LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Deeplearning

Chapter 6: Autoencoders

Bernd Bischl

Department of Statistics – LMU Munich

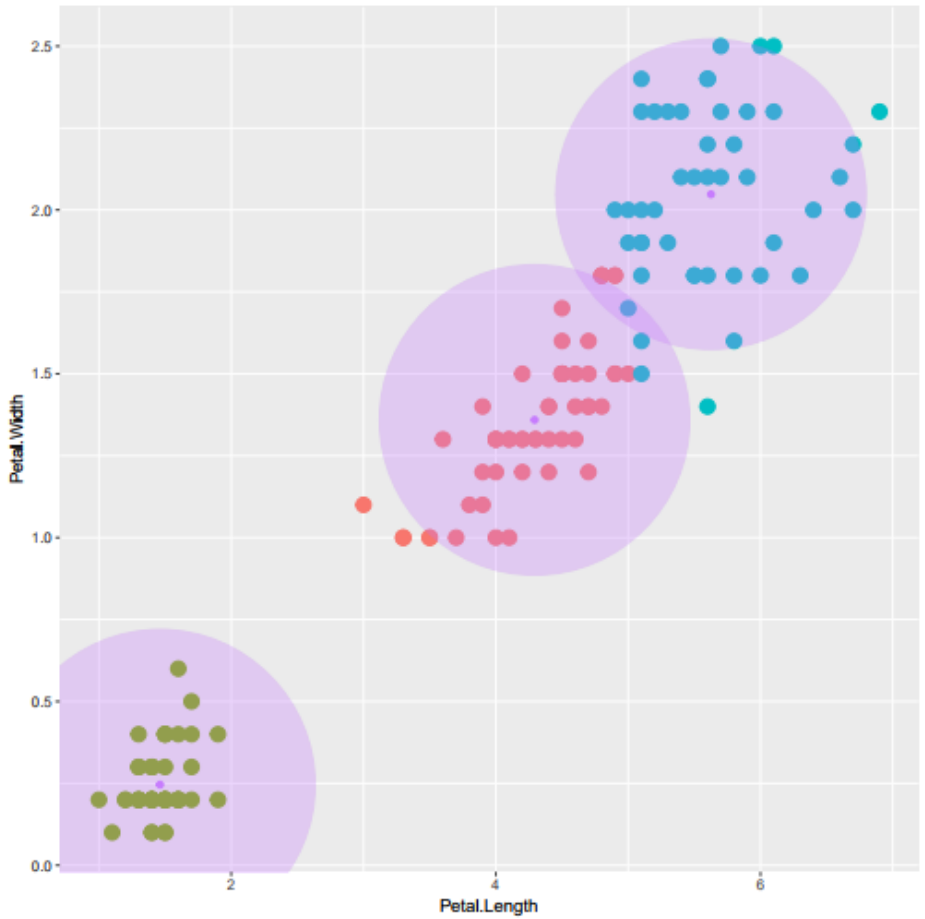
Winter term 2018

INTRODUCTION TO AUTOENCODERS

- So far we were dealing with different types of neural networks to infer classification and regression tasks.
- In the world of **supervised learning**, we exploit information of class memberships (or numeric values) to train our algorithm. That means in particular, that we have access to labeled data.
- Think of mnist:
  - We know the ground truth for each image of our dataset.
  - Our goal was to train a network that predicts the class labels.

INTRODUCTION TO AUTOENCODERS

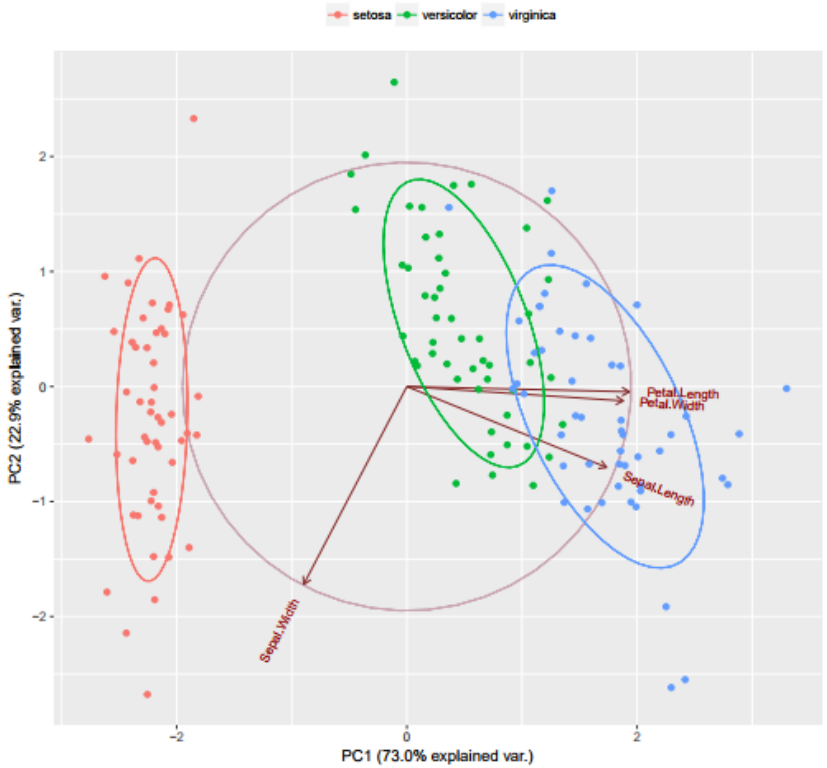
Clustering



INTRODUCTION TO AUTOENCODERS

Dimensionality reduction

- Principle Component Analysis (PCA)
- Linear Discriminant Analysis (LDA)
- Filter Methods



INTRODUCTION TO AUTOENCODERS

- The basic idea of an autoencoder is to obtain a neural network, that is able to **reconstruct its input**.

  $\Rightarrow$ Traditionally used for dimensionality reduction!
- (Very) simple example: given the input vector (1, 0, 1, 0, 0), an autoencoder will try to output (1, 0, 1, 0, 0).
- Therefore, we do not need class labels and pass over into the world of **unsupervised learning**.

REVISION OF PCA

- The purpose of PCA is to project the data $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}$ onto a lower-dimensional subspace (e.g. to save memory).
- For each point $\mathbf{x}^{(i)} \in \mathbb{R}^p$ we need to find a corresponding code vector $\mathbf{c}^{(i)} \in \mathbb{R}^l$ with $l < p$. That step is accomplished by the encoding function which produces the code for an input:

$$f(\mathbf{x}) = \mathbf{c}$$

- Additionally, we need a decoding function to produce the reconstruction of the input given its code:

$$\mathbf{x} \approx g(f(\mathbf{x}))$$

- We can choose matrix multiplication to map the data back into $\mathbb{R}^p$: $g(\mathbf{c}) = \mathbf{D}\mathbf{c}$, with $\mathbf{D} \in \mathbb{R}^{p \times l}$, defining the decoding.

REVISION OF PCA

- To keep the encoding problem easy, PCA constrains the columns of **D** to be orthogonal.
- One way to obtain the optimal code $\mathbf{c}^*$ is to minimize the distance between the input $\mathbf{x}$ and its reconstruction $g(\mathbf{c})$ (that means, linear transformation with minimum reconstruction error):

$$\mathbf{c}^* = \arg\min_{\mathbf{c}} ||\mathbf{x} - g(\mathbf{c})||_2^2$$

- Solving this optimization problem leads to

$$\mathbf{c} = \mathbf{D}^T\mathbf{x}$$

- Thus, to encode a vector, we apply the encoder function

$$f(\mathbf{x}) = \mathbf{D}^T\mathbf{x}$$

REVISION OF PCA

- We can also define the PCA as the reconstruction operation:

$$r(\mathbf{x}) = g(f(\mathbf{x})) = \mathbf{D}\mathbf{D}^T\mathbf{x}$$

- To find the encoding matrix $\mathbf{D}^*$, we minimize the Frobenius norm of the matrix of errors computed over all dimensions and points:

$$\mathbf{D}^* = \arg\min_{\mathbf{D}} \sqrt{\sum_{i,j} \left(x_j^{(i)} - r(x^{(i)})_j \right)^2}, \text{ subject to } \mathbf{D}^\mathbf{T}\mathbf{D} = \mathbf{I}_l$$

- for $l = 1$, $\mathbf{D}^*$ collapses to a single vector and we can rewrite the equation as

$$\mathbf{d}^* = \arg\min_{\mathbf{d}} ||\mathbf{X} - \mathbf{X}\mathbf{d}\mathbf{d}^T||_F^2, \text{ subjected to } \mathbf{d}^T\mathbf{d} = 1$$

- The optimal $\mathbf{d}^*$ is given by the eigenvector of $\mathbf{X}^T\mathbf{X}$ corresponding to the largest eigenvalue.

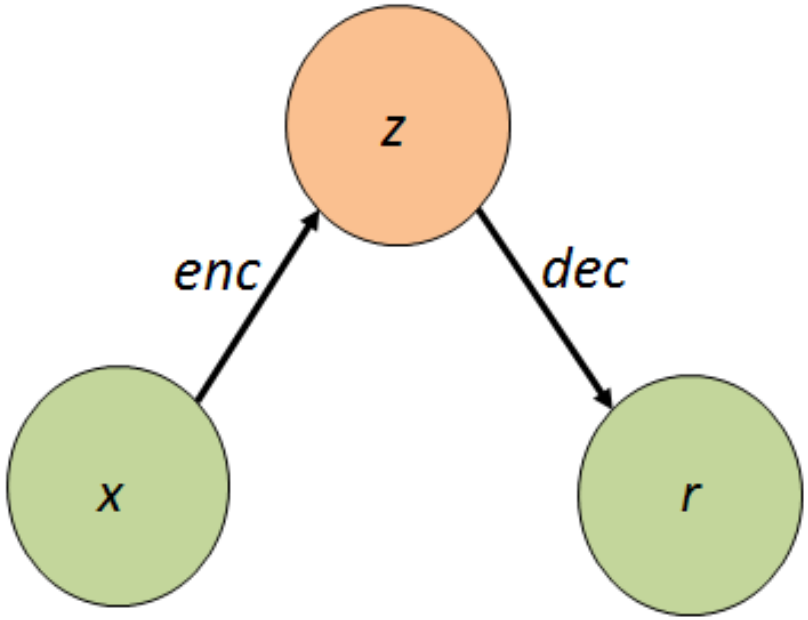
GENERAL STRUCTURE

- An autoencoder maps an input x to an output r (reconstruction)
- We can distinguish its general architecture in the following manner:
  - an encoder $z = enc(x)$ and
  - a decoder $r = dec(z)$

  to generate the reconstruction.
- We call z the **internal representation** or **code**

GENERAL STRUCTURE

The general structure of an autoencoder:



- An autoencoder has two computational steps:
  - the encoder *enc*, mapping *x* to *z*
  - the decoder *dec*, mapping *z* to *r*

USE CASE

- Practitioners utilize autoencoders, although they have access to labeled data.

  $\Rightarrow$ Why?
- Autoencoders may be used for:
  - Learning a representation of the input data
    $\Rightarrow$ dimensionality reduction & pretraining
  - Advanced: some variants of the autoencoder can be regarded as generative models
    $\Rightarrow$ may be used to draw synthetic data

UNDERCOMPLETE AUTOENCODERS

- If an autoencoder simply learns the identity $dec(enc(x)) = x$, it would be of no use. In fact, we want the autoencoder to learn "useful" or "significant" properties of the data.
- To extract such "useful" or "significant" properties, we introduce the **undercomplete autoencoder**.
- Therefore, we restrict the architecture, such that

$$dim(z) < dim(x)$$

  Or in other words: the hidden layer must have fewer neurons than the input layer.

  $\Rightarrow$ That will force the autoencoder to capture only the most salient features of the training data!
- Consequently, the undercomplete autoencoder tries to learn a "compressed" representation of the input.

UNDERCOMPLETE AUTOENCODERS

- Learning such a net is done by minimizing a loss function

$$L(x, dec(enc(x)))$$

  which penalizes the autoencoder $dec(enc(x))$ for being distinct of x.

    - Typical choice: MSE

- To carry out training, the very same optimization techniques of the previous chapters are applied (stochastic gradient descent, ..)

- How could a potential architecture of an undercomplete autoencoder look like for our (very) simple example?

  Reminder: $x = (1, 0, 1, 0, 0)$

UNDERCOMPLETE AUTOENCODERS

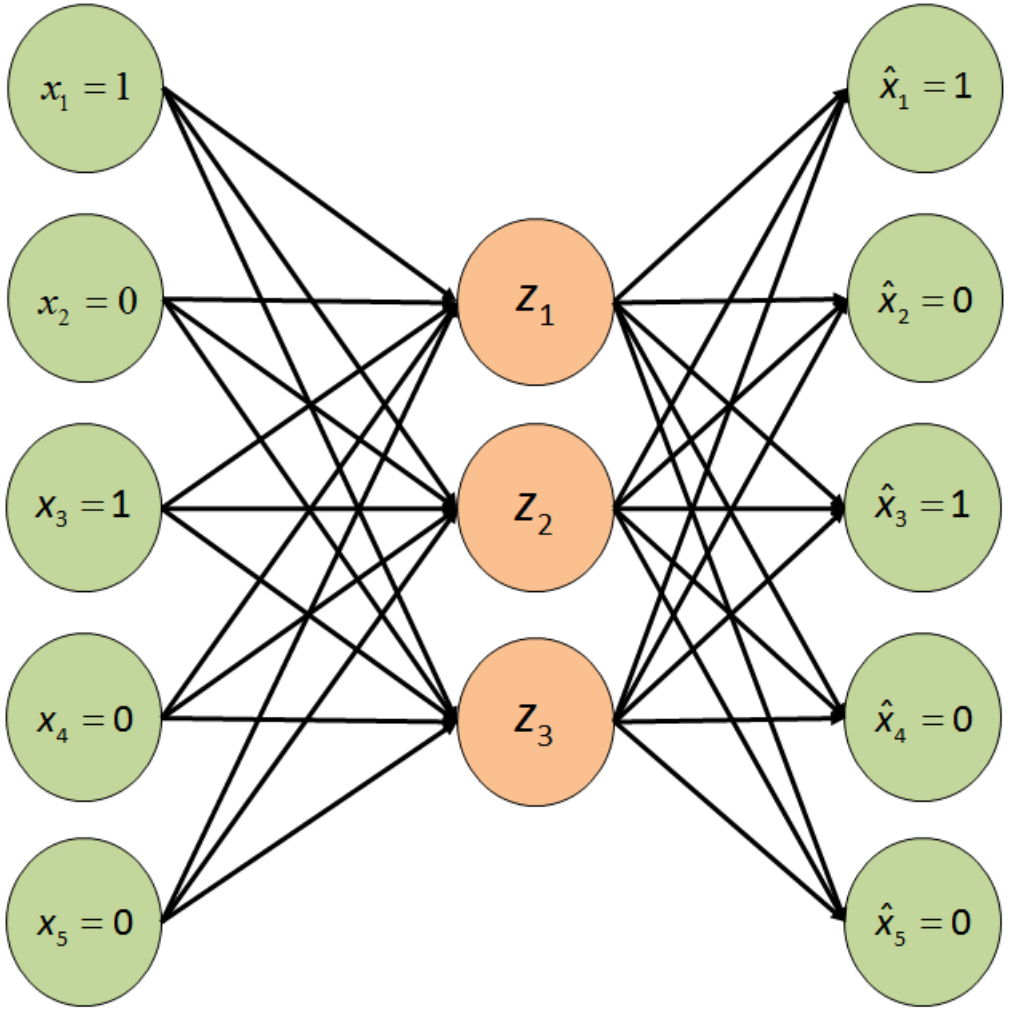


Figure: Potential architecture. On a first glimpse, the architecture looks just like a standard feed forward neural network.

EXAMPLE

- Let us try to compress the mnist data as good as possible.
- Therefore, we will fit an undercomplete autoencoder to learn the best possible representation,

  $\Rightarrow$ as few as possible dimensions in the internal representation h.

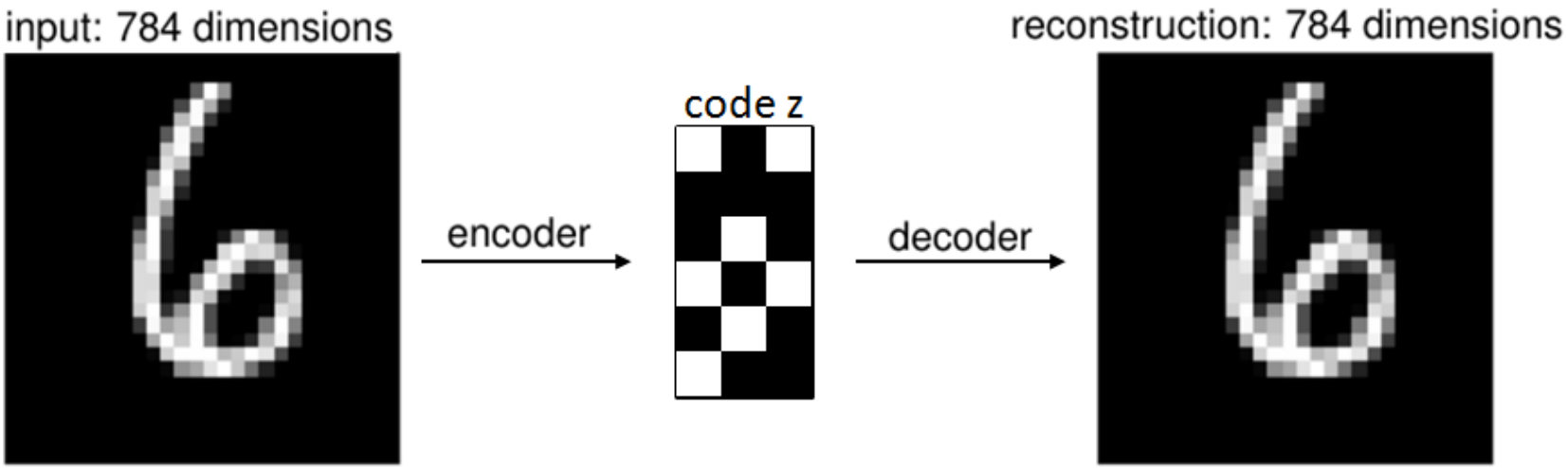


Figure: Flow chart of our our autoencoder: reconstruct the input with fixed dimensions $dim(z) << dim(x)$

EXAMPLE

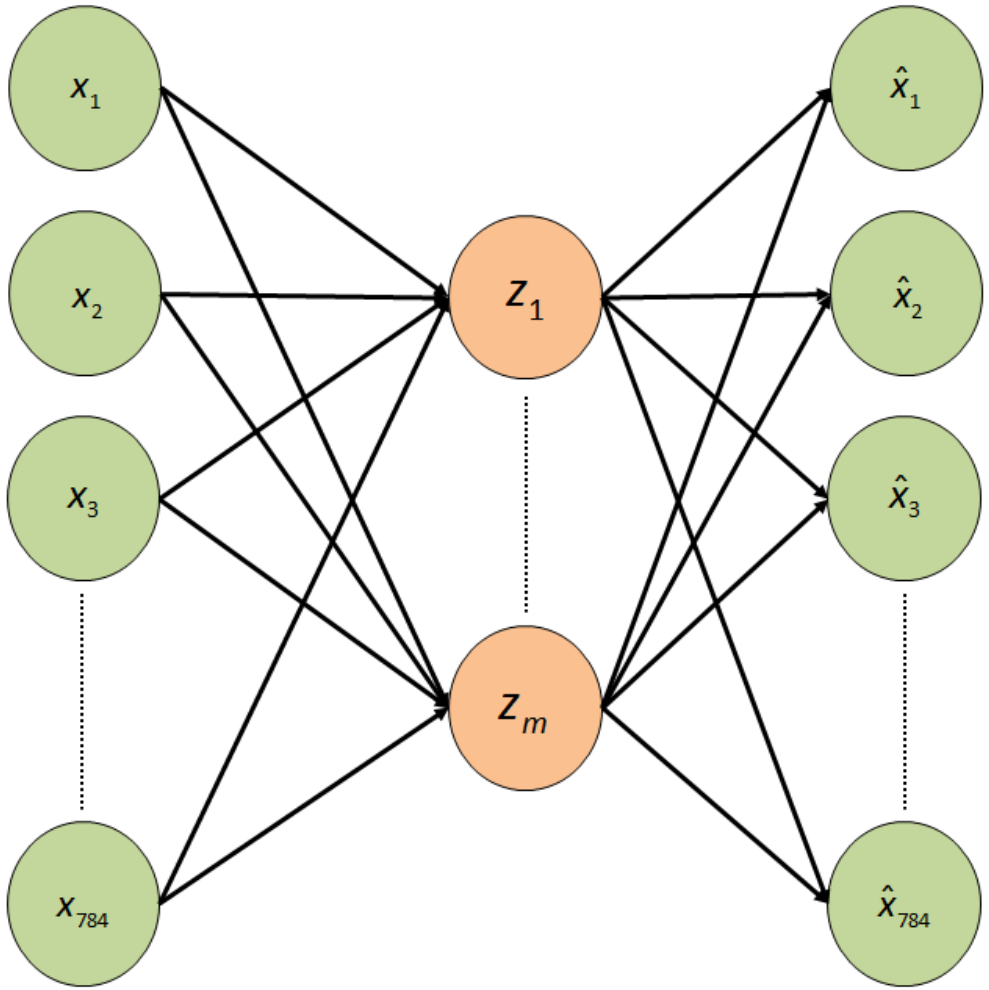


Figure: Architecture of the autoencoder.

EXAMPLE

```
z = c(784, 256, 64, 32, 16, 8, 4, 2, 1)

input = mx.symbol.Variable("data") # mnist with 28x28 = 784
encoder = mx.symbol.FullyConnected(input, num_hidden = z[i])
decoder = mx.symbol.FullyConnected(encoder, num_hidden = 784)
activation = mx.symbol.Activation(decoder, "sigmoid")
output = mx.symbol.LinearRegressionOutput(activation)

model = mx.model.FeedForward.create(output,
  X = train.x, y = train.x,
  num.round = 50,
  array.batch.size = 32,
  optimizer = "adam",
  initializer = mx.init.uniform(0.01),
  eval.metric = mx.metric.mse
)
```

EXAMPLE

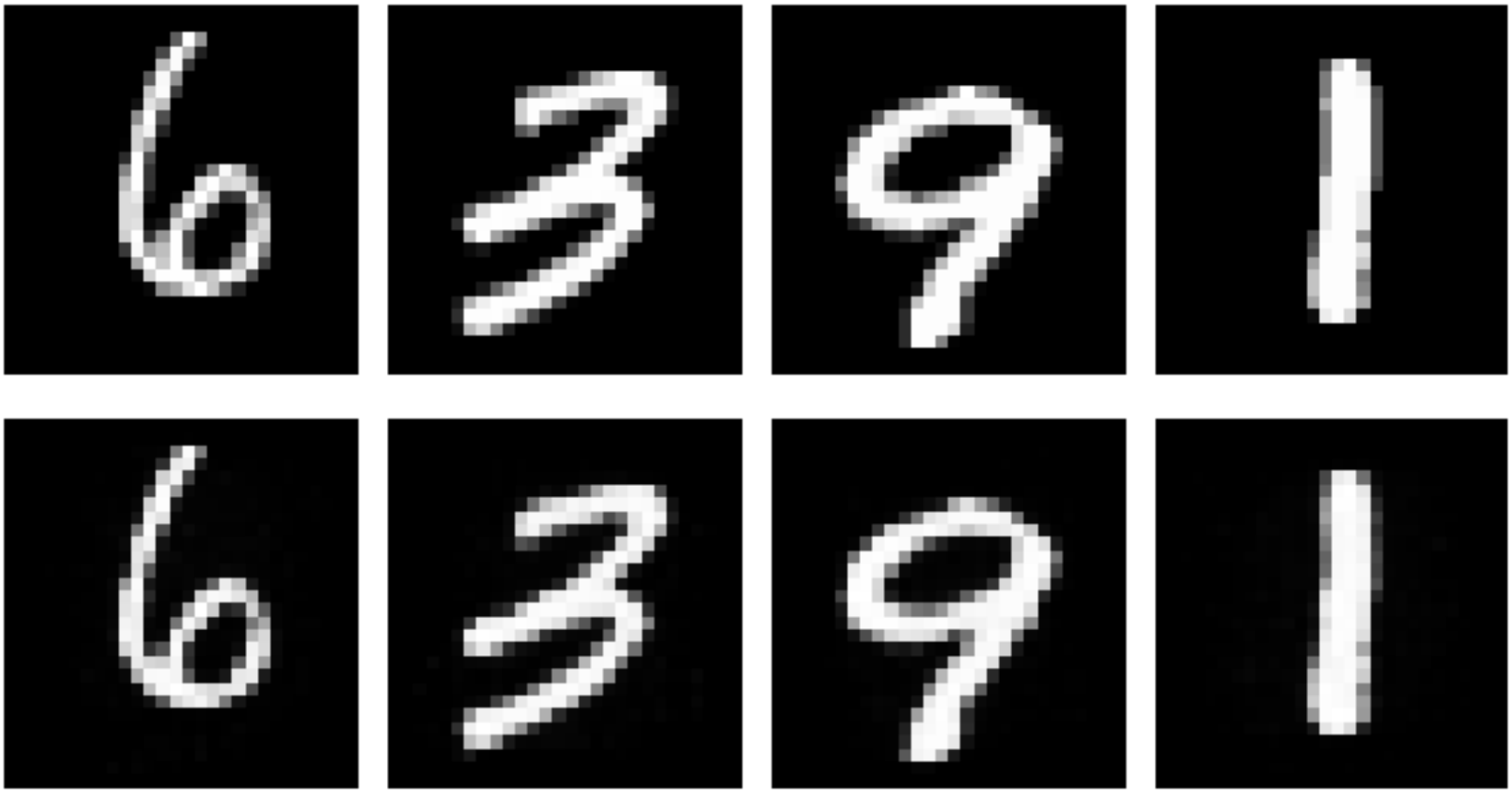


Figure: The top row shows the original digits, the bottom row the reconstructed ones.

- $dim(z) = 784$ (identity)

EXAMPLE

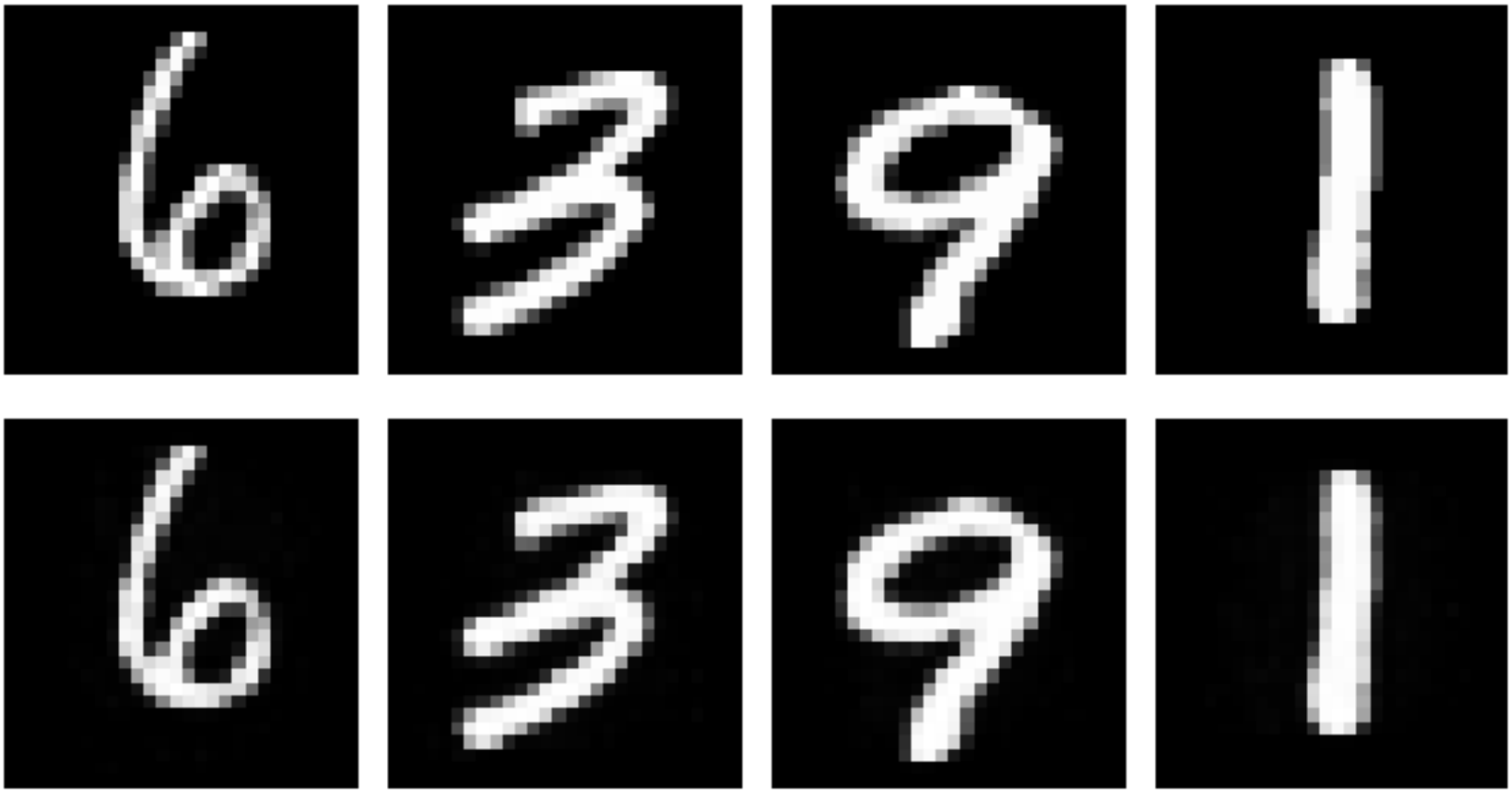


Figure: The top row shows the original digits, the bottom row the reconstructed ones.

- $dim(z) = 256$

EXAMPLE

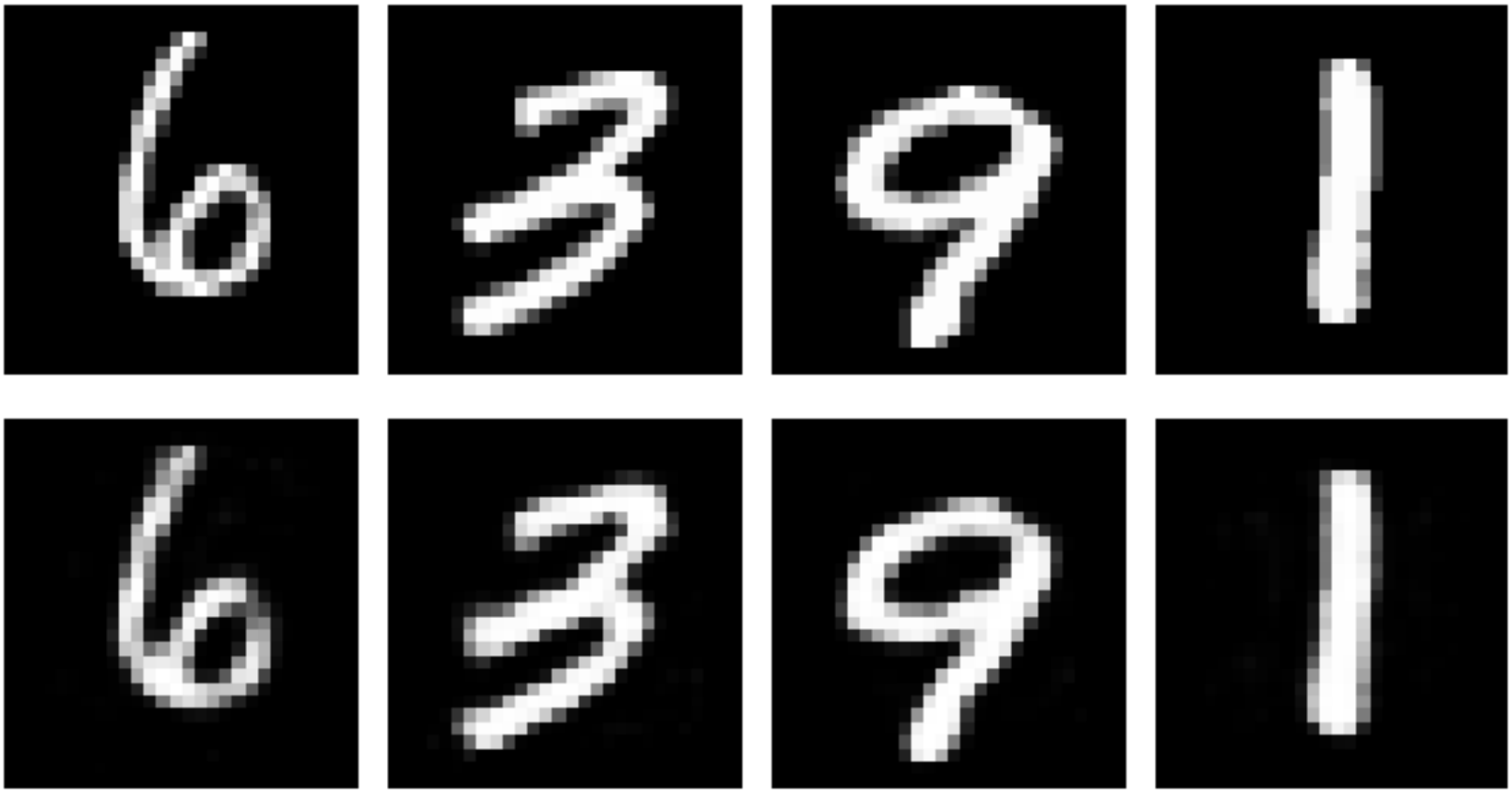


Figure: The top row shows the original digits, the bottom row the reconstructed ones.

- $dim(z) = 64$

EXAMPLE

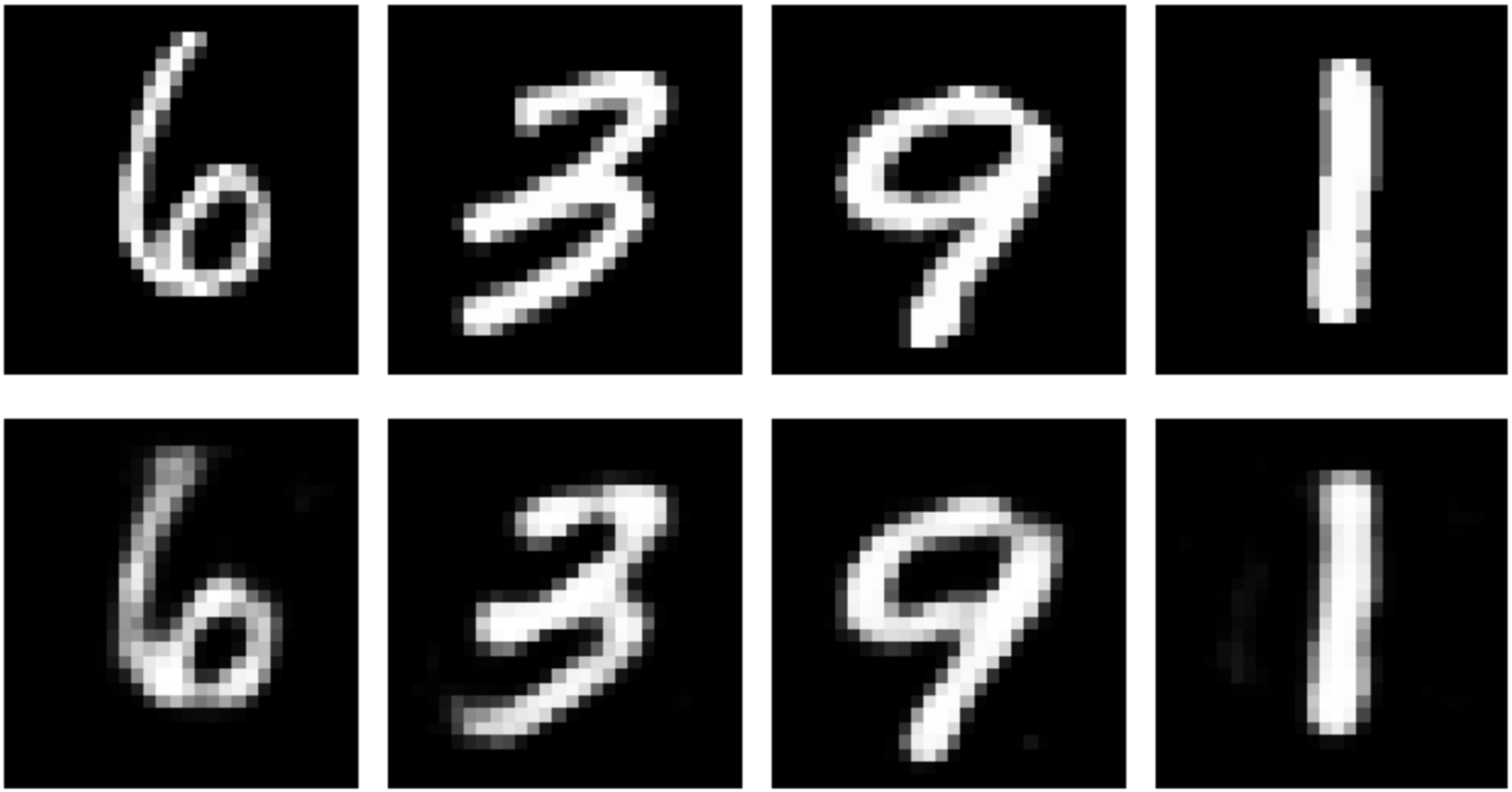


Figure: The top row shows the original digits, the bottom row the reconstructed ones.

- $dim(z) = 32$

EXAMPLE

Figure: The top row shows the original digits, the bottom row the reconstructed ones.

- $dim(z) = 16$

EXAMPLE

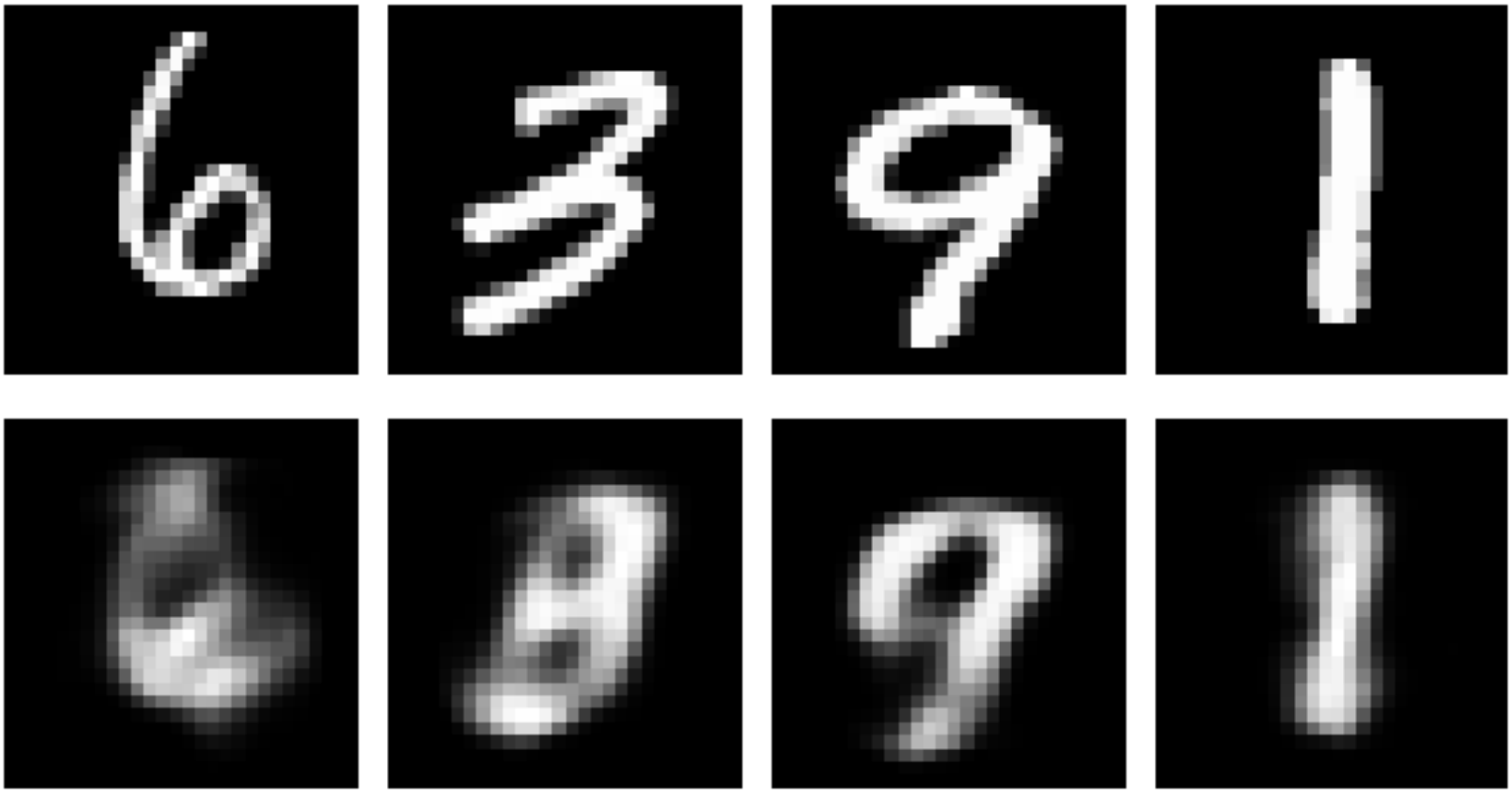


Figure: The top row shows the original digits, the bottom row the reconstructed ones.

- $dim(z) = 8$

EXAMPLE

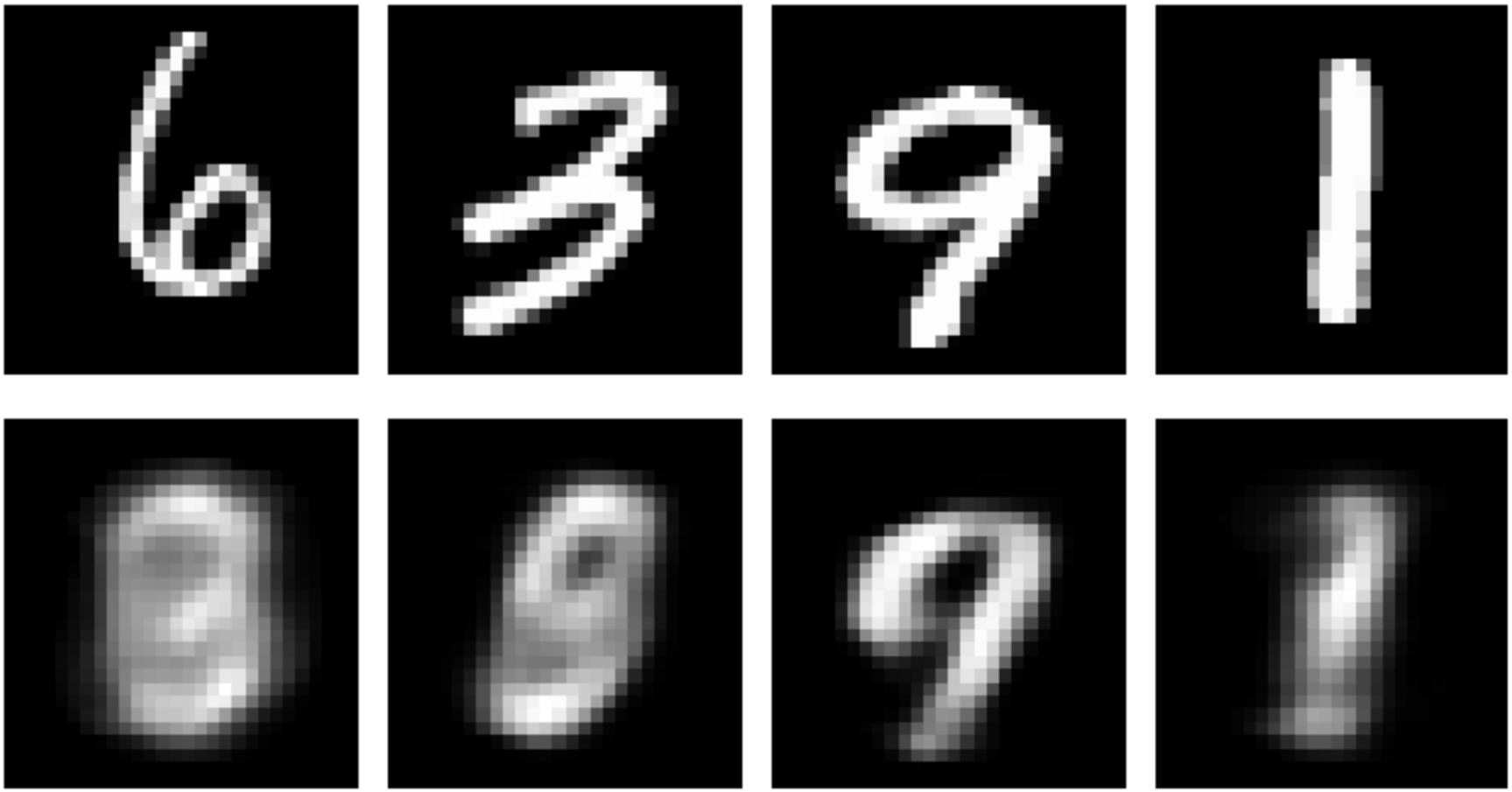


Figure: The top row shows the original digits, the bottom row the reconstructed ones.

- $dim(z) = 4$

EXAMPLE

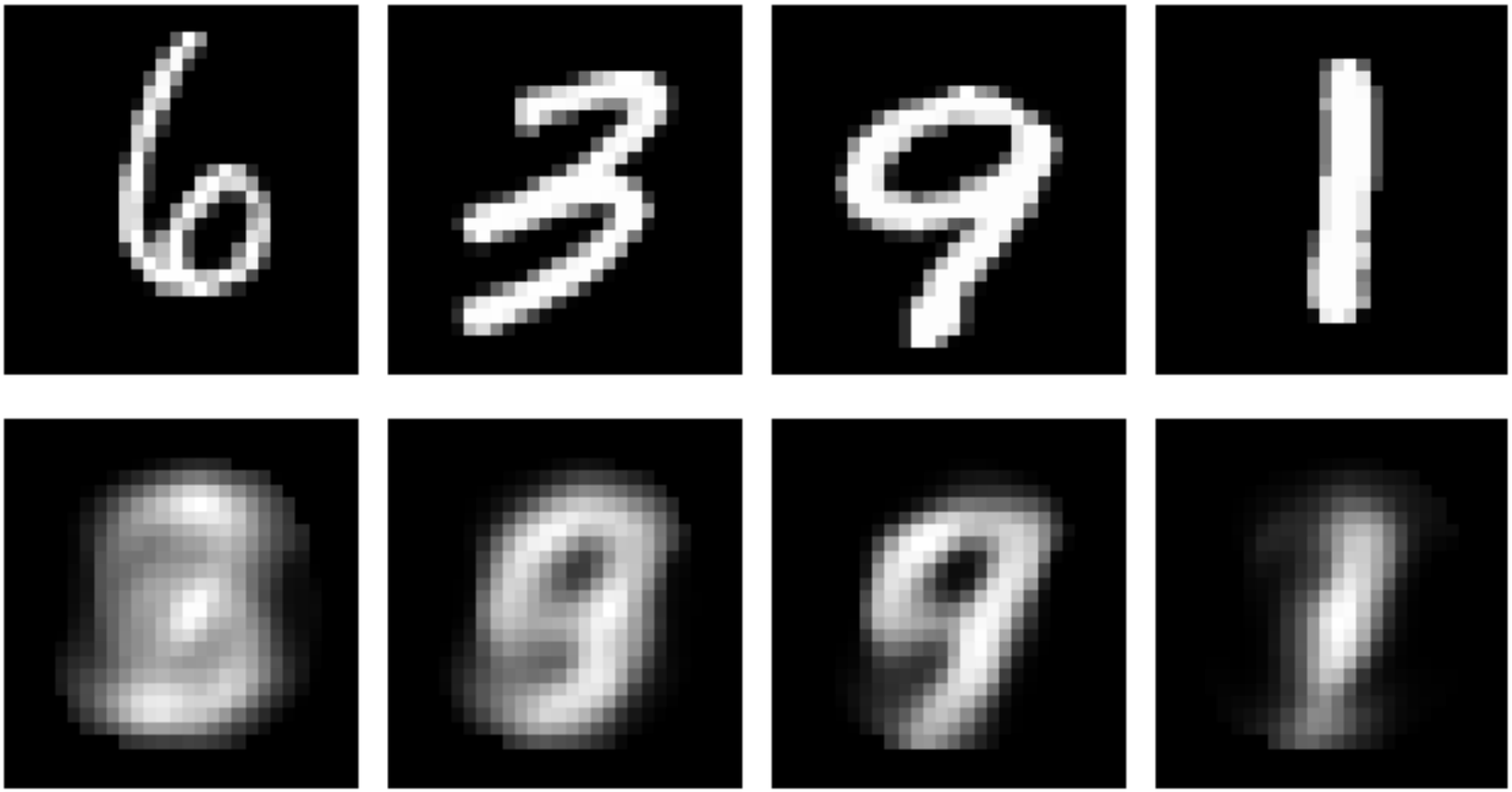


Figure: The top row shows the original digits, the bottom row the reconstructed ones.

- $dim(z) = 2$

EXAMPLE

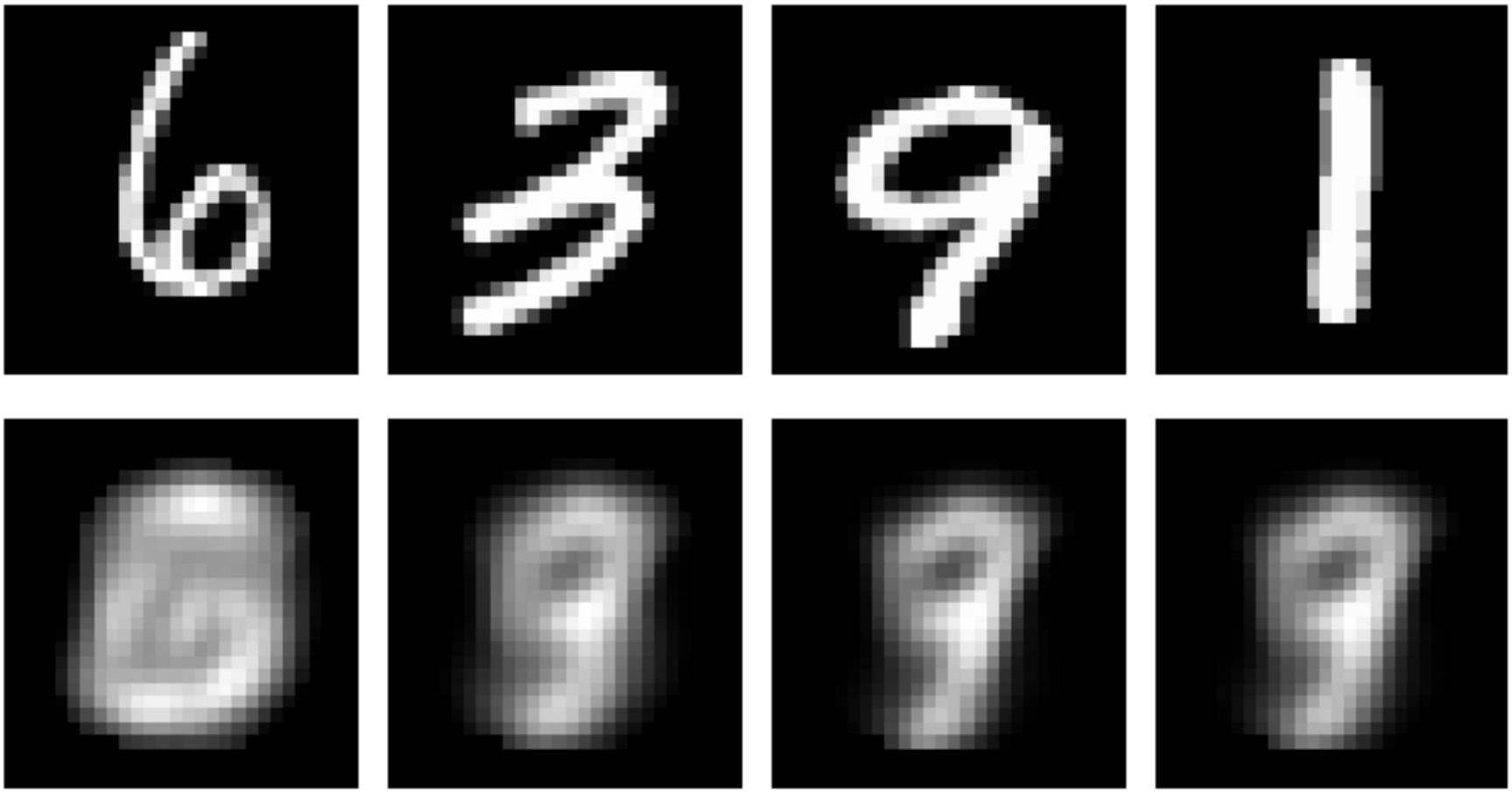


Figure: The top row shows the original digits, the bottom row the reconstructed ones.

- $dim(z) = 1$

UNDERCOMPLETE AUTOENCODERS

- If our decoder is linear and the loss function the mean squared error, the undercomplete autoencoder learns to span the same subspace as principal component analysis.
- Autoencoders with nonlinear encoder and nonlinear decoder functions can learn more powerful gerneralizations of PCA.
- Overcomplete autoencoders without **regularization** make no sense. Even a linear encoder and linear decoder will just learn to copy the complete input layer and thus, extract no meaningful features.

  $\Rightarrow$ we will introduce variants of **regularized** autoencoders, including **sparse** and **denoising** autoencoders.

DENOISING AUTOENCODER (DAE)

- A denoising autoencoder minimizes

$$L(x, dec(enc(\tilde{x})))$$

  where $\tilde{x}$ is a copy of x, which has been corrupted by noise.
- Denoising autoencoders must therefore undo this corruption rather than simply copying their input. Training forces f and g to implicitly learn the structure of $p_{data}(x)$. Thus, the denoising autoencoder is a stochastic version of the autoencoder!
- We introduce a corruption process $C(\tilde{x}|x)$ to represent the conditional distribution of corrupted samples $\tilde{x}$, given a data sample x.
- The autoencoder then learns the reconstruction distribution $p_{reconstruct}(x|\tilde{x})$ estimated from training pairs $(x, \tilde{x})$.

DENOISING AUTOENCODER (DAE)

The general structure of a denoising autoencoder:

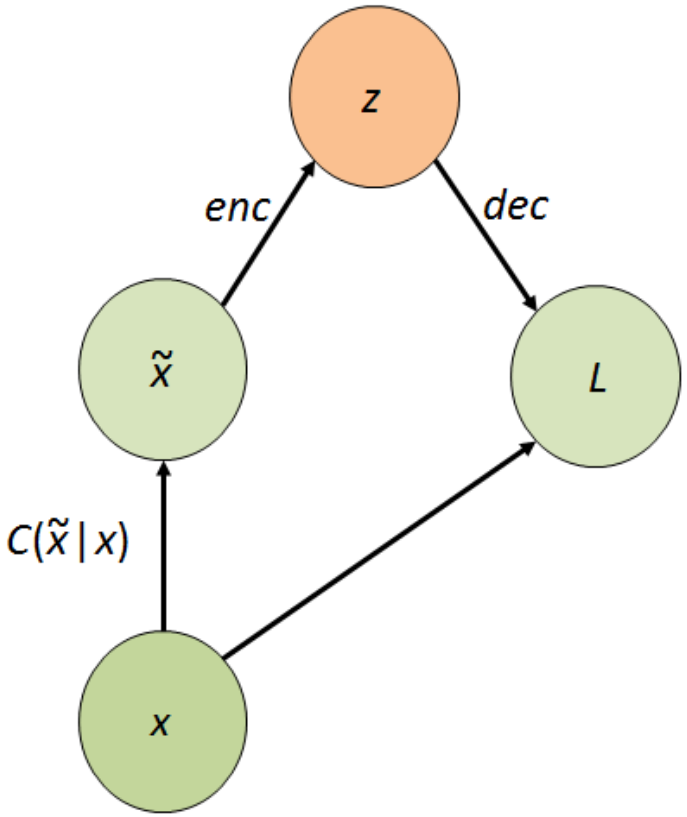


Figure: Denoising autoencoder: “idea of making the learned representation robust to partial corruption of the input pattern”

DENOISING AUTOENCODER (DAE)

Algorithm 1 Training denoising autoencoders

1: Sample a training example x from the training data.
2: Sample a corrupted version $\tilde{x}$ from $C(\tilde{x}|x)$
3: Use $(x, \tilde{x})$ as a training example for estimating the autoencoder reconstruction $p_{reconstruct}(x|\tilde{x}) = p_{decoder}(x|z)$, where
- h is the output of the encoder $enc(\tilde{x})$ and
- $p_{decoder}$ defined by a decoder $dec(z)$

- So the denoising autoencoder still tries to preserve the information about the input (encode it), but also to undo the effect of a corruption process!
- All we have to do to transform a autoencoder to a denoising autoencoder is to add a stochastic corruption process on the input.

DENOISING AUTOENCODER (DAE)

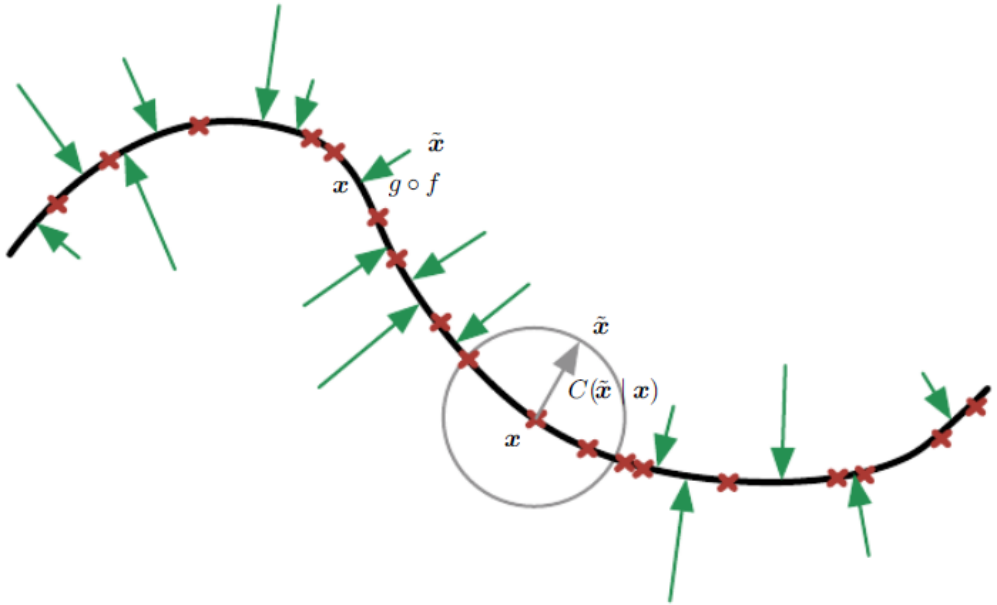


Figure: Denoising autoencoders “manifold perspective” (Ian Goodfellow et al. (2016))

- A denoising autoencoder is trained to map a corrupted data point $\tilde{x}$ back to the original data point x.
- The corruption process $C(\tilde{x}|x)$ is displayed by the gray circle of equiprobable corruptions

DENOISING AUTOENCODER (DAE)

- When the denoising autoencoder is trained to minimize the MSE $||dec(enc(\tilde{x}) - x||^2$, the reconstruction $dec(enc(\tilde{x}))$ estimates

$$\mathbb{E}_{x,\tilde{x}\sim p_{data}(x)C(\tilde{x}|x)}[x|\tilde{x}]$$

- The vector $dec(enc(\tilde{x})) - \tilde{x}$ points approximately towards the nearest point in the manifold, since $dec(enc(\tilde{x}))$ estimates the center of mass of clean points x which could have given rise to $\tilde{x}$.
- Thus, the autoencoder learns a vector field $dec(enc(x)) - x$ indicated by the green arrows.

EXAMPLE

- We will now corrupt the mnist data with gaussian noise and then try to denoise it as good as possible.

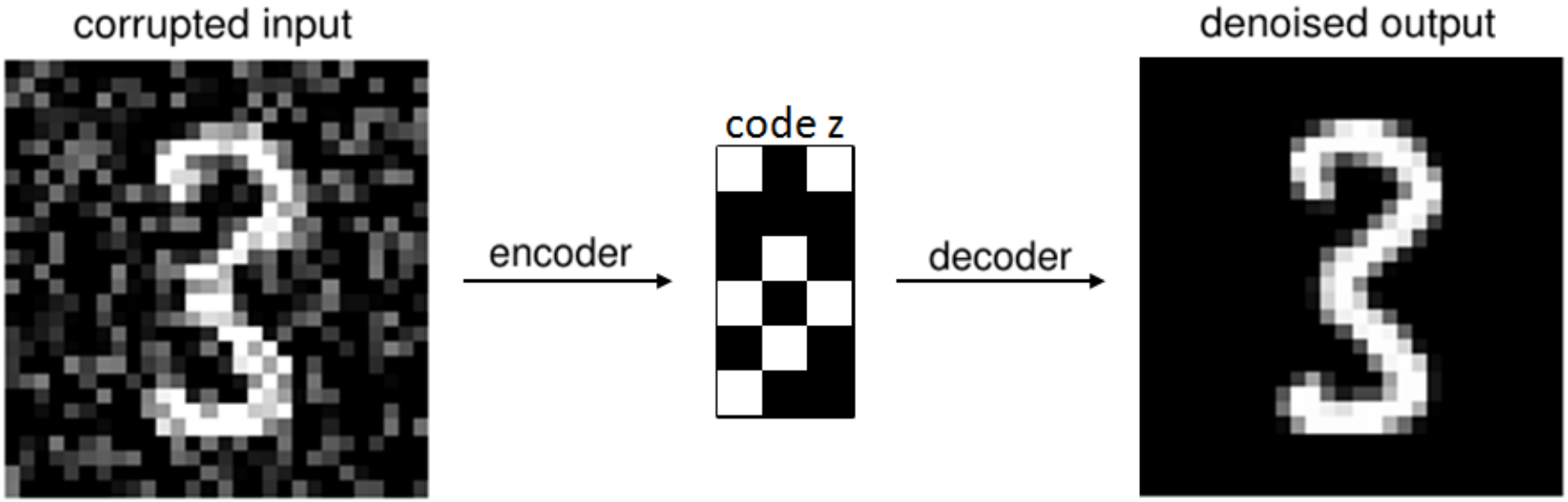


Figure: Flow chart of our our autoencoder: denoise the corrupted input

EXAMPLE

- To corrupt the input, we randomly add or subtract values from a uniform distribution to each of the image entries.

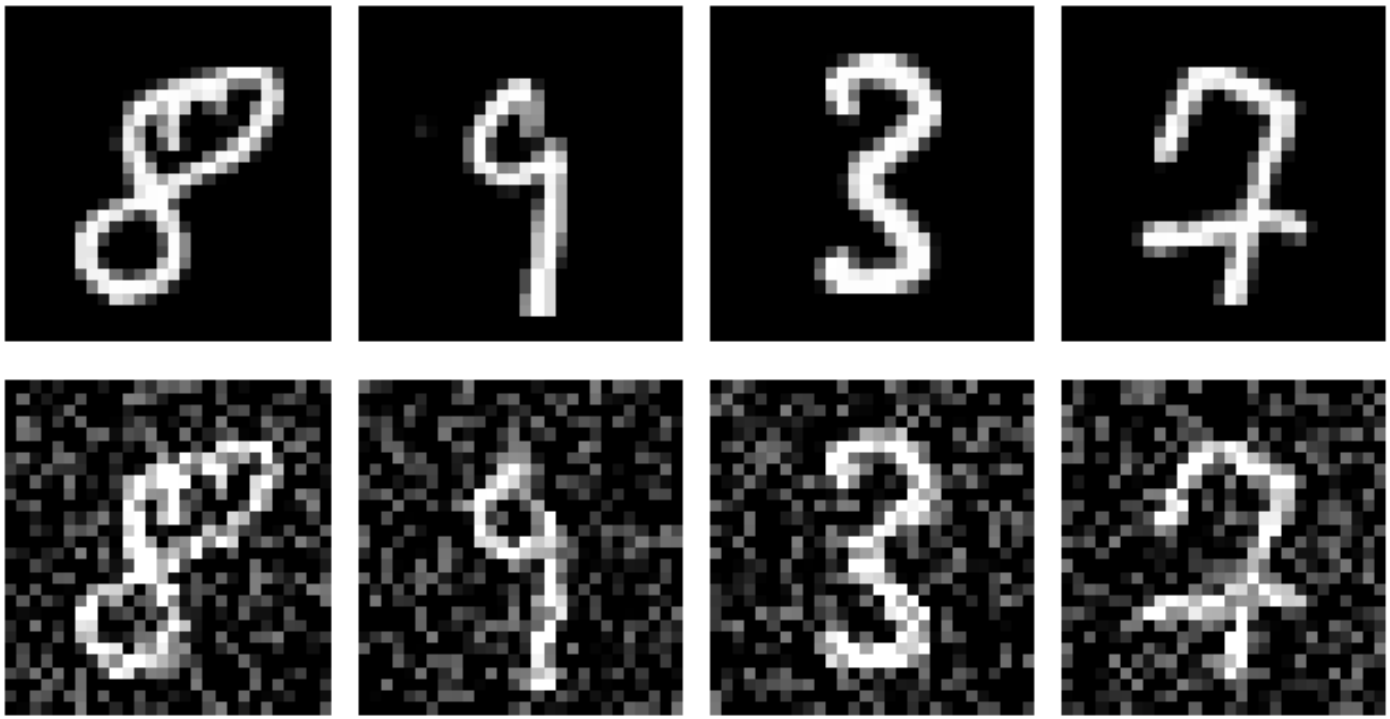


Figure: Top row: original data, bottom row: corrupted mnist data

EXAMPLE

```
z = c(1568, 784, 256, 64, 32, 16, 8)

input = mx.symbol.Variable("data") # mnist with 28x28 = 784
encoder = mx.symbol.FullyConnected(input, num_hidden = z[i])
decoder = mx.symbol.FullyConnected(encoder, num_hidden = 784)
activation = mx.symbol.Activation(decoder, "sigmoid")
output = mx.symbol.LinearRegressionOutput(activation)

model = mx.model.FeedForward.create(output,
  X = train.x, y = train.x,
  num.round = 50,
  array.batch.size = 32,
  optimizer = "adam",
  initializer = mx.init.uniform(0.01),
  eval.metric = mx.metric.mse
)
```

EXAMPLE

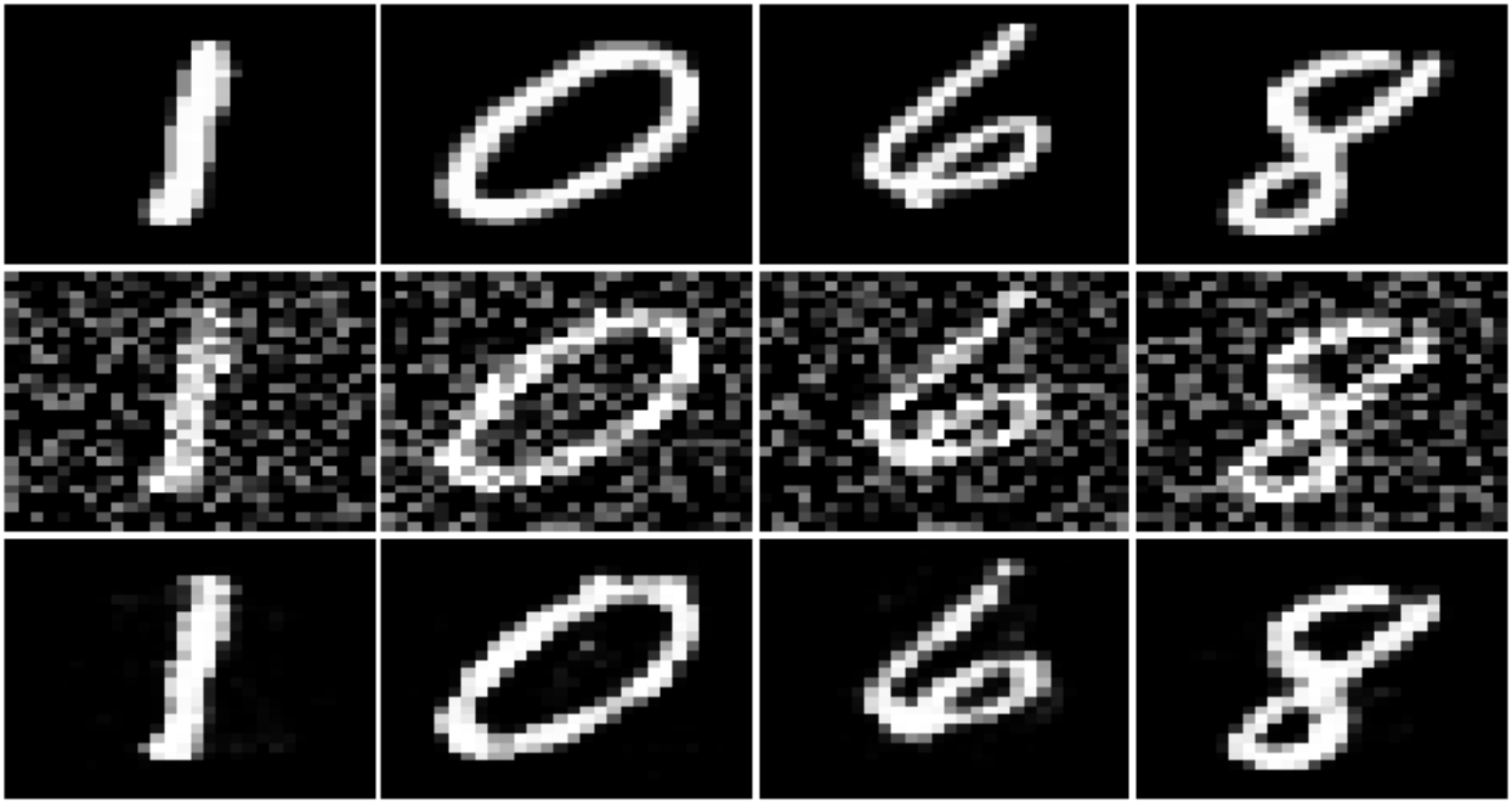


Figure: The top row shows the original digits, the intermediate one the corrupted and the bottom row the denoised/recontructed digits (prediction).

- $dim(z) = 1568$ (overcomplete)

EXAMPLE

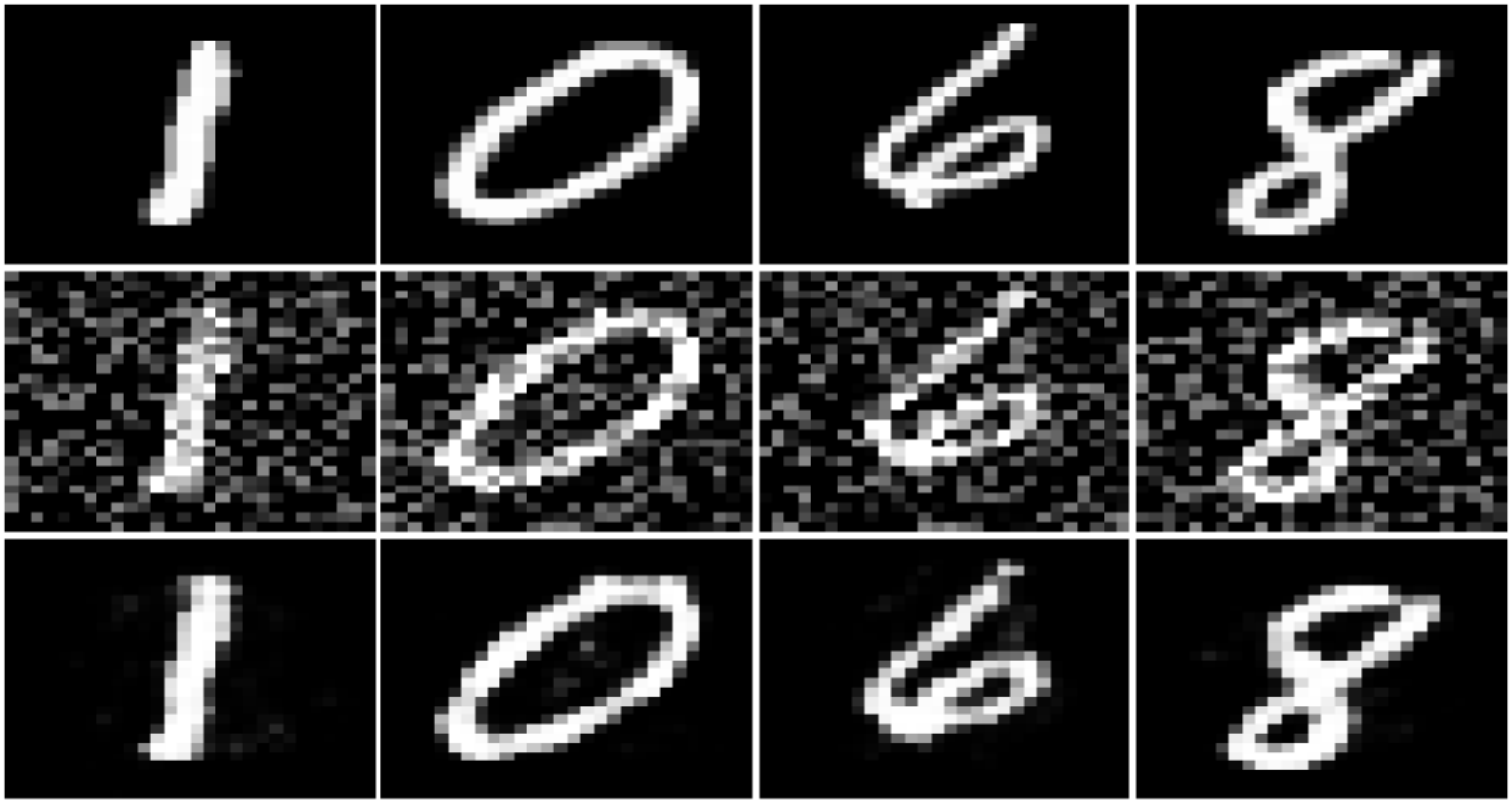


Figure: The top row shows the original digits, the intermediate one the corrupted and the bottom row the denoised/recontructed digits (prediction).

- $dim(z) = 784$ (identity)

EXAMPLE

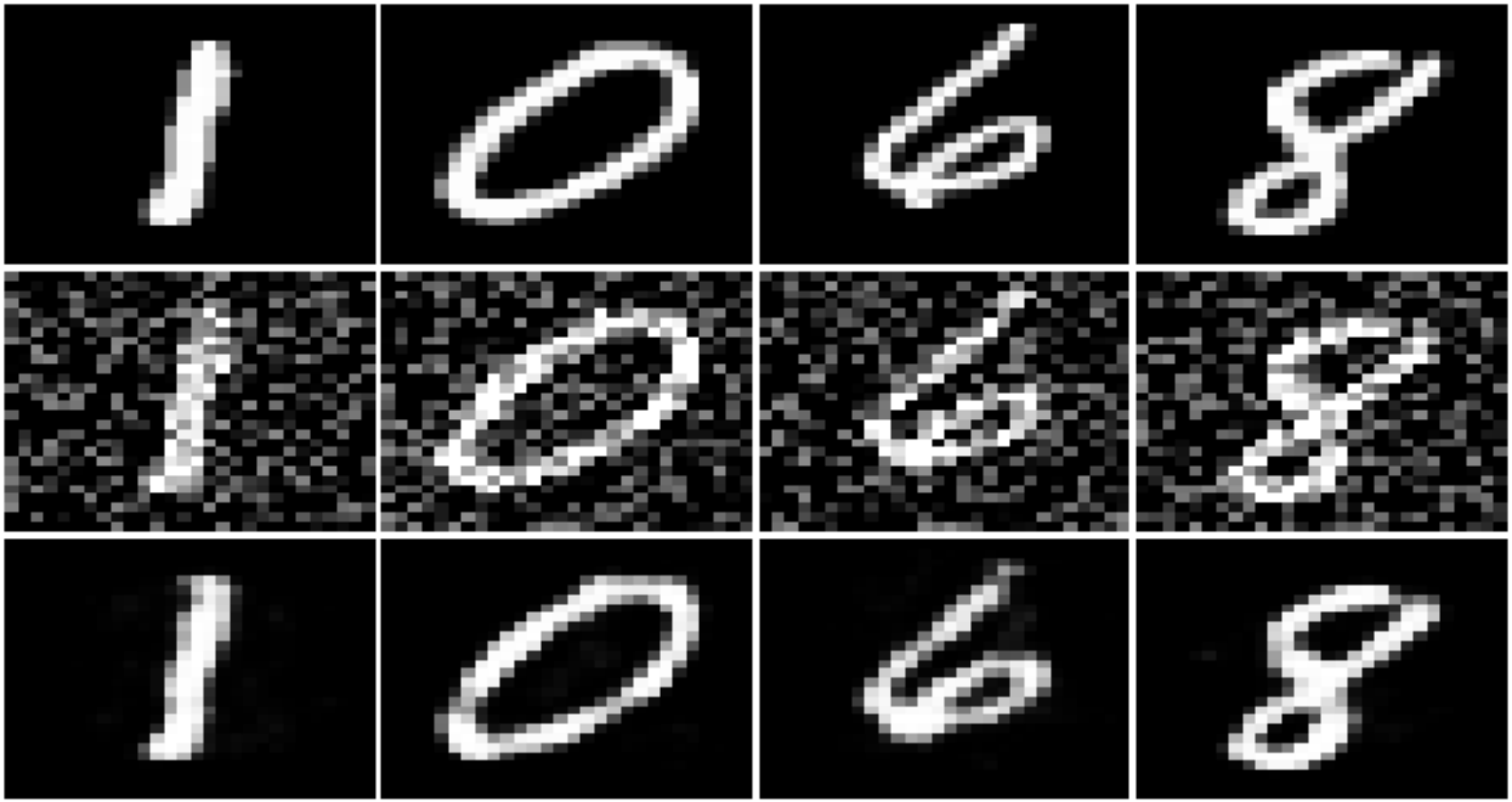


Figure: The top row shows the original digits, the intermediate one the corrupted and the bottom row the denoised/recontructed digits (prediction).

- $dim(z) = 256$

EXAMPLE

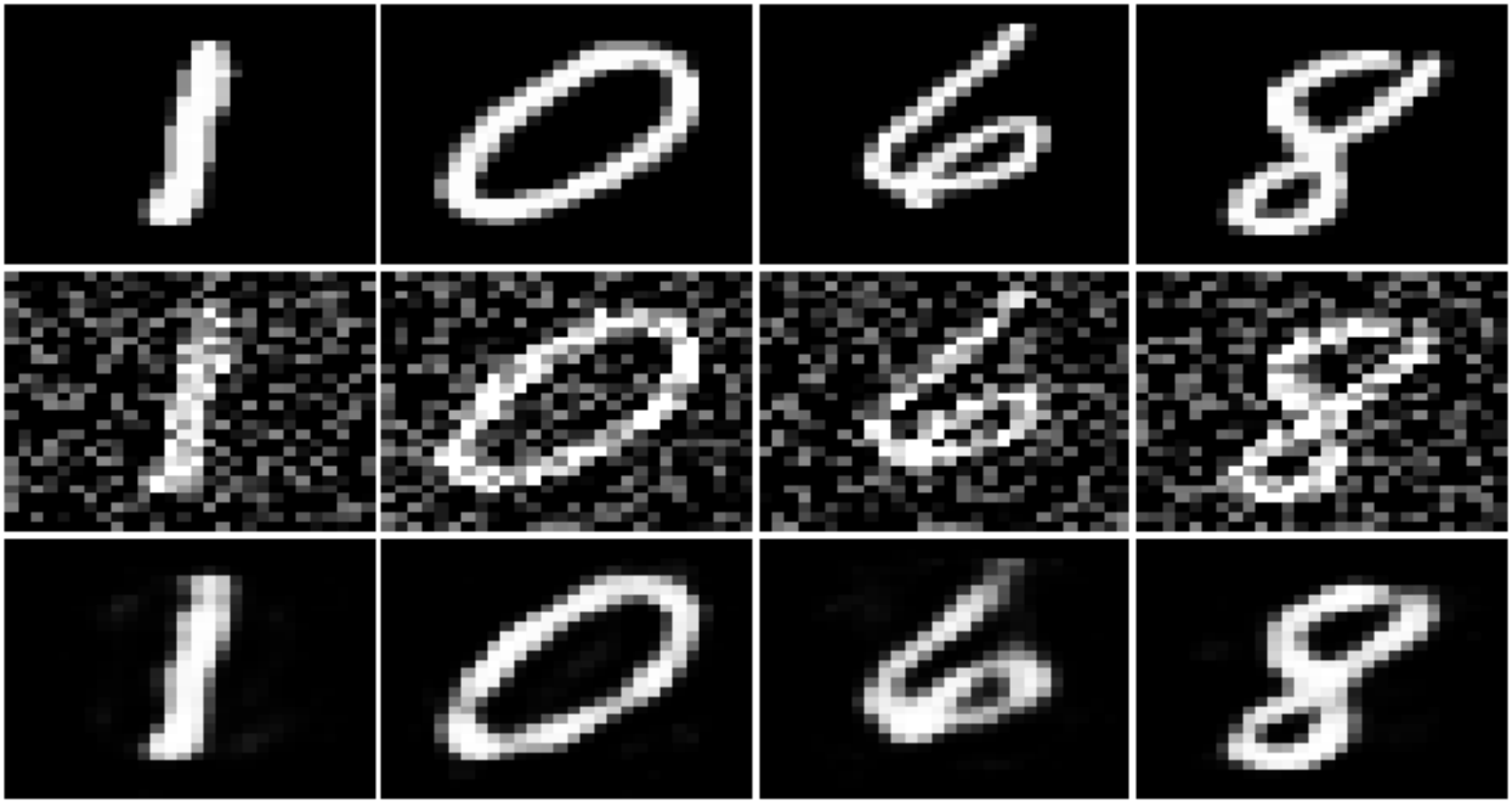


Figure: The top row shows the original digits, the intermediate one the corrupted and the bottom row the denoised/recontructed digits (prediction).

- $dim(z) = 64$

EXAMPLE

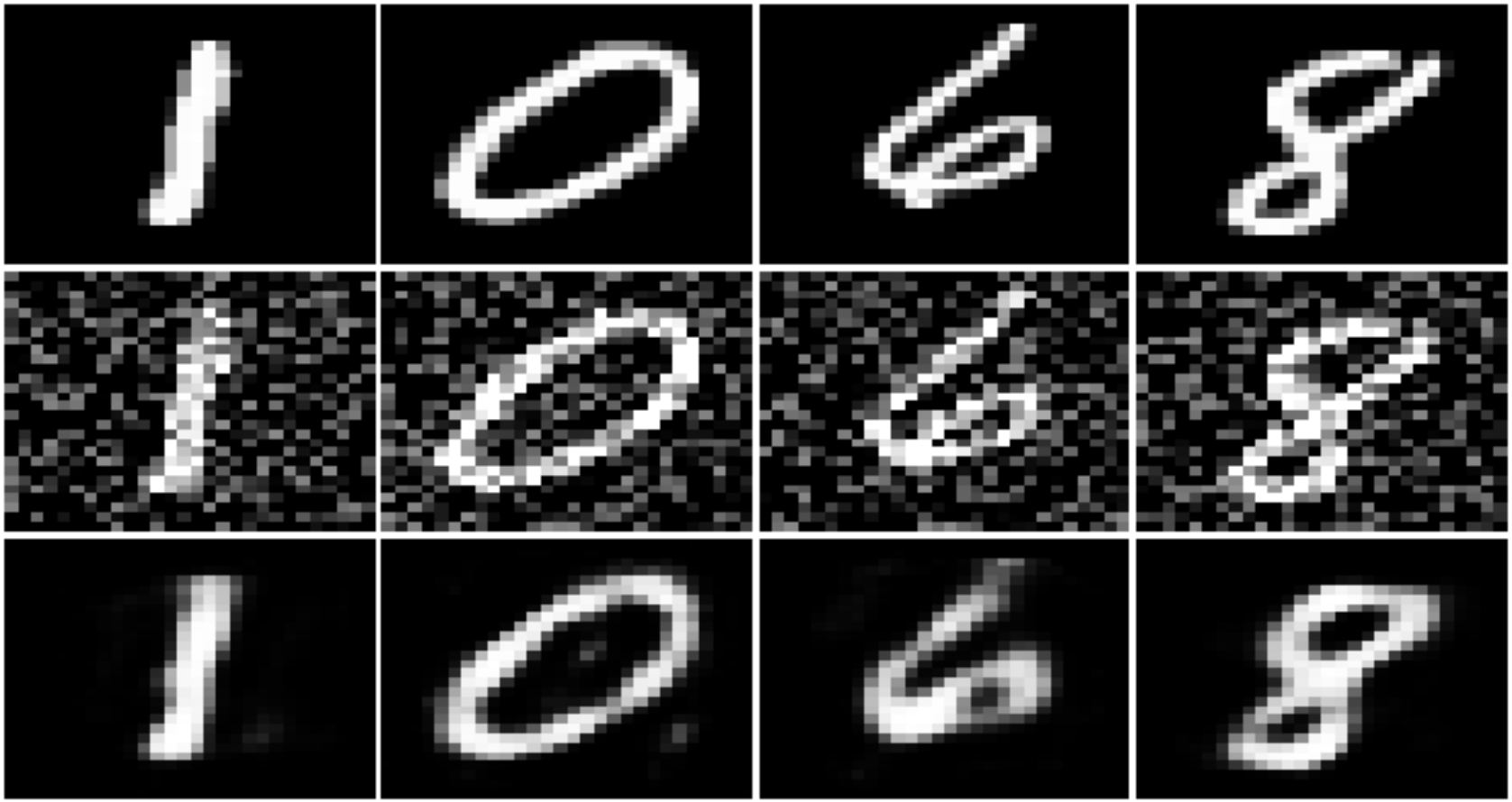


Figure: The top row shows the original digits, the intermediate one the corrupted and the bottom row the denoised/recontructed digits (prediction).

- $dim(z) = 32$

EXAMPLE

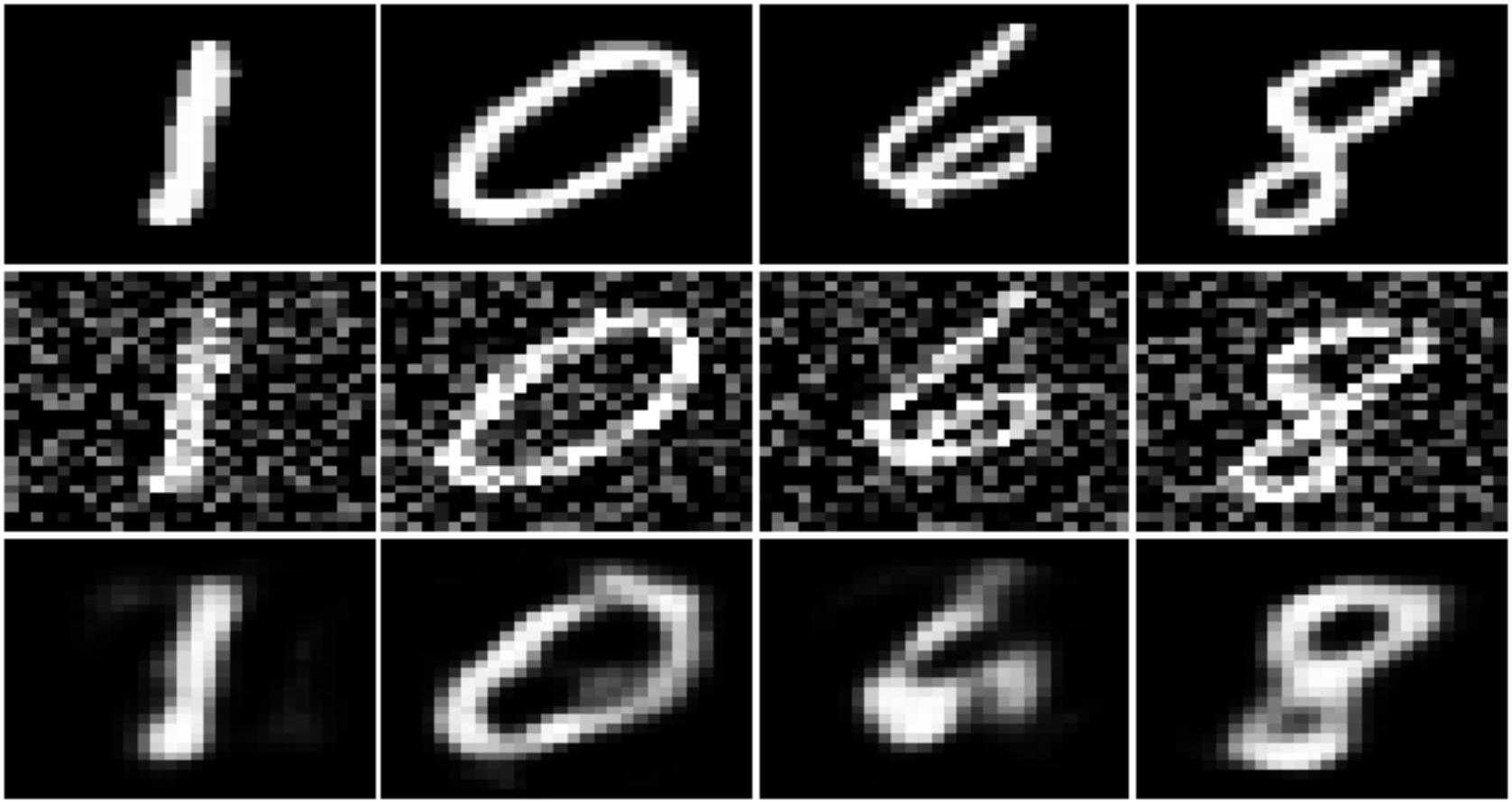


Figure: The top row shows the original digits, the intermediate one the corrupted and the bottom row the denoised/recontructed digits (prediction).

- $dim(z) = 16$

EXAMPLE

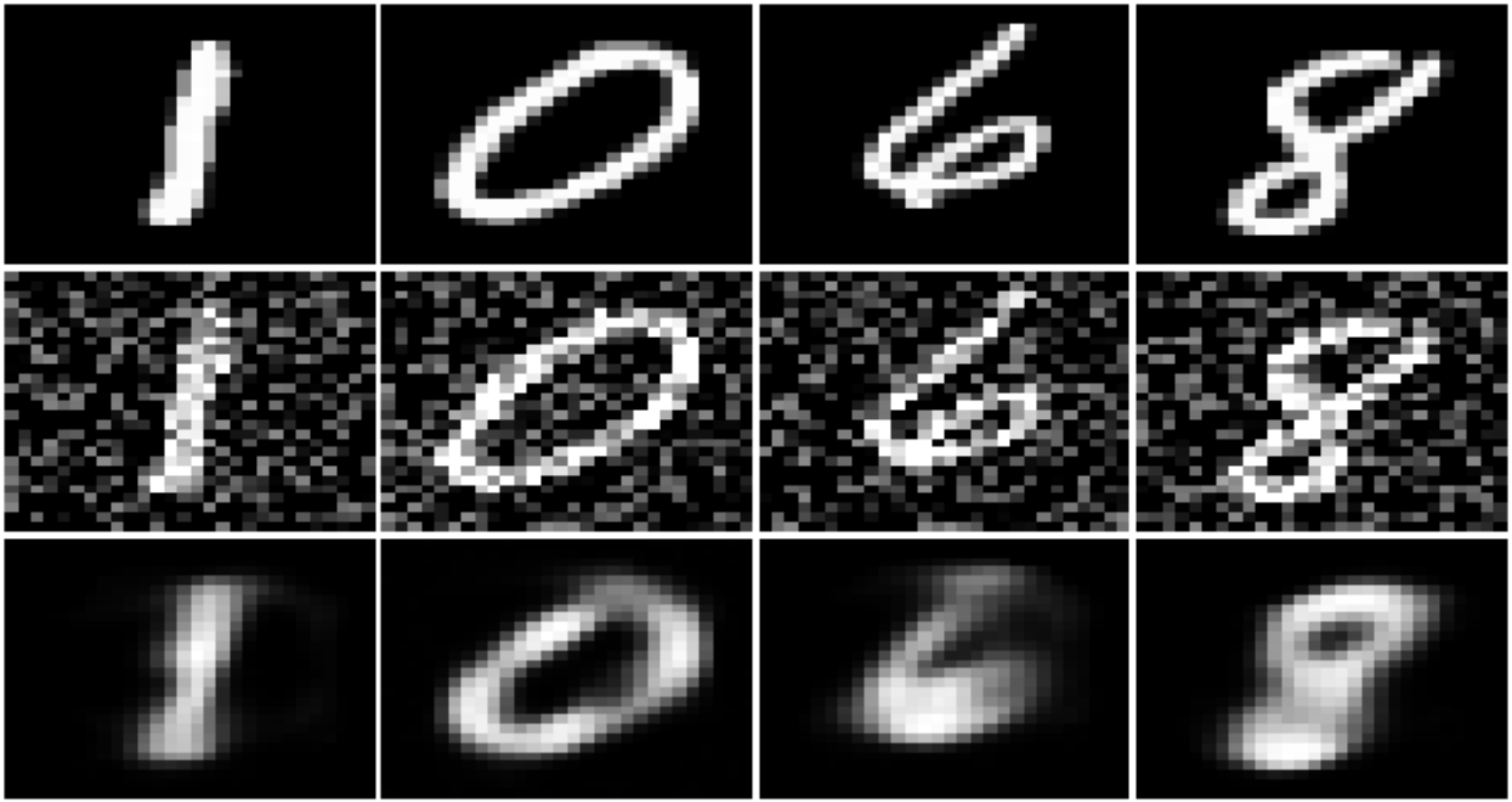


Figure: The top row shows the original digits, the intermediate one the corrupted and the bottom row the denoised/recontructed digits (prediction).

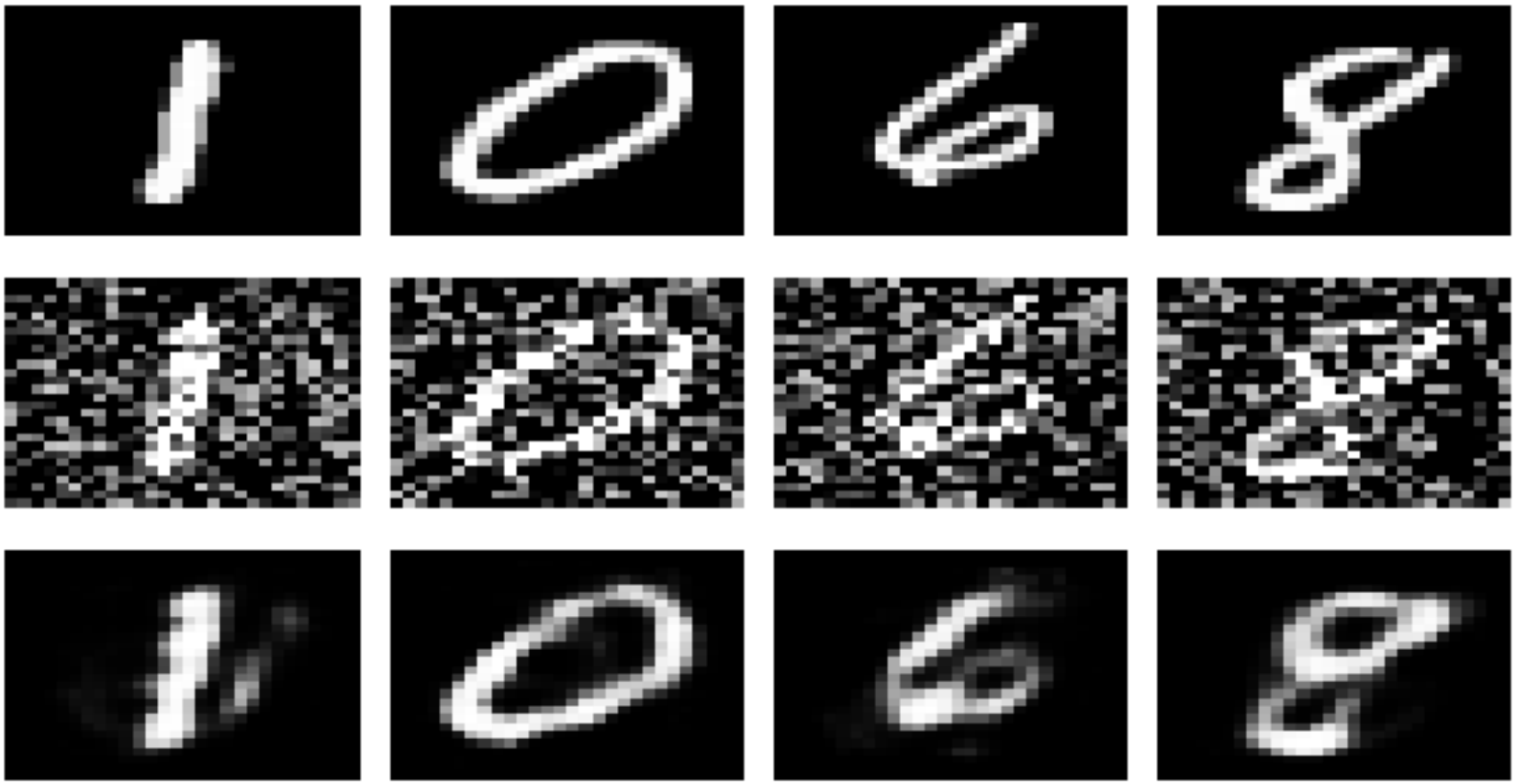
- $dim(z) = 8$

EXAMPLE

- Let us increase the amount of noise and see how the autoencoder with $dim(z) = 64$ deals with it (for science!)

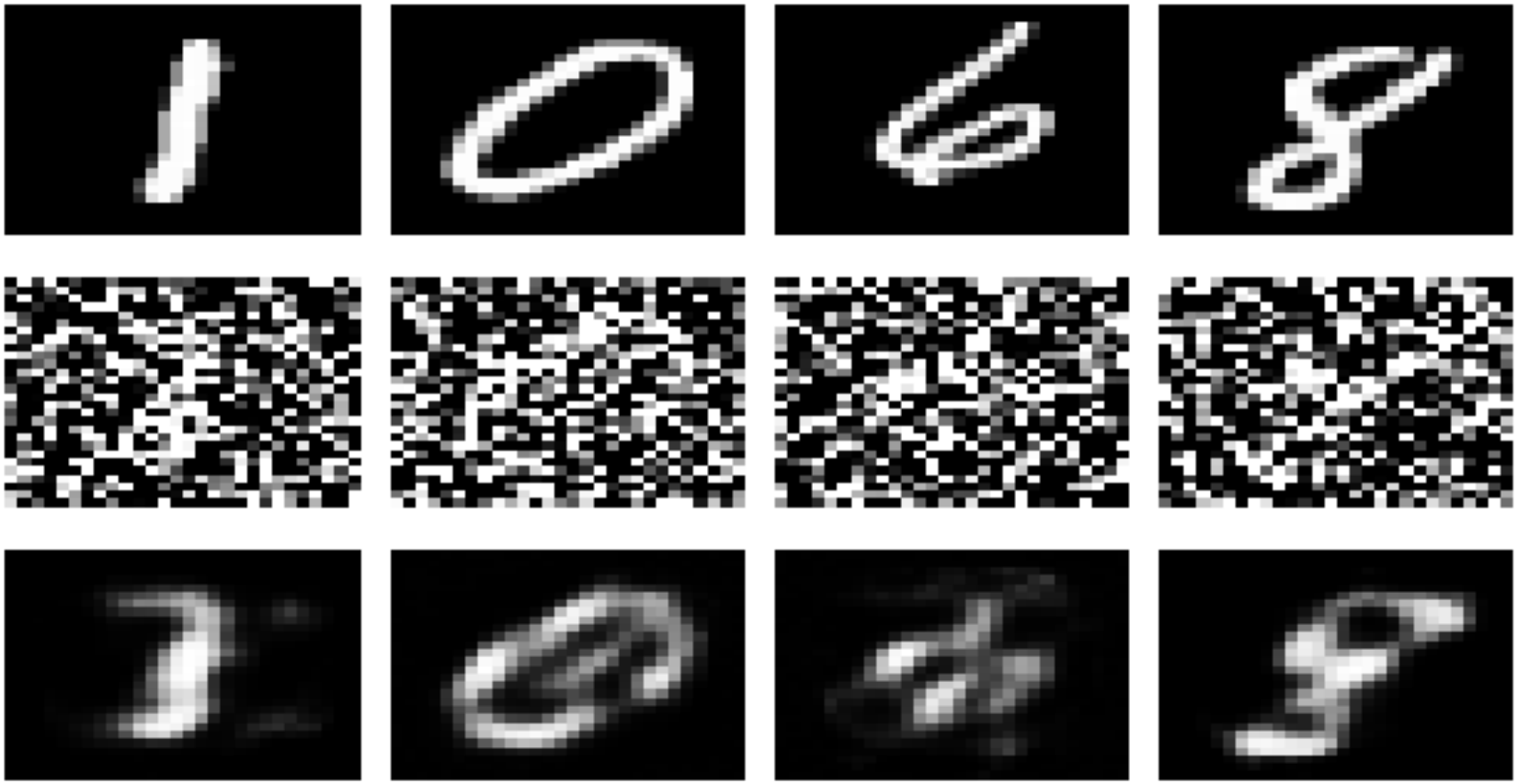
EXAMPLE

- Alot of noise



EXAMPLE

- **Alot** of noise



CONVOLUTIONAL AUTOENCODER (CONVAE)

- In the last example, we have seen that autoencoder inputs are images. So, it makes sense to ask whether a convolutional architecture can work better than the autoencoder architectures discussed previously.
- The encoder consists of convolutional layers, the decoder on the other hand of transpose convolution layers (sometimes mistakenly called deconvolution layers) or simple upsampling operations.
  - Instead of convolving a filter mask with an image, to get a set of activations as in a convolutional neural net, we are trying to infer the activations that when convolved with the filter mask, would yield the image.
- The original aim was to learn a hierarchy of features in an unsupervised manner. However, now its more commonly being used to invert the downsampling that takes place in a convolutional network and “expand” the image back to its original size.

CONVOLUTIONAL AUTOENCODER (CONVAE)

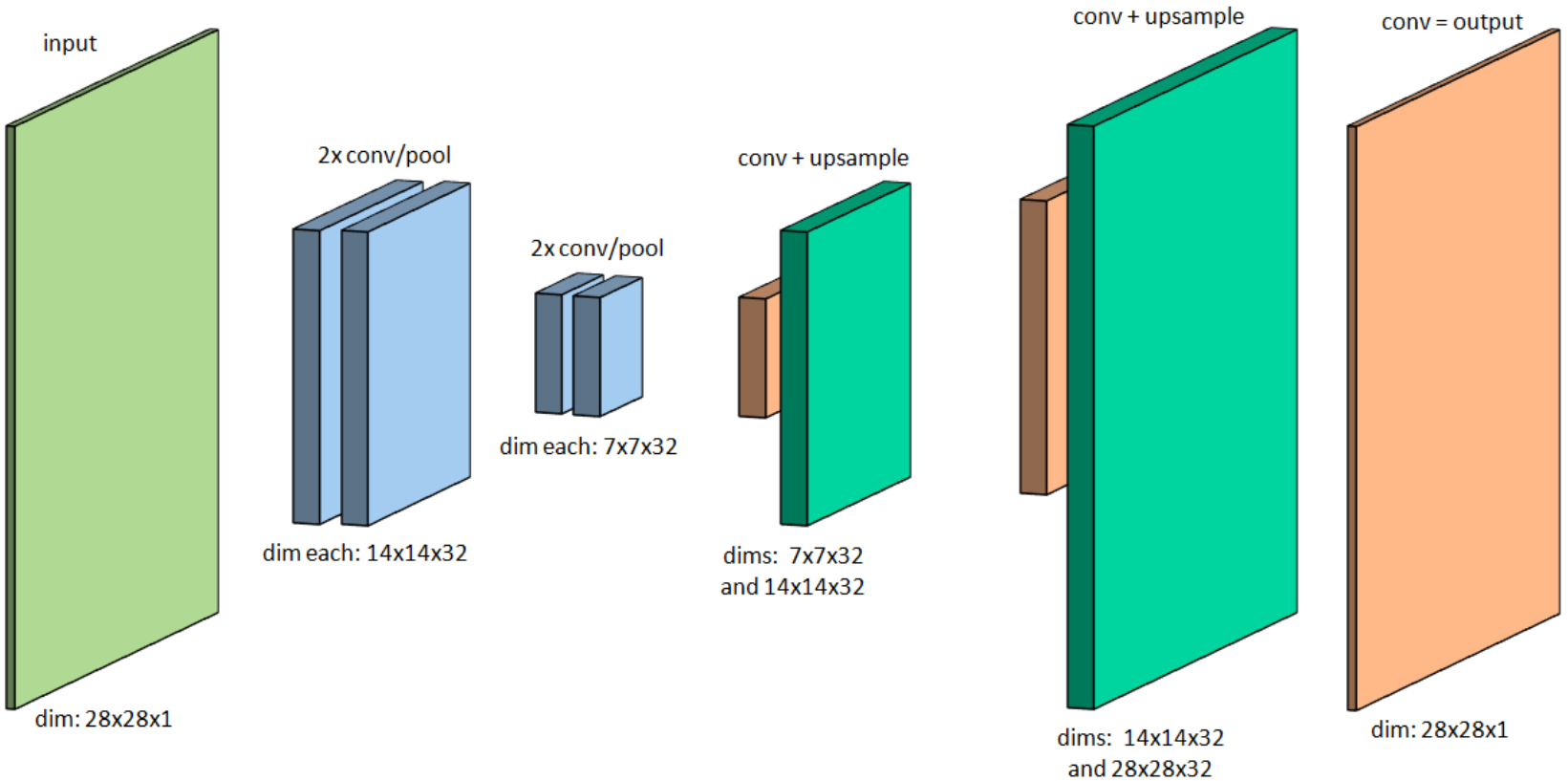


Figure: Potential architecture of an convolutional autoencoder

We now apply this architecture to denoise mnist.

CONVOLUTIONAL AUTOENCODER (CONVAE)

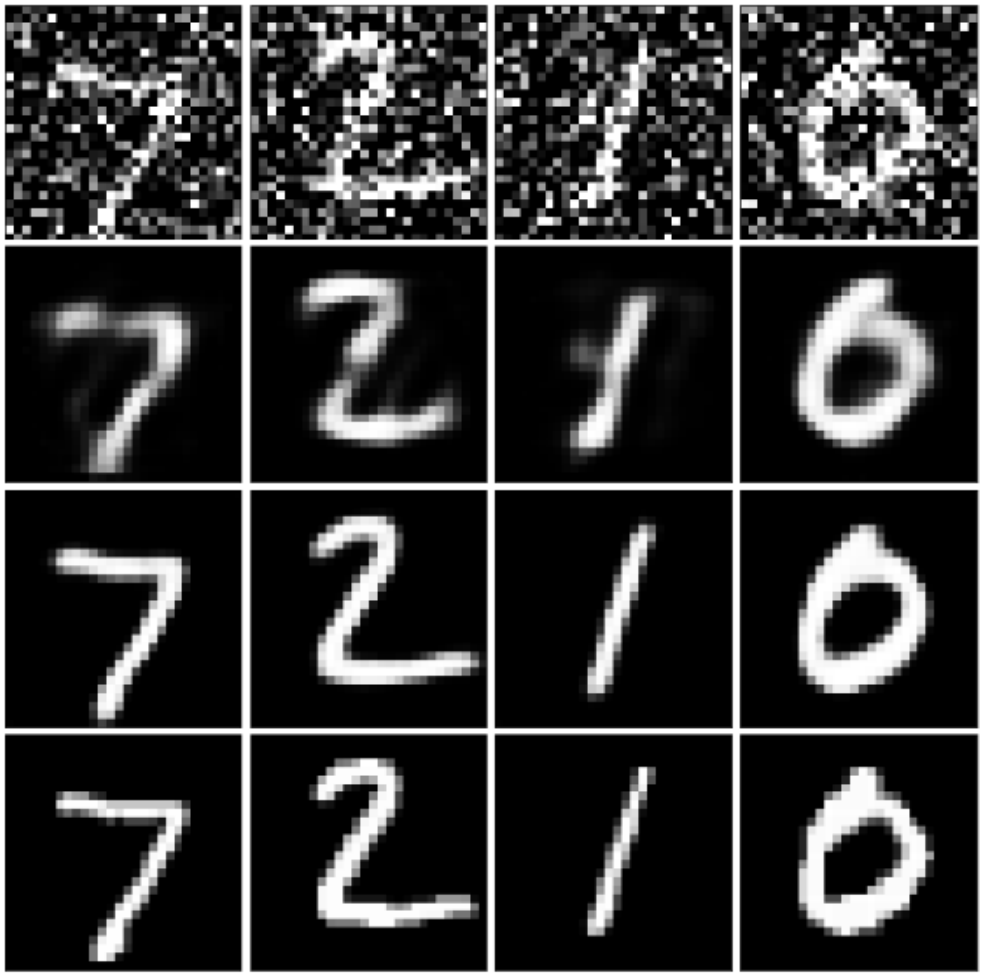


Figure: Top row: noised data, second row: auto encoder with $dim(z) = 32$ (roughly 50k params), third row: convolutional autoencoder(roughly 25k params), fourth row: ground truth

CONVOLUTIONAL AUTOENCODER (CONVAE)

Convolutional autoencoders may also be used for image segmentation:

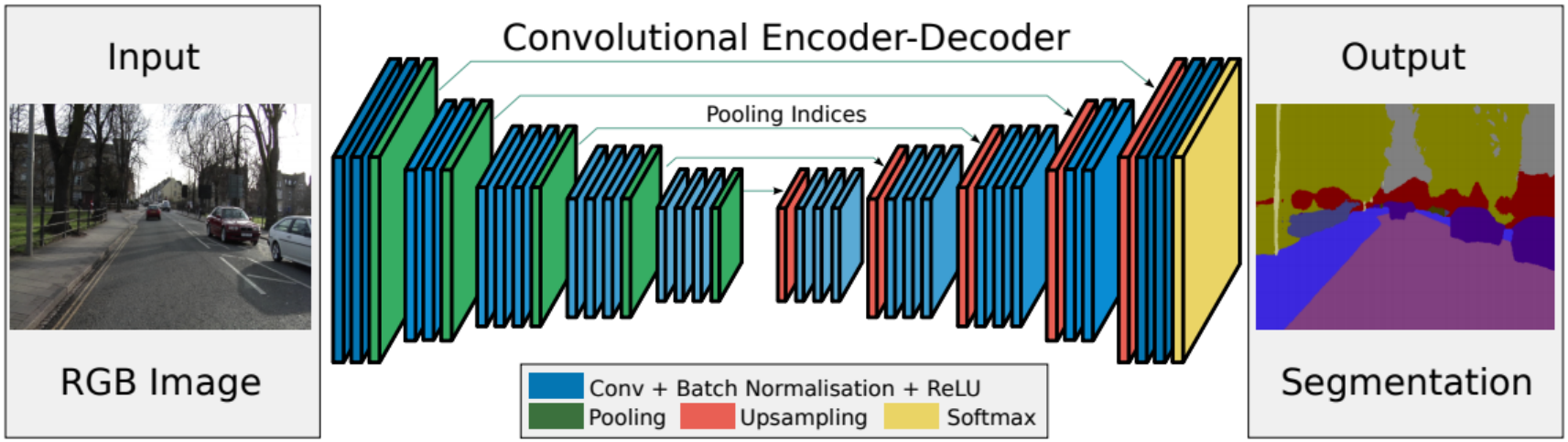


Figure: SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation (Vijay Badrinarayanan et al. (2016))

REALWORLD APPLICATION

Medical image denoising using convolutional denoising autoencoders

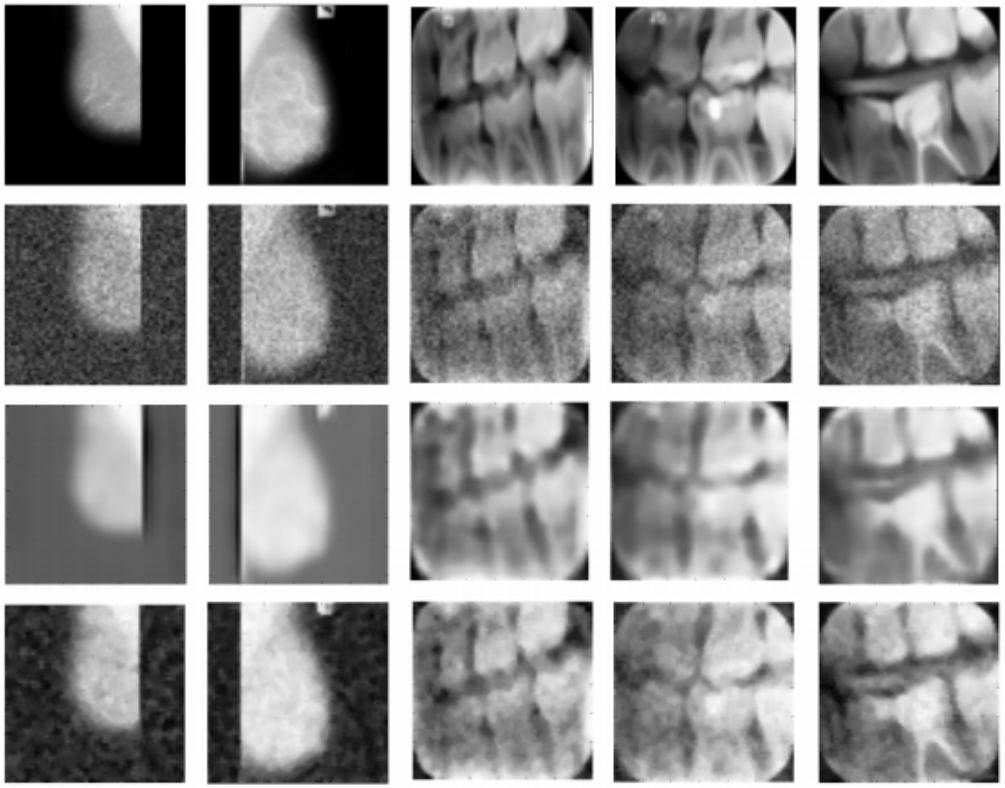


Figure: Top row shows real image, second row noisy version, third row results of a (convolutional) denoising autoencoder and fourth row results of a median filter (Lovedeep Gondara (2016))

STACKING AUTOENCODERS

- Stacked autoencoder is used to pretrain each layer unsupervised to have better weight initialization for the actual supervised training.
- The better weight initialization usually eliminates the risk of vanishing gradients in feed forward nets.

VARIATIONAL AUTOENCODER (VAE)

- VAEs stand for Variational Autoencoders. By its name, you can probably tell that a VAE is pretty similar to an autoencoder, and it technically is, but with one major twist.
- While an autoencoder just has to reproduce its input, a variational autoencoder has to reproduce its output, **while keeping its hidden neurons to a specific distribution**.
  - That means, the output of the network will have to get used to the hidden neurons output based on a distribution, and so we can generate new images, just by sampling from that distribution, and inputting it into the network's hidden layer.

REFERENCES

Ian Goodfellow, Yoshua Bengio and Aaron Courville (2016)

Deep Learning

http://www.deeplearningbook.org/

Lovedeep Gondara (2016)

Medical image denoising using convolutional denoising autoencoders

https://arxiv.org/abs/1608.04667

Vijay Badrinarayanan, Alex Kendall and Roberto Cipolla (2016)

SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation

https://arxiv.org/abs/1511.00561