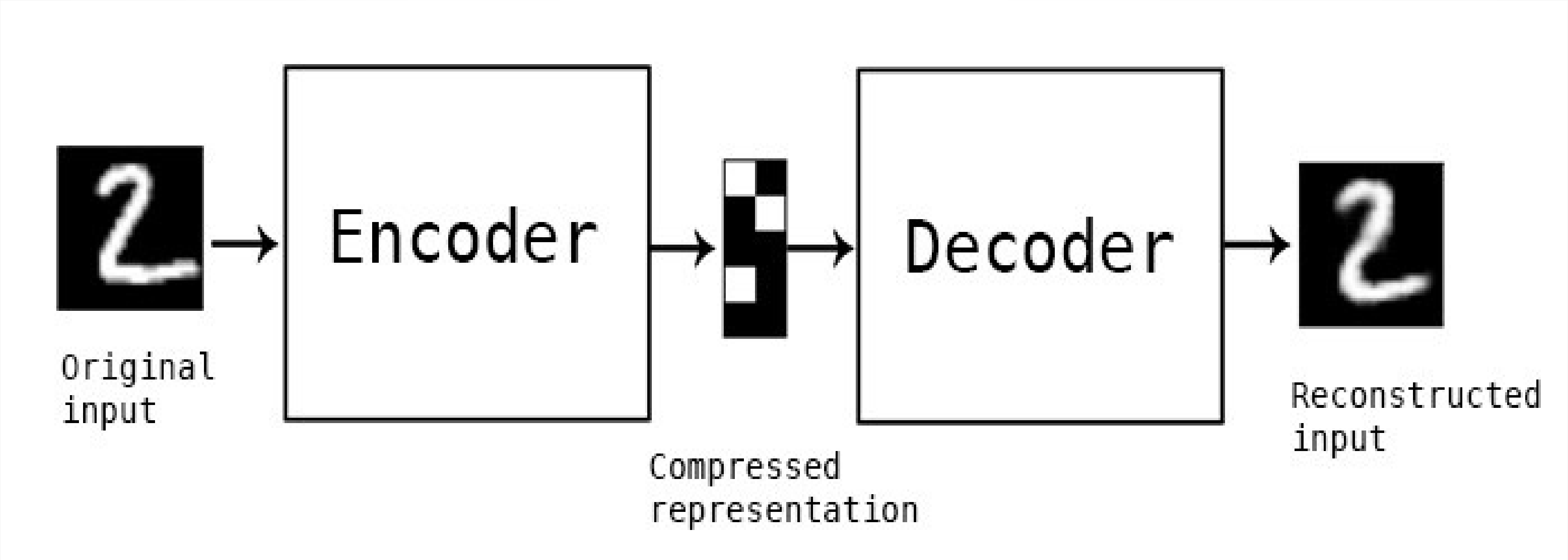
Introduction to Autoencoders

What are autoencoders?



- Autoencoding is a data compression algorithm where the compression and decompression functions are
  - 1) data-specific
  - 2) lossy, and
  - 3) learned automatically from examples rather than engineered by a human
  - functions are implemented with neural networks.

- Autoencoders are data-specific, which means that they will only be able to compress data similar to what they have been trained on (unlike MP3)
- Autoencoders are lossy, which means that the decompressed outputs will be degraded compared to the original inputs
- Autoencoders are learned automatically from data examples, which is a useful property
- To build an autoencoder, you need three things: an encoding function, a decoding function, and a distance function between the amount of information loss between the compressed representation of your data and the decompressed representation

Are they good at data compression?

- Usually, not really. In picture compression for instance, it is pretty difficult to train an autoencoder that does a better job than a basic algorithm like JPEG,
- and typically the only way it can be achieved is by restricting yourself to a very specific type of picture (e.g. one for which JPEG does not do a good job).

What are autoencoders good for?

- In 2012 they briefly found an application in greedy layer-wise pretraining for deep convolutional neural networks
- but this quickly fell out of fashion as we started realizing that better random weight initialization schemes were sufficient for training deep networks from scratch
- In 2014, batch normalization started allowing for even deeper networks,
- from late 2015 we could train arbitrarily deep networks from scratch using residual learning

What are autoencoders good for?

- Today two interesting practical applications of autoencoders are **data denoising** and **dimensionality reduction for data visualization**

- encoding_dim = 32
- input_img = Input(shape=(784,))
- encoded = Dense(encoding_dim, activation='relu')(input_img)
- decoded = Dense(784, activation='sigmoid')(encoded)
- autoencoder = Model(input_img, decoded)

Deep autoencoder

- We do not have to limit ourselves to a single layer as encoder or decoder, we could instead use a stack of layers, such as:

```
input_img = Input(shape=(784,))
encoded = Dense(128, activation='relu')(input_img)
encoded = Dense(64, activation='relu')(encoded)
encoded = Dense(32, activation='relu')(encoded)

decoded = Dense(64, activation='relu')(encoded)
decoded = Dense(128, activation='relu')(decoded)
decoded = Dense(784, activation='sigmoid')(decoded)
```

Convolutional autoencoder

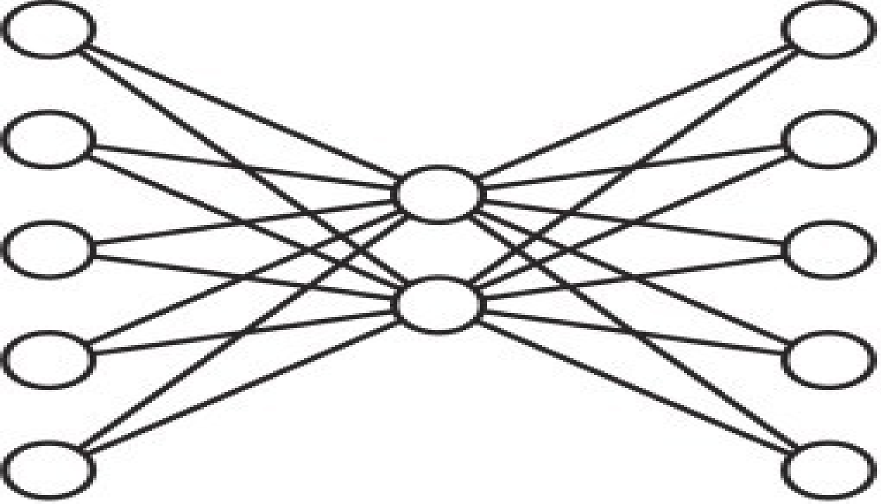
- If convolutional neural networks (convnets) as encoders and decoders
- Then those autoencoders are called conv AE
- Used for images

Convolutional autoencoder

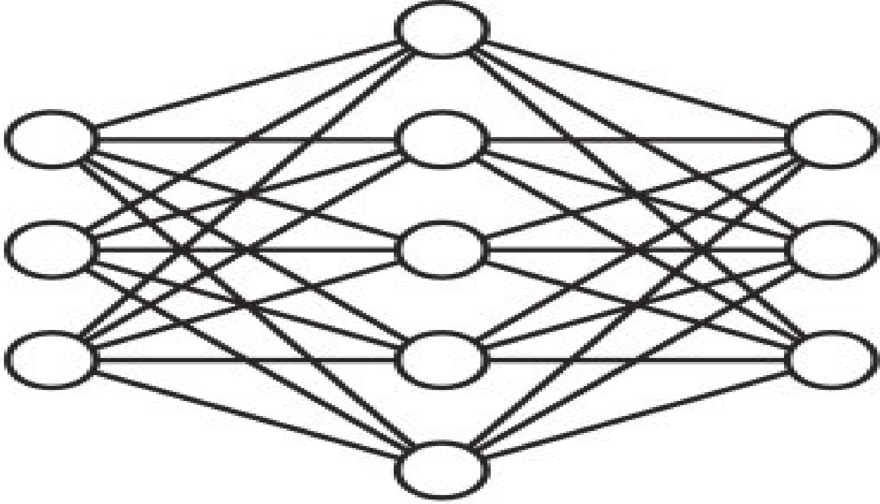
```
input_img = Input(shape=(28, 28, 1))
x = Conv2D(16, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
encoded = MaxPooling2D((2, 2), padding='same')(x)
```

Convolutional autoencoder

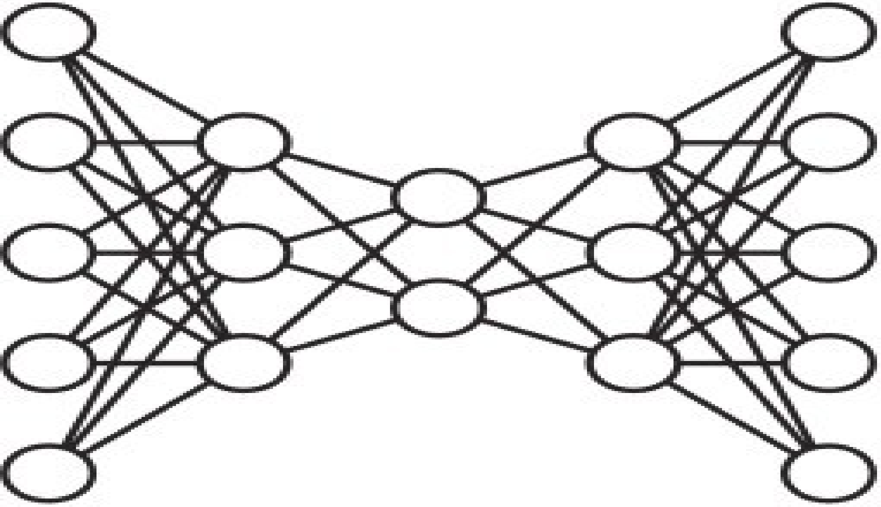
```
x = Conv2D(8, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(16, (3, 3), activation='relu')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)
```



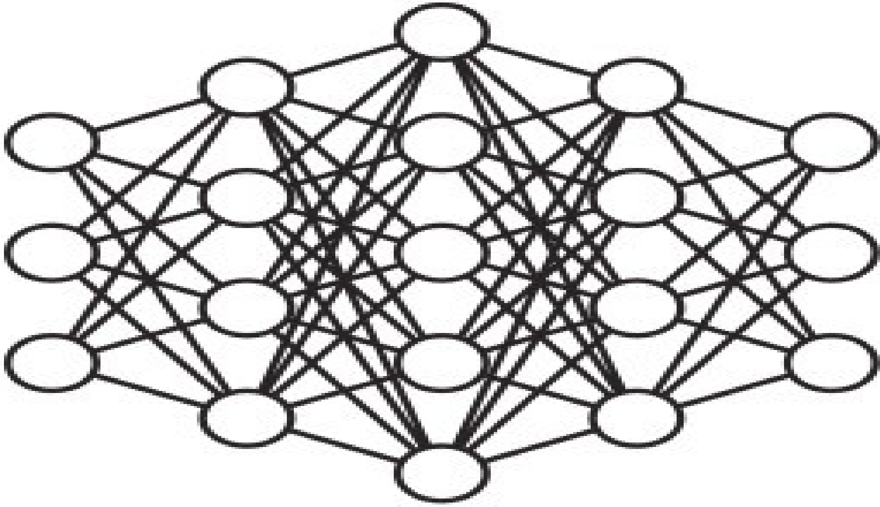
(a) Shallow undercomplete



(b) Shallow overcomplete



(c) Deep undercomplete



(d) Deep overcomplete

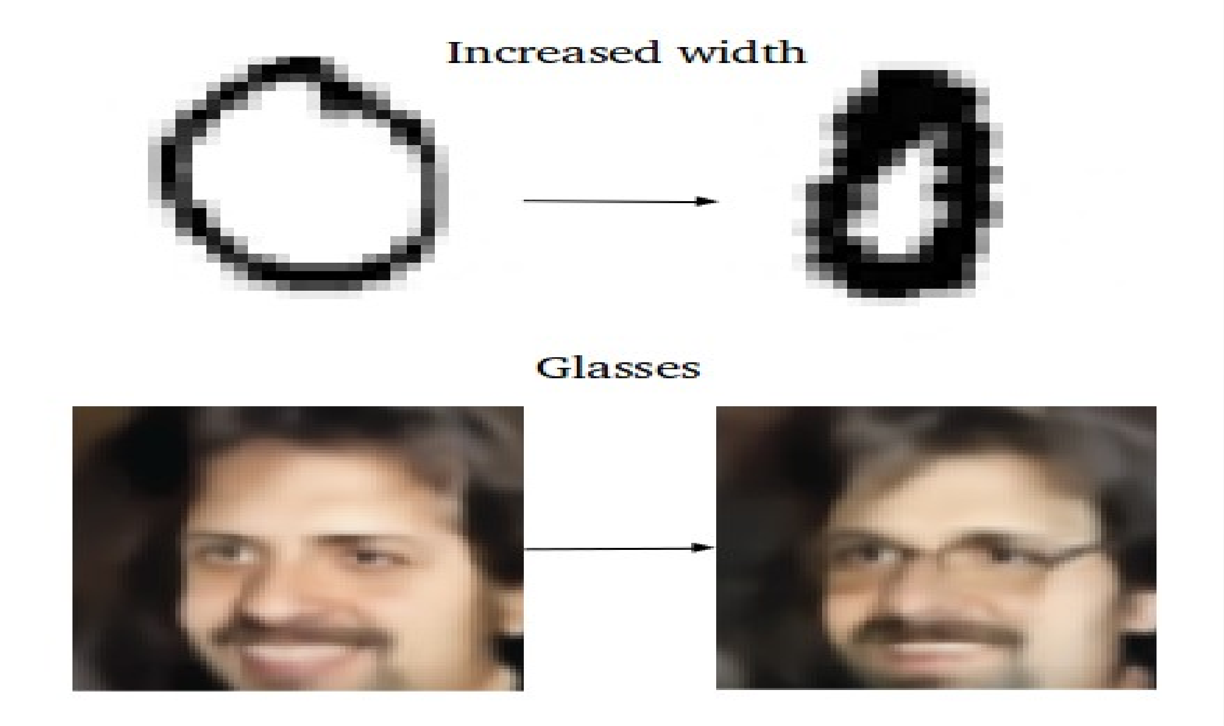
Sparse Autoencoder

```
inputs = Input(shape=(784,))
h = Dense(64, activation='sigmoid',
activity_regularizer=activity_l1(1e-5))(inputs)
outputs = Dense(784)(h)
```

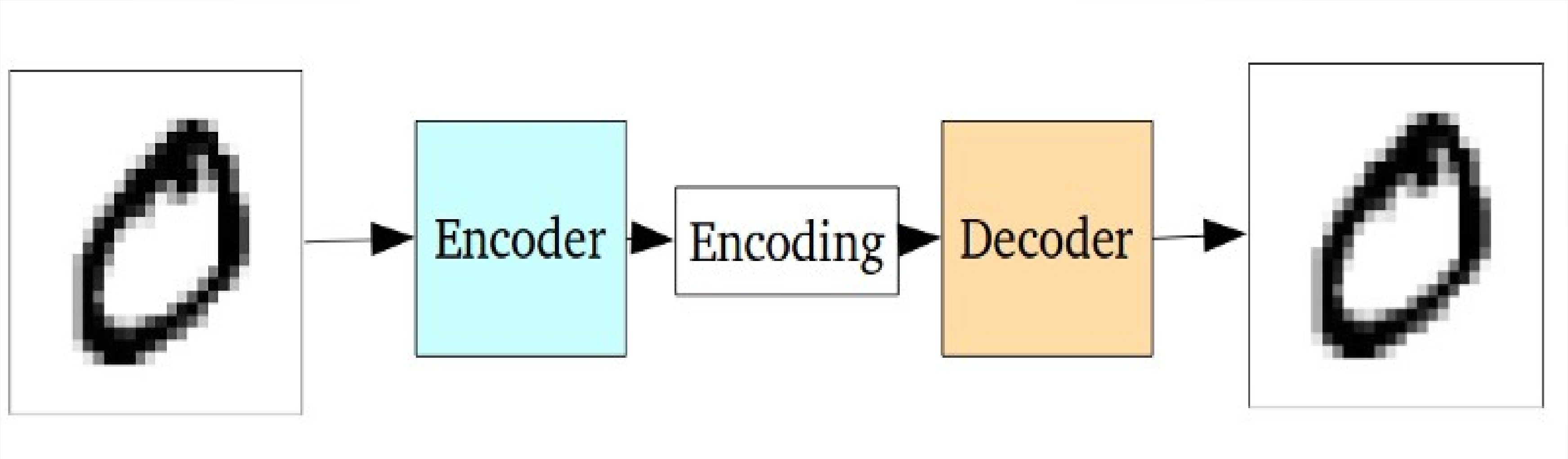
- Notice in our hidden layer, we added an $\ell 1$ penalty.
- As a result, the representation is now sparser compared to the standard Autoencoder

Variational Autoencoders

- used in creating your own generative text, art,images and even music
-



- When using generative models, you could simply want to generate a random, new output, that looks similar to the training data
- you can certainly do that too with VAEs. But more often, you'd like to alter, or explore variations on data you already have, and not just in a random way either, but in a desired, specific direction.
- This is where VAEs work better than any other method currently available.


Encoder
Encoding
Decoder

- The entire network is usually trained as a whole.
- The loss function is usually either the mean-squared error or cross-entropy between the output and the input
- known as the reconstruction loss,
- which penalizes the network for creating outputs different from the input.

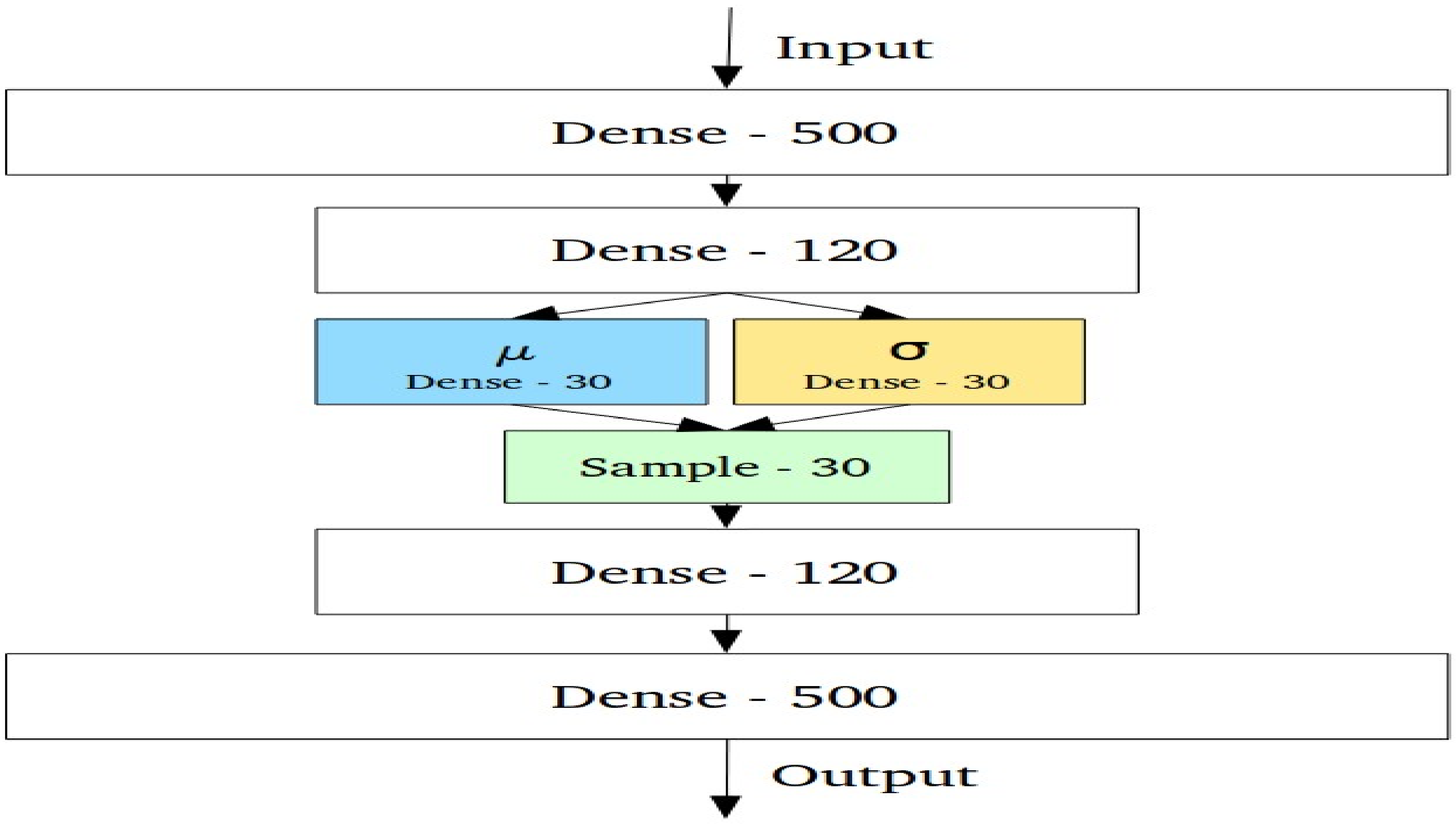
- The encoder learns to preserve as much of the relevant information as possible in the limited encoding
- Intelligently discard irrelevant parts.
- The decoder learns to take the encoding and properly reconstruct it into a full image.
- Together, they form an autoencoder

- Standard autoencoders learn to generate compact representations and reconstruct their inputs well
- Used in few applications like denoising autoencoders
- they are fairly limited
- The fundamental problem with autoencoders, for generation, is that the latent space they convert their inputs to and where their encoded vectors lie, may not be continuous

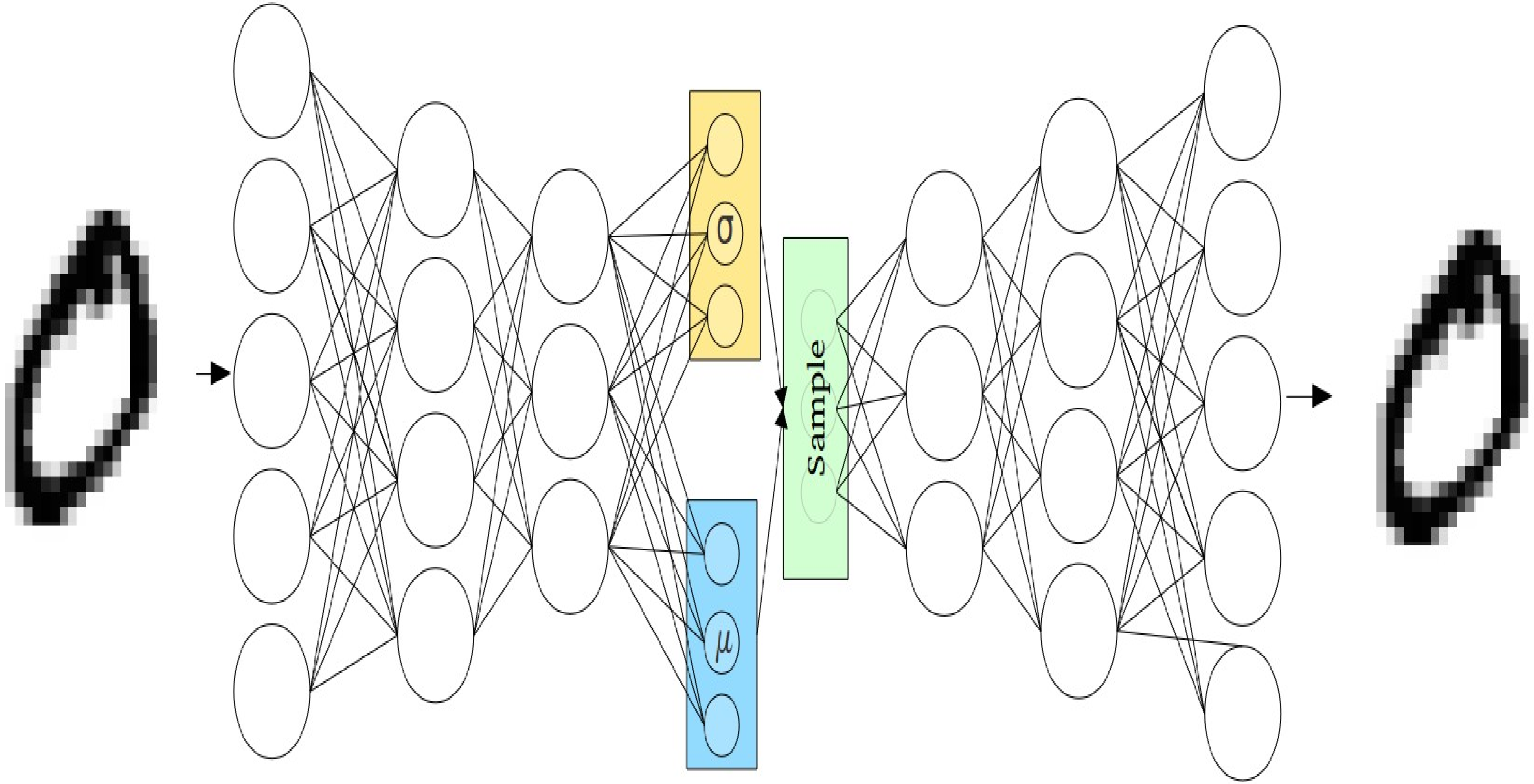
- But when you’re building a generative model,
- you don’t want to prepare to replicate the same image you put in.
- You want to randomly sample from the latent space, or generate variations on an input image, from a continuous latent space.

- Variational Autoencoders (VAEs) have one fundamentally unique property that separates them from vanilla autoencoders
- this property that makes them so useful for generative modeling: their latent spaces are, by design, continuous, allowing easy random sampling and interpolation.

- encoder not output an encoding vector of size n, rather, outputting two vectors of size n:
- a vector of means, μ
- another vector of standard deviations, σ.



Input
Dense - 500
Dense - 120
μ
Dense - 30
σ
Dense - 30
Sample - 30
Dense - 120
Dense - 500
Output


σ
Sample
μ