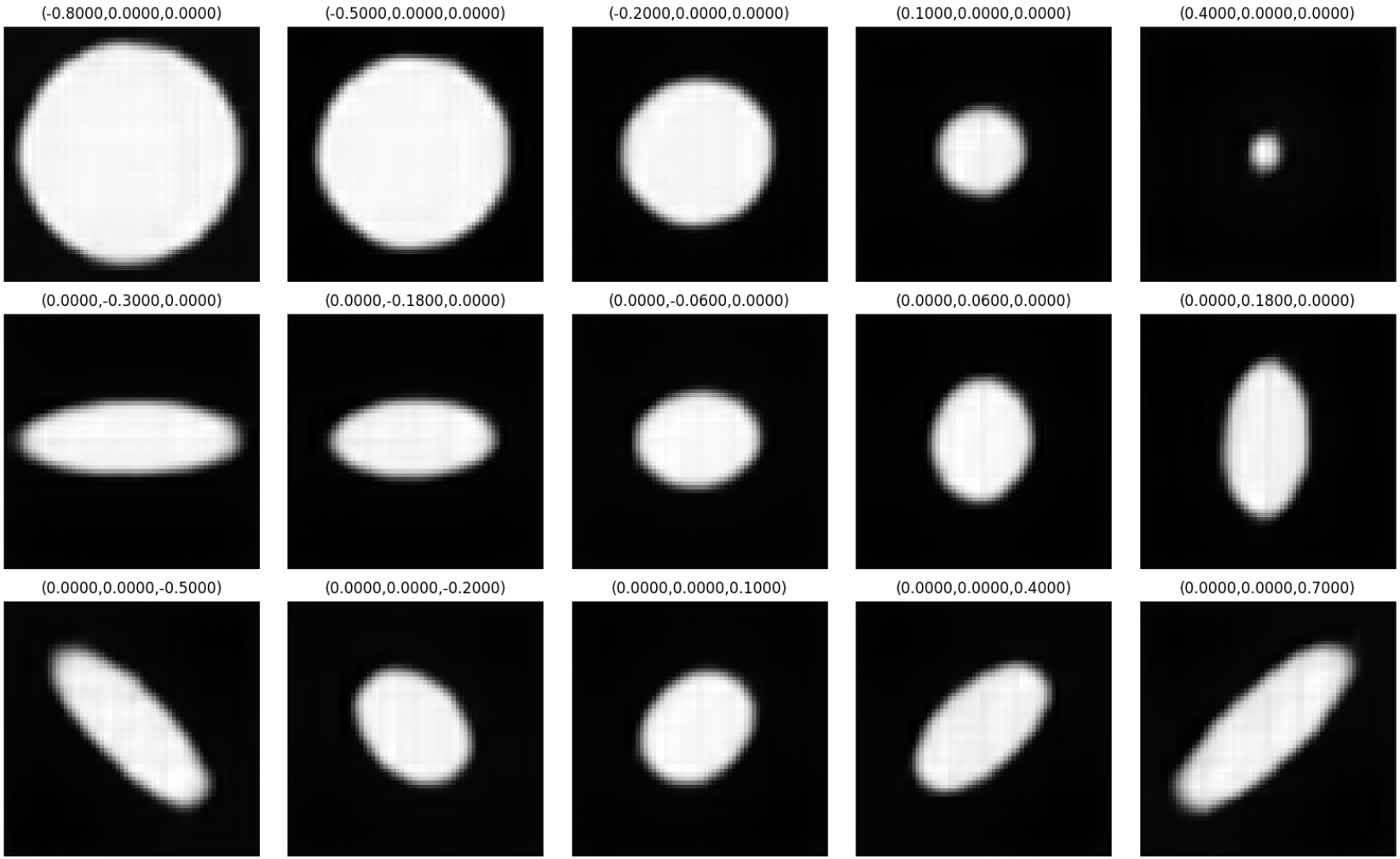
PCA Autoencoder - results

Latent space navigation - standard autoencoder



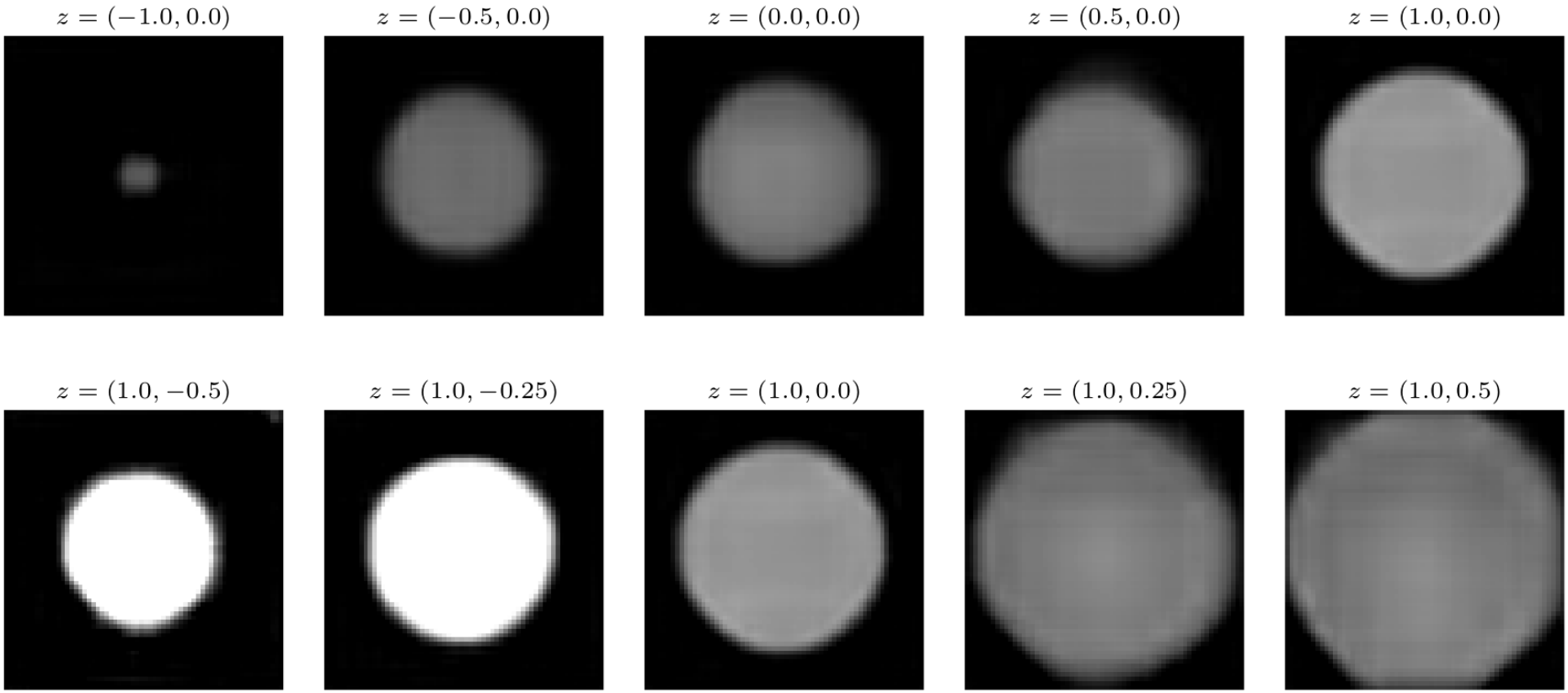
PCA Autoencoder - results

Latent space navigation - PCA autoencoder



PCA Autoencoder - results

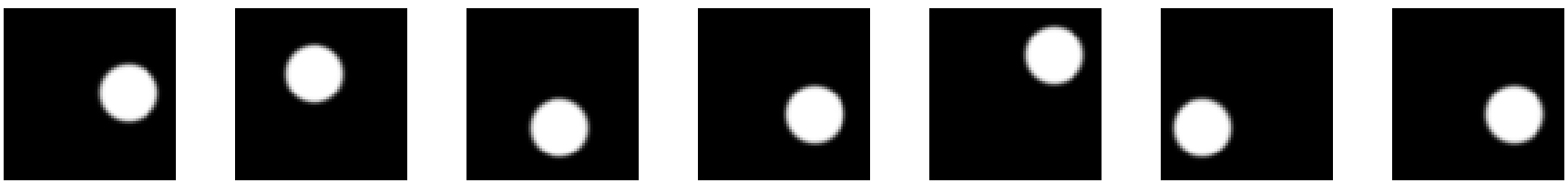
Disks with varying grey-level, failure case



- In this case, the second axis of the PCA autoencoder is not well-learned

PCA Autoencoder

- The PCA autoencoder allows for meaningful navigation and interpretation of the latent space
- However, there are certain drawbacks of our approach :
  - There are cases where progressively increasing the latent space might not work
  - Translation for example, where the autoencoder might need two coordinates to be trained together
  - Possible solution : increase the latent space size in code packets, rather than component by component



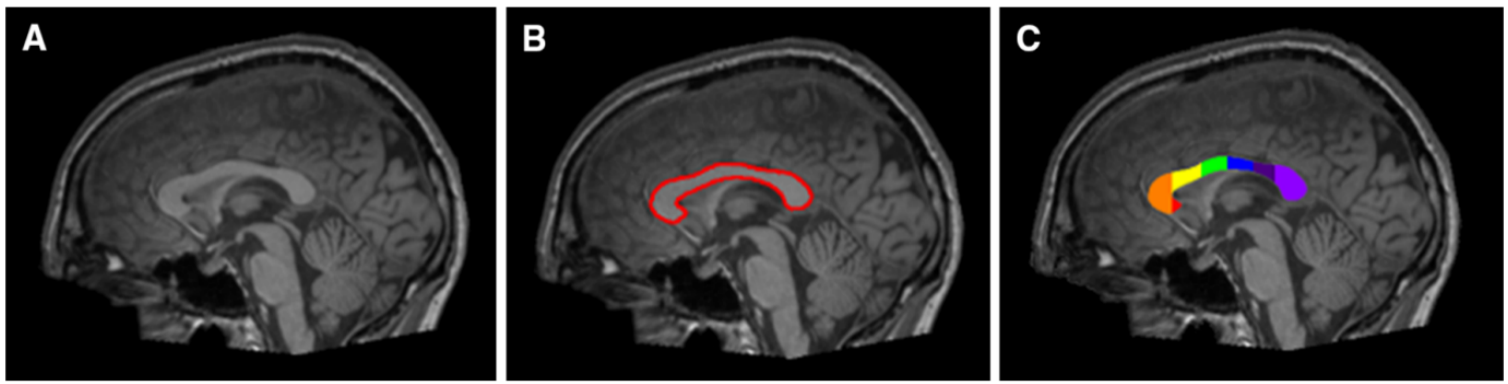
- Crucial : what kind of data can the PCA autoencoder deal with ?

Summary

Geometric autoencoder application

Application example : Corpus Callosum analysis

- Corpus callosum part of brain responsible for communication between hemispheres
- Recently linked to autism spectrum disorder[†]



- Automatic analysis of callosum geometry crucial
- Ongoing joint work with Pietro Gori (Télécom ParisTech)

[†] *Kucharsky Hiess, R., Alter R., Sojoudi S., Ardekani B.A., Kuzniecky, R., Pardoe H.R. Corpus callosum area and brain volume in autism spectrum disorder: quantitative analysis of structural MRI from the ABIDE database., Journal of Autism and Developmental Disorders, 2015.*

Geometric autoencoder application

Application example : Corpus Callosum analysis

- Example of segmented corpus callosum segmentation

- Goal : analyse latent space to extract geometrical markers
- Extract geometrical properties to predict illness

Conclusion

Conclusion

Summary

- Investigated how autoencoders process simple geometric shapes
- Proposed an autoencoder architecture and a covariance loss function which encourage independence and interpretability of the latent space

- Work remains to understand the autoencoder decoding process for position
  - How does the autoencoder place an object in an image ?
- Our work on PCA autoencoder has only been applied to simple, synthetic data
- Extend this to more complex data
  - Training of PCA autoencoder in small latent code packets, rather than component by component

Thank you !