Deep Learning

CSCI 1470

Eric Ewing

Monday, 3/31/25

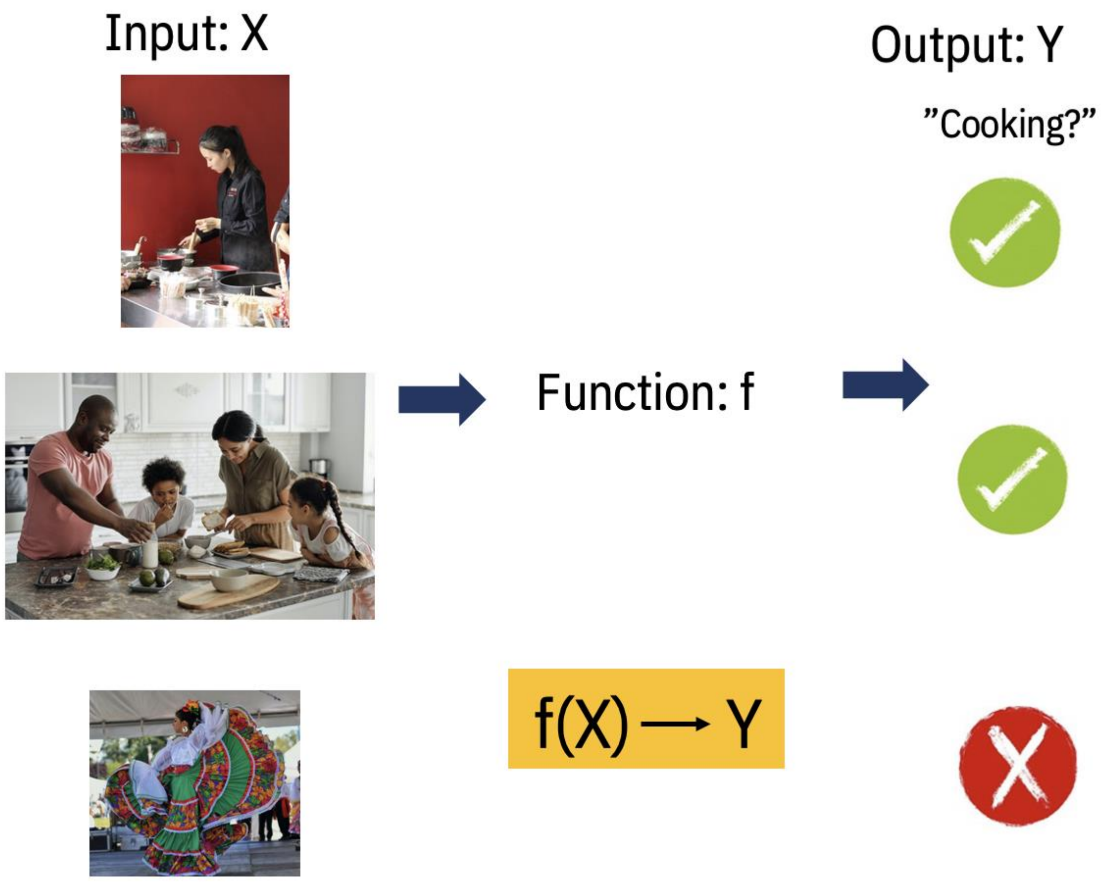
Day 25: Image Generation Day 1

ChatGPT: "Deep Space animated image"

Recap: Supervised Learning

Is it an image of someone cooking?

- Supervised learning requires **labels**
- Learn a function that takes in input features and outputs labels



What are some pros and cons of supervised learning?

Recap: Supervised Learning

Pros:

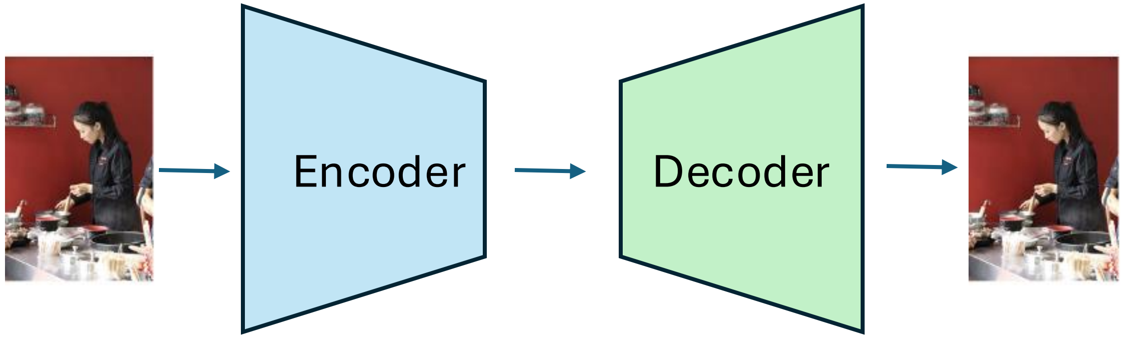
- Can produce very high quality models with sufficient data
- Performance is easy to measure (e.g., accuracy)

Cons:

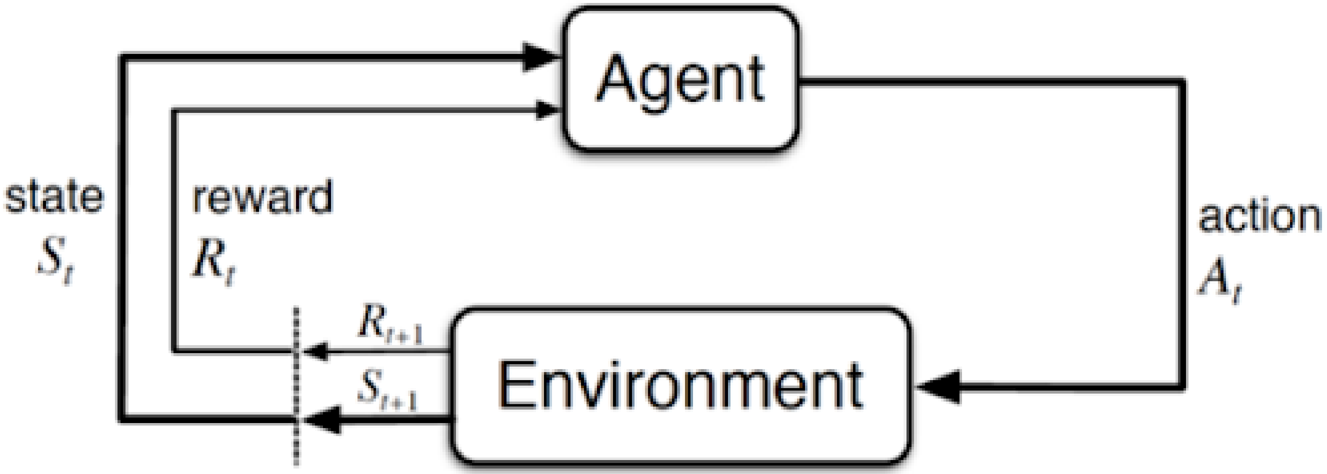
- Reliant on **availability** of labels
- Reliant on **quality** of labels

What's Left?

Unsupervised Learning: Learning without labels



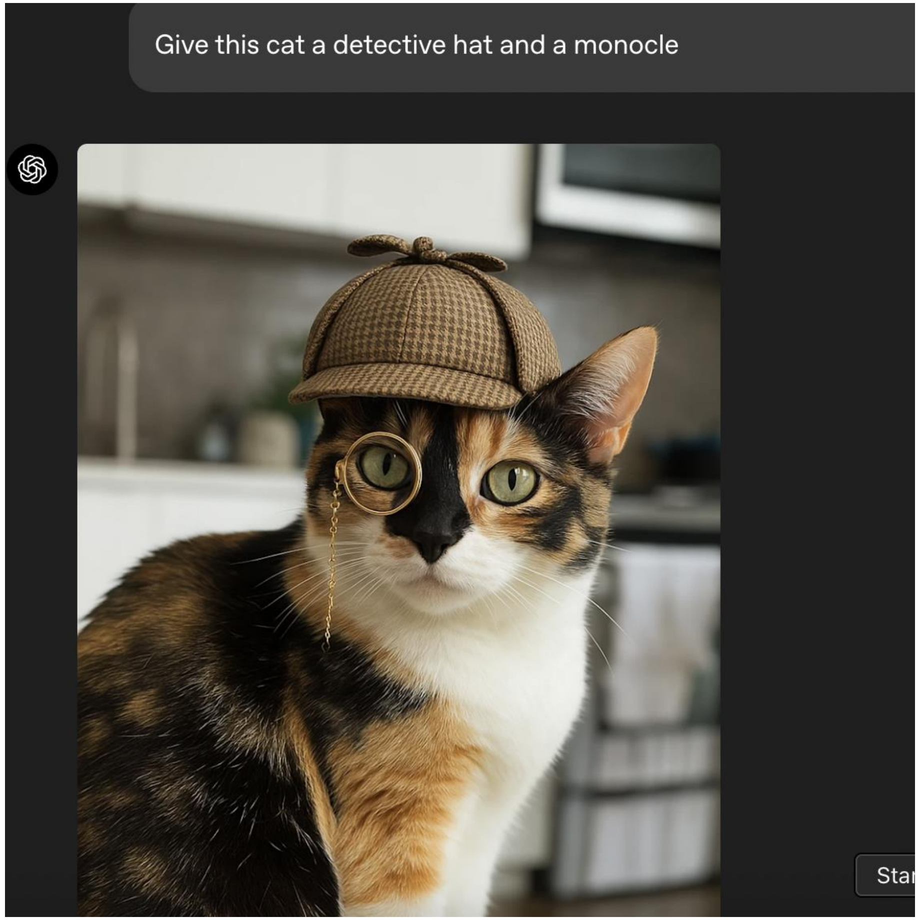
Reinforcement Learning: Learning from experiences



This Week in Deep Learning

ChatGPT 4o added image generation and editing capabilities

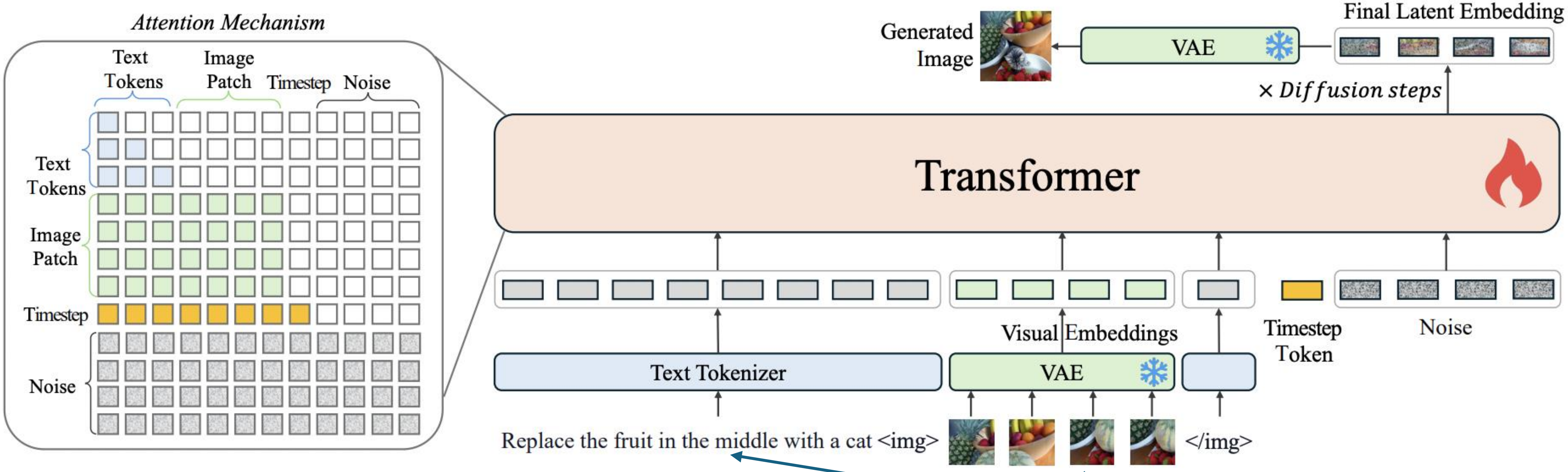
- ChatGPT used to rely on Dall-E for image generation, now they've introduced a new model
- Reliably incorporates text commands and improved image generation capabilities



What *might* this look like?

Output image

Output embedding vector



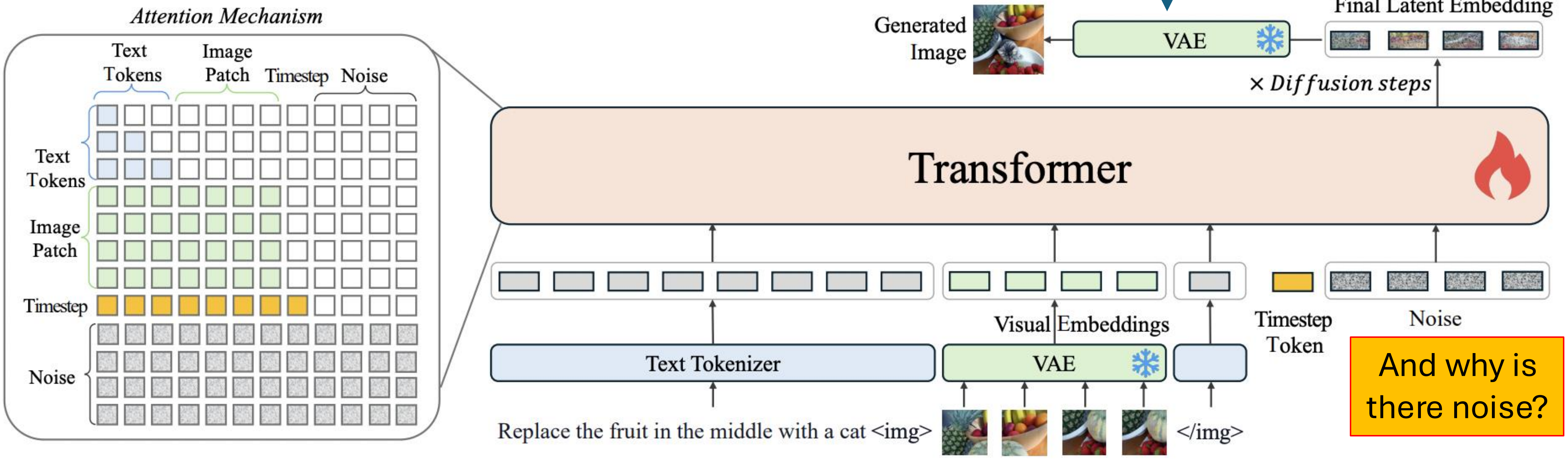
Take in multiple modes of input (text, images, etc.)
(Multi-modal learning)

Source: OmniGen: Unified Image Generation: https://arxiv.org/pdf/2409.11340

What *might* this look like?

VAE: Variational Auto-Encoder
Noise: for “Diffusion” process

What’s this?



And why is there noise?

And This?

Source: OmniGen: Unified Image Generation: https://arxiv.org/pdf/2409.11340

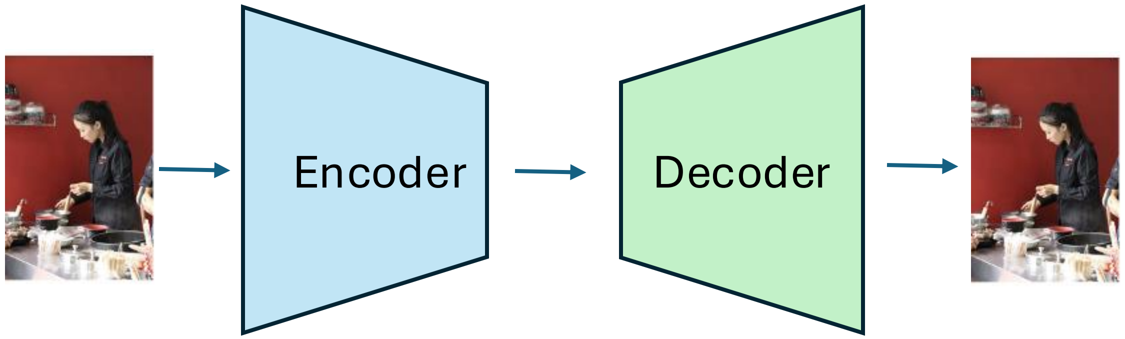
Foundation Models

Key Question: What is the equivalent of Language Modeling for data other than natural language? (From last class)

Desired Properties:

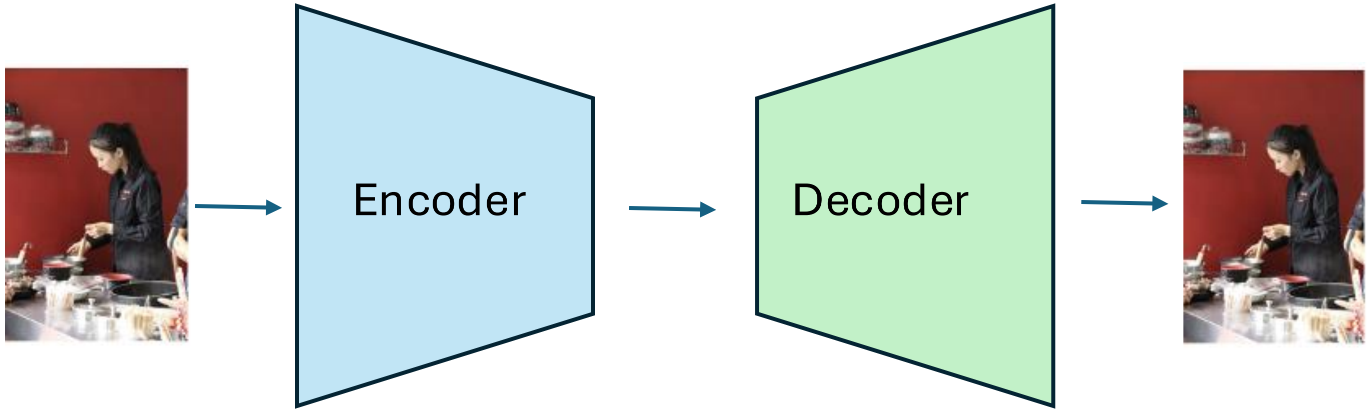
1. No need for human labeling
2. Large amount of data available
3. Ability to learn a "general" representation of the data

One such method for images: Can you output the same image that comes as input?



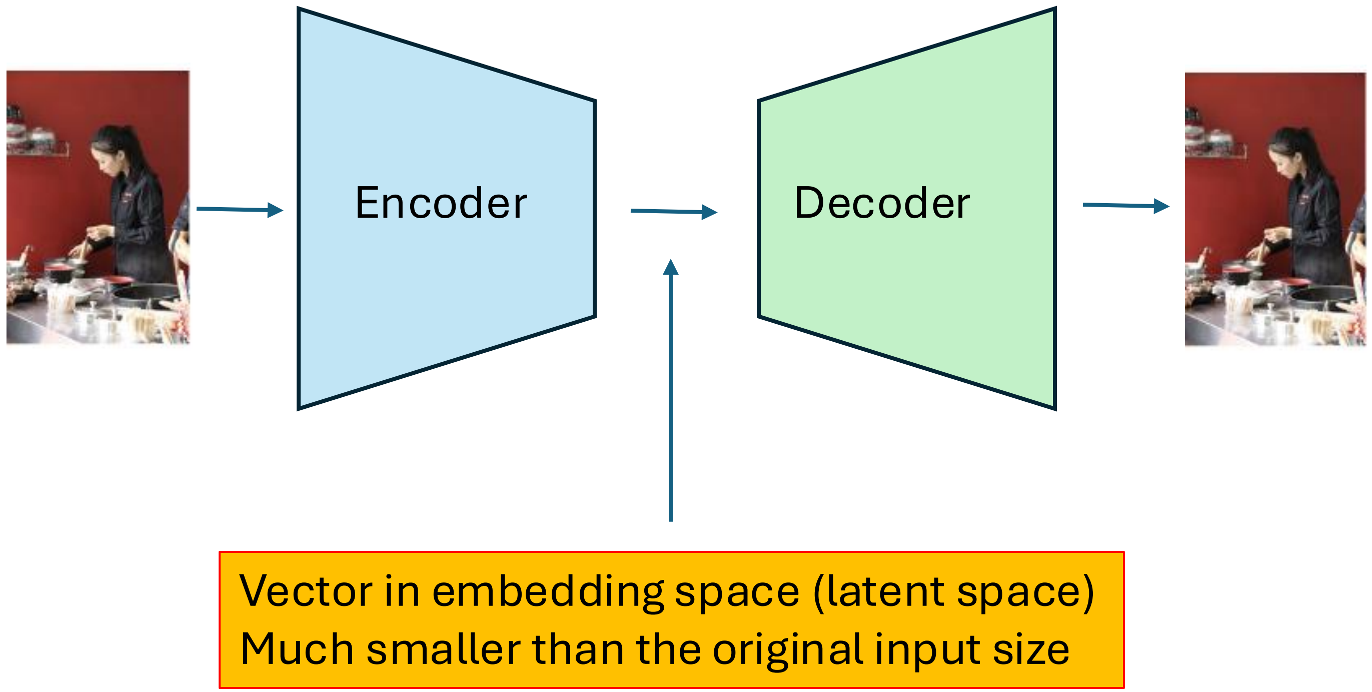
Autoencoders

Autoencoder: an encoder-decoder architecture that tries to produce its own input



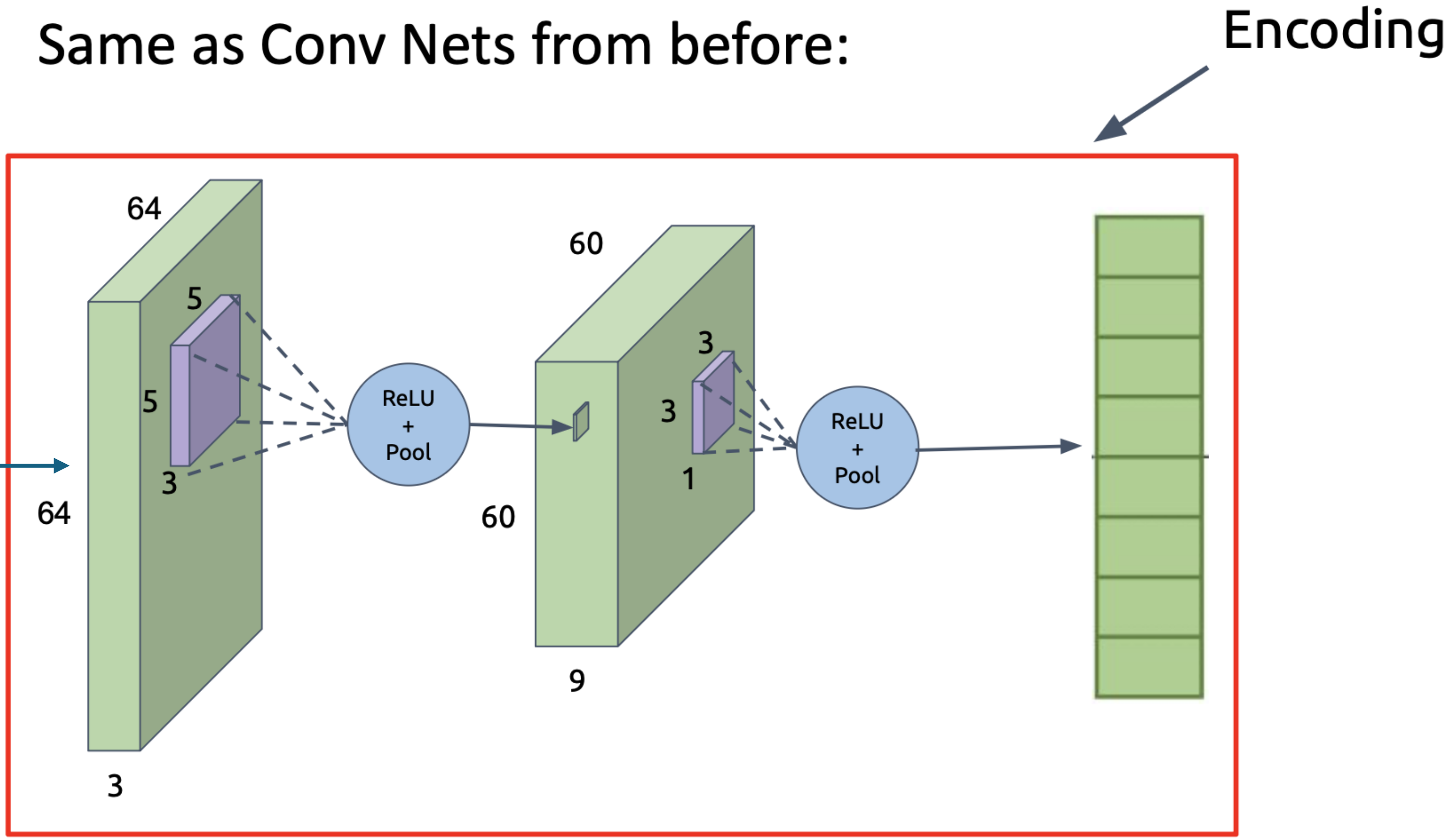
But what's hard about this? It's very easy to learn a function that outputs the input to the function (i.e., the identity function)

Autoencoders



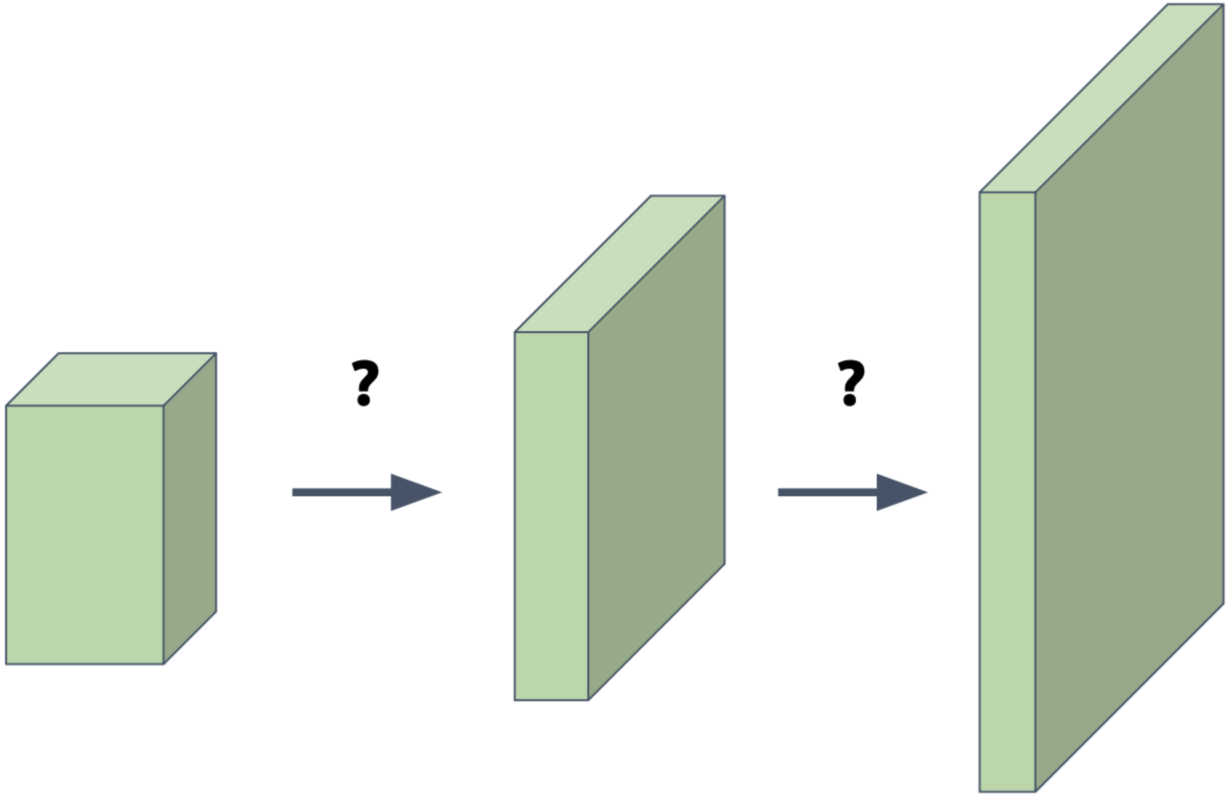
Convolutional Autoencoders: Encoding

Same as Conv Nets from before:



Autoencoders: Decoding

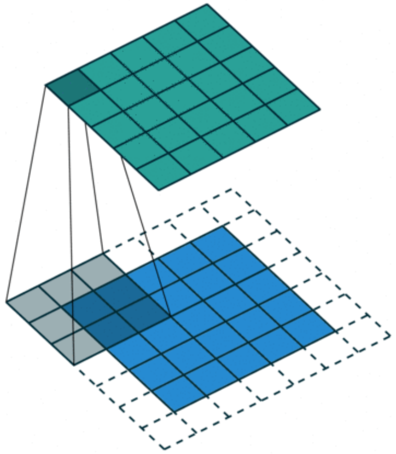
- Convolution as we know it only keeps resolution same or decreases it
- How do we go up in resolution?



Transposed Convolution

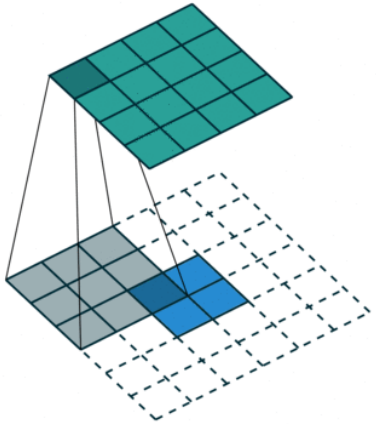
- Convolution Idea: Slide kernel along an input matrix

Blue: output
Green: input

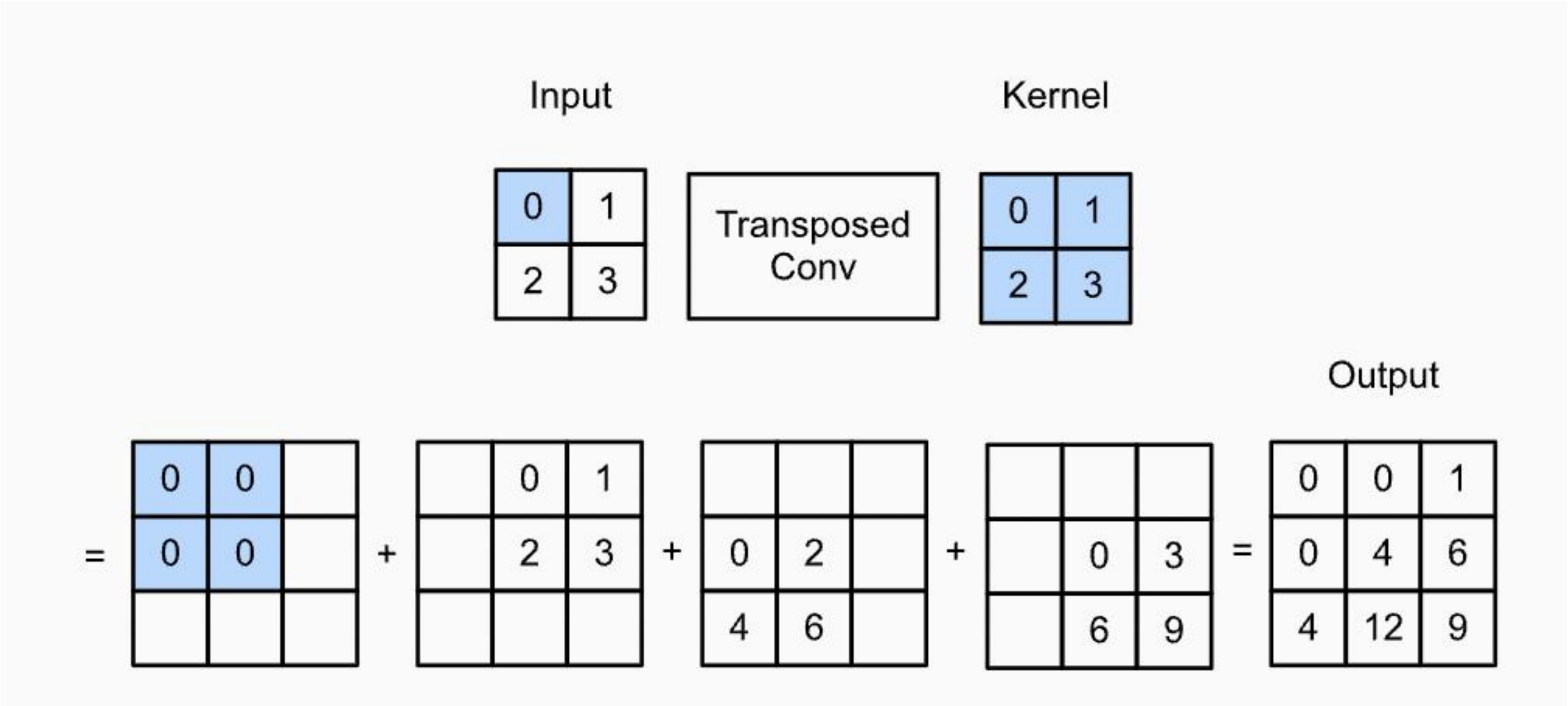


- Transpose Convolution:

Blue: Input
Green: Output



Source: https://github.com/vdumoulin/conv_arithmetic/tree/master



(stride of 1)

Source: https://d2l.ai/chapter_computer-vision/transposed-conv.html

Transpose Convolution in Tensorflow

```
tf.nn.conv2d_transpose(input, filters, output_shape, strides, padding='SAME')
```

4D tensor of shape [batch, height, width, in_channels]

4-D Tensor with shape [height, width, output_channels, in_channels]

length 4 1D tensor representing the output shape.

Strides along each dimension (list of integers)

String representing type of padding

Documentation here: https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/conv2d

Specifying Output Size

- An image can be the result of the same convolution on images of different resolution
- We need to specify which one we want.

57	60
66	61

1	2	3
4	5	6
7	8	9

Kernel

2	1	0	3
0	0	1	2
3	1	2	0
0	2	2	1

2	1	0	3	0
0	0	1	2	0
3	1	2	0	0
0	2	2	1	0
0	0	0	0	0

Any questions?

Transpose Convolution in Keras

```
tf.keras.layers.Conv2DTranspose(filters, kernel_size, strides, padding='SAME')
```

Number of filters (Integer)

Size of Convolution Window (tuple)

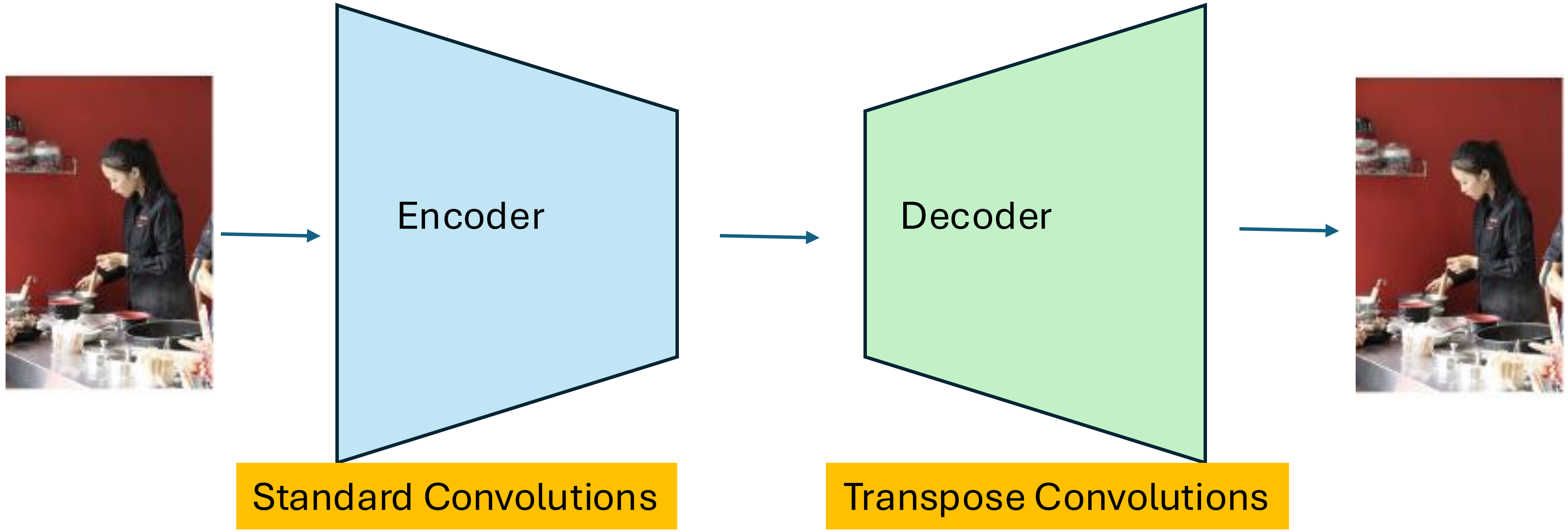
Strides along each dimension (list of integers)

String representing type of padding

Note: Output Shape is inferred, but can be specified via the "output_padding" parameter

Documentation here: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2DTranspose

Convolutional Autoencoder



Loss Function

What do you think is an appropriate loss function?

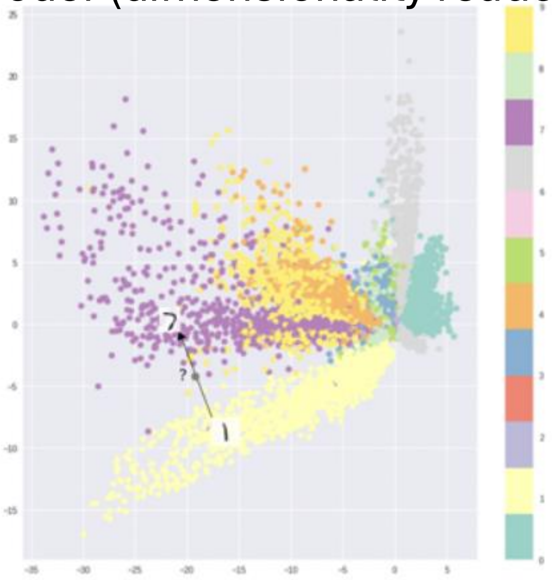
Reconstruction loss (MSE): How far is each output pixel from the corresponding input pixel?

Autoencoders

What do Autoencoders actually learn?

1. Encoder learns a dimensionality reduction (from image to vector)
2. Decoder learns an image generation function (from vector to image)

Encodings of MNIST data points with a trained autoencoder (dimensionality reduced further by PCA)

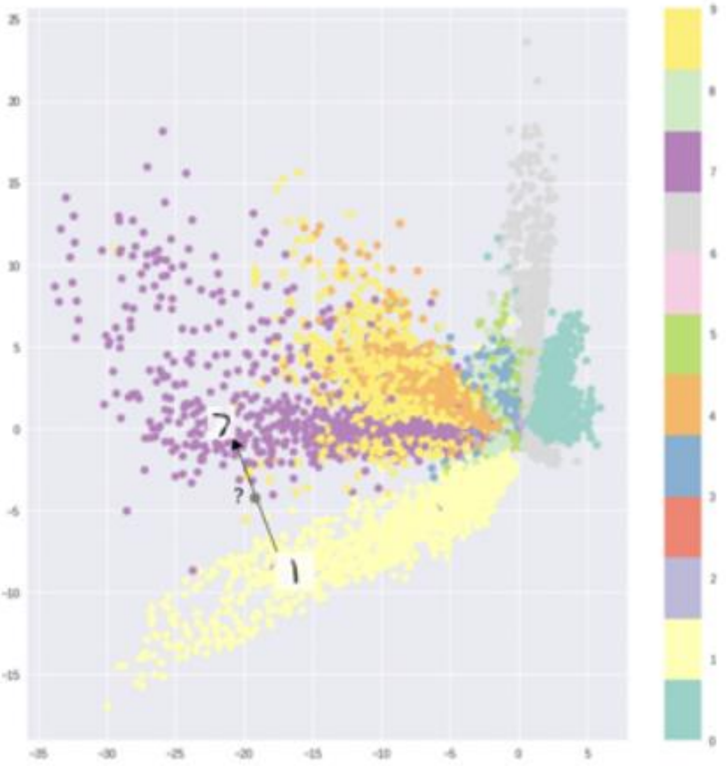


Encoders can be used to learn insights into structure of data

Decoders can be used to generate “new” images

Generating Images

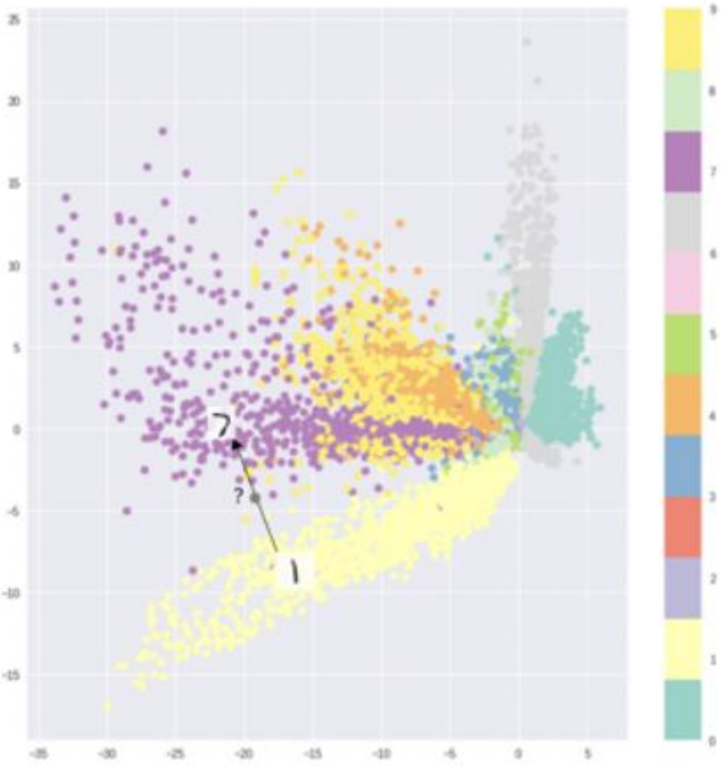
- How can we generate a “new” image using a decoder?
- Sample a vector in latent space and send it to the decoder…
- But how do you choose which vector?
- What if you wanted to generate a specific image? How would you find the right vector?



Issues with Autoencoders

- Vectors close together in latent space may not produce similar outputs
- Tend to overfit data (struggle to produce "new" outputs)

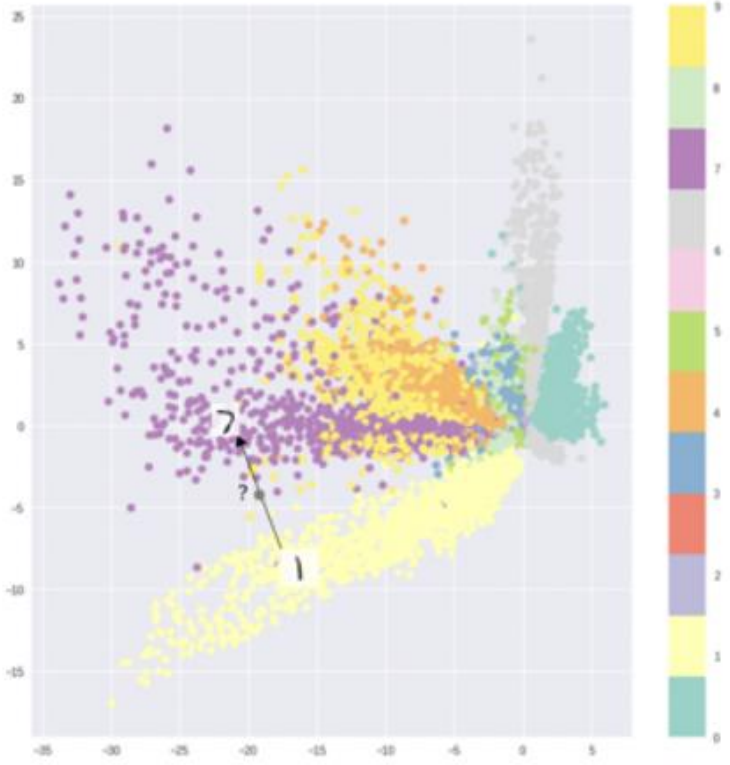
How to address issues with overfitting outputs? Try to learn more *variation* in outputs.



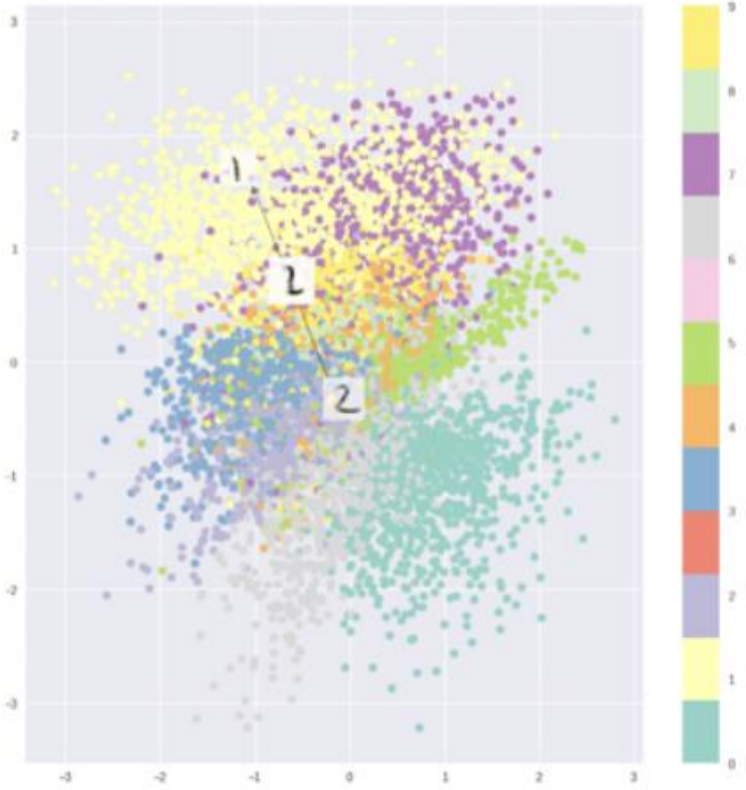
Issues with Autoencoders

What might a better latent space look like for generation?

Autoencoder



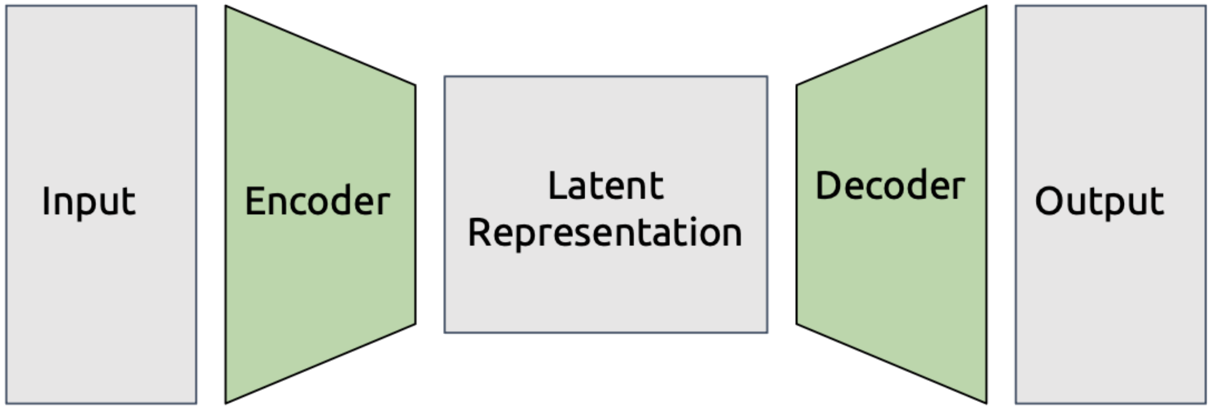
Variational Autoencoder



Building up the VAE Architecture

If we were to describe an autoencoder functionally:

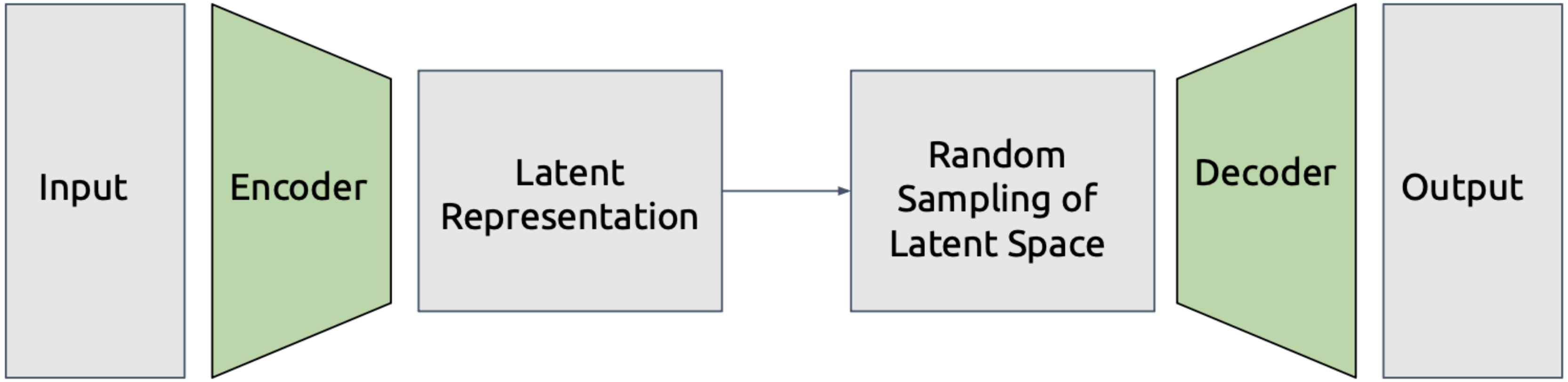
$$\texttt{Output = Decoder}(\underbrace{\texttt{Encoder(Input)}}_{\text{Latent Representation}})$$



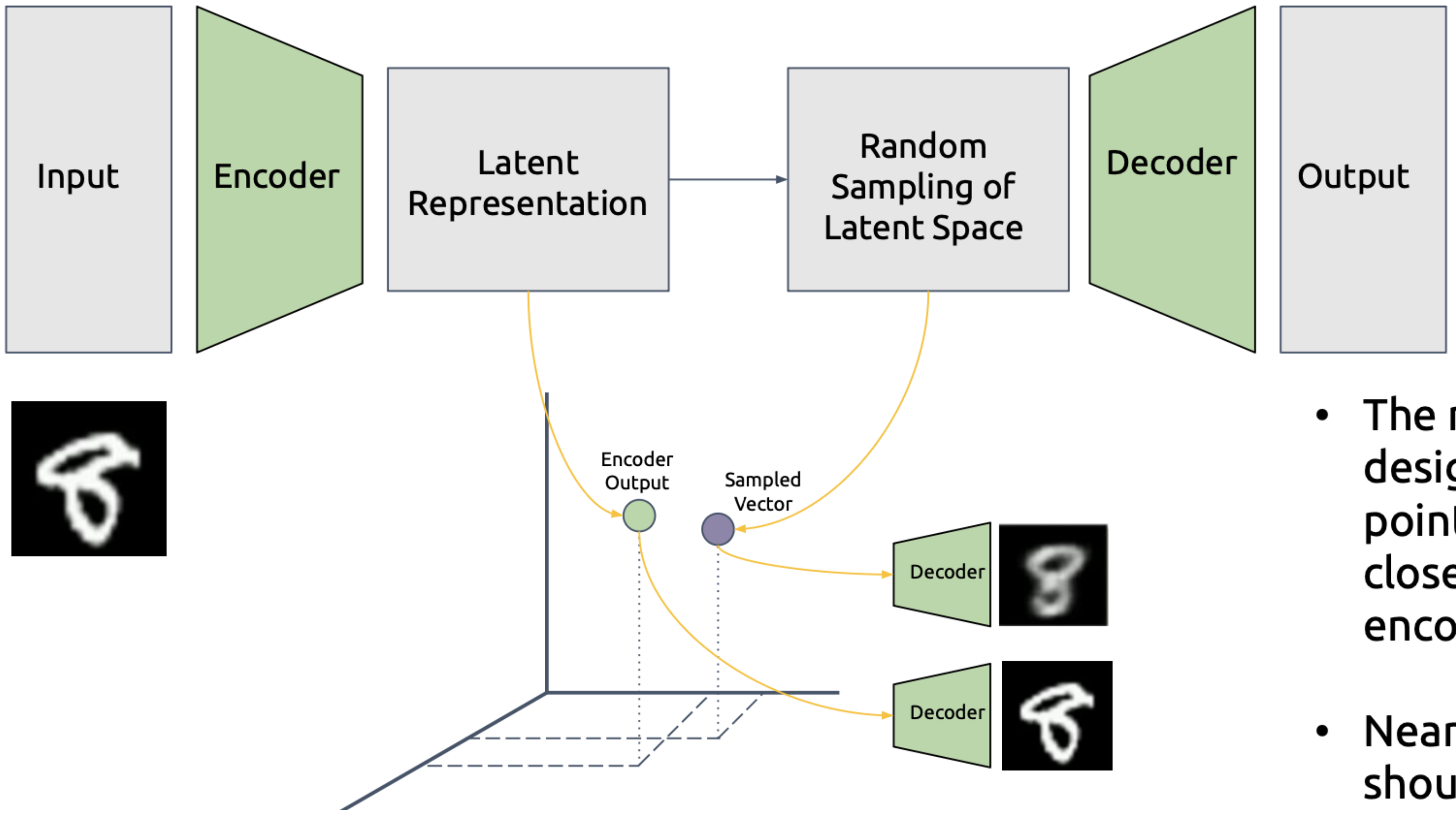
Building up the VAE Architecture

For variational autoencoders, we also do a random sampling operation at the bottleneck

```
Output = Decoder(random_sample(Encoder(Input)))
```



How does random sampling in latent space lead to variation?



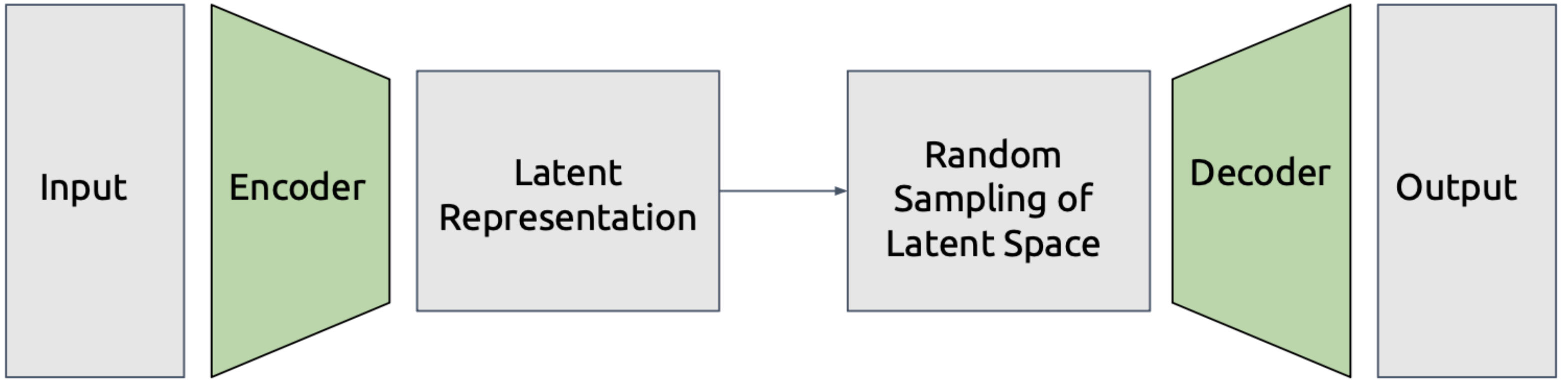
- The random sampling should be designed to produce random points in latent space that are close to the output of the encoder
- Nearby points in the latent space should decode to similar images

How should `random_sample` be defined?

`Output = Decoder(`**`random_sample`**`(Encoder(Input)))`

- We want the sample to be close to the encoder output
- One option: sample from a Gaussian centered at `Encoder(Input)`

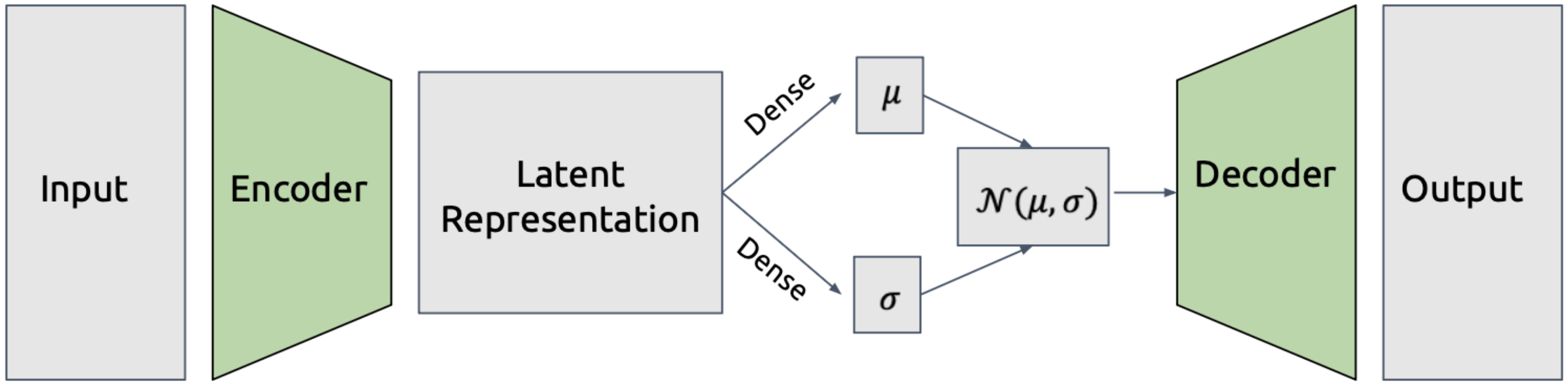
What can we modify?



How should **`random_sample`** be defined?

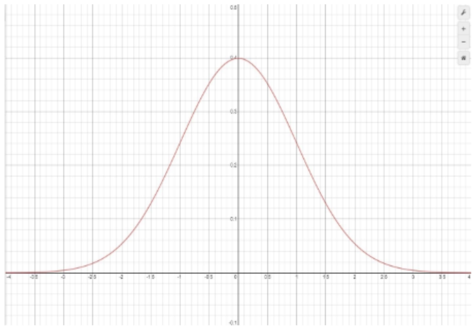
`Output = Decoder(`**`random_sample`**`(Encoder(Input)))`

- We want the sample to be close to the encoder output
- One option: sample from a Gaussian centered at `Encoder(Input)`
- Use two dense layers to convert the encoder output into the mean and standard deviation of the Gaussian

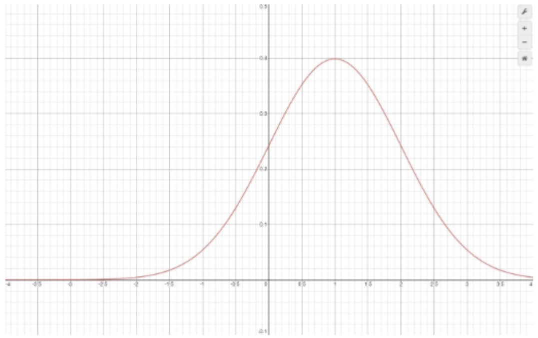


Any questions?

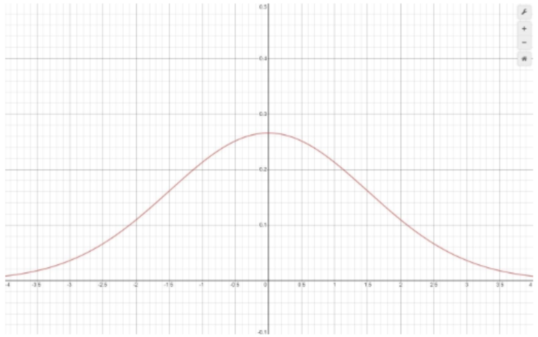
How should `random_sample` be defined?



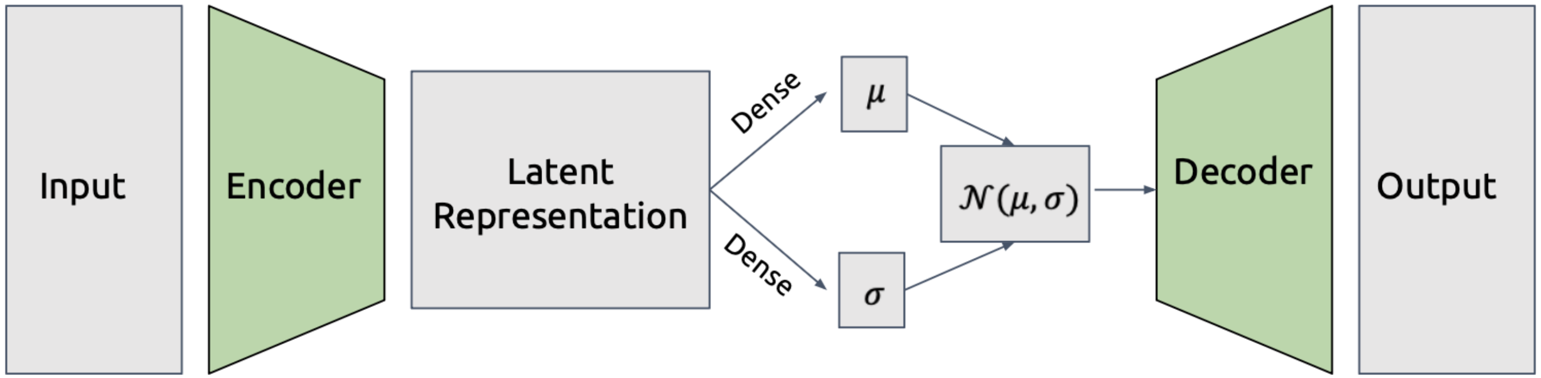
$\mu = 0$
$\sigma = 1$



$\mu = 1$
$\sigma = 1$



$\mu = 0$
$\sigma = 1.5$

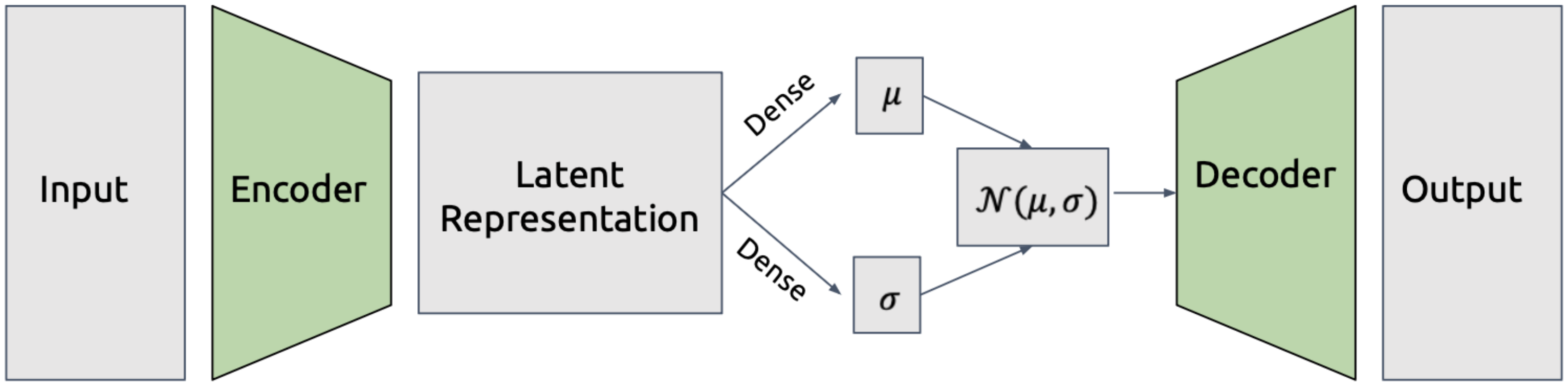


Training a VAE

Two goals:

1. Reproduce an output similar to the input (Input ≈ Output)
2. Have some variation in our output (Input $\neq$ Output)

- Seems like two conflicting goals!
- How do we resolve these two goals?



Weighted Combination of Losses

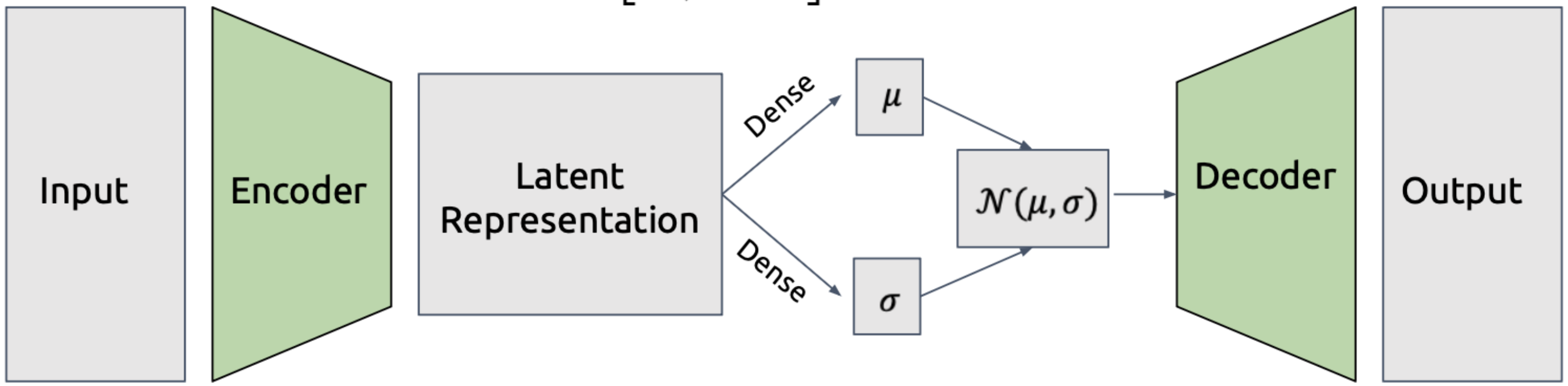
L_1 = loss associated with producing output similar to input

L_2 = loss associated with producing output with some variation to input

$$L = L_1 + \lambda L_2$$

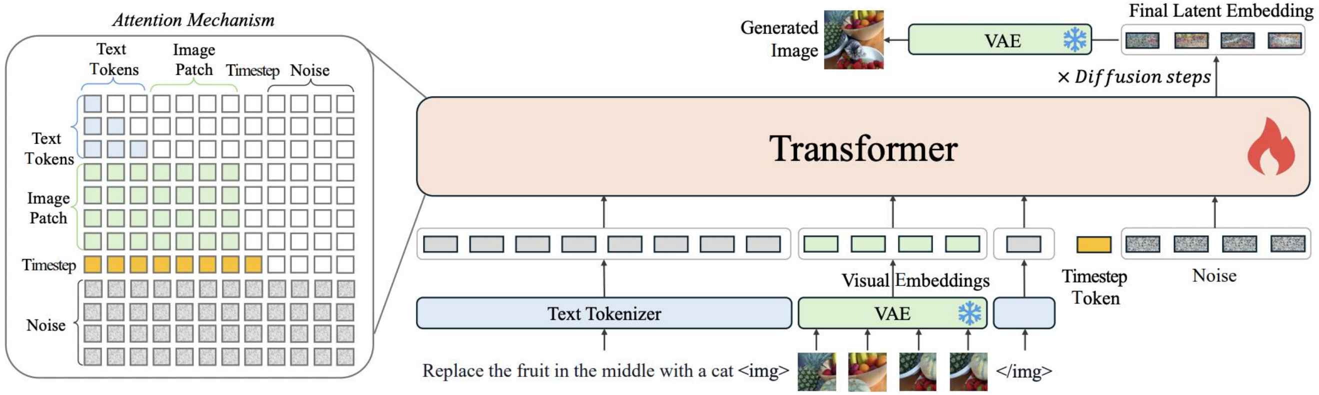
Total Loss:

$$\lambda \in [0, \infty]$$



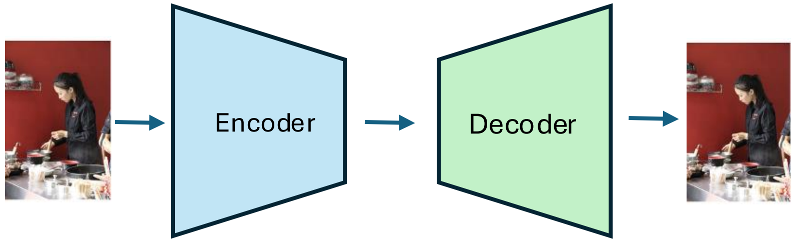
Remaining Questions

- Backprop requires that each individual step of a neural network be differentiable. VAEs sample from a Gaussian. Is that differentiable?
- How do we encourage variation in output?
- How do we generate desired types of outputs? (i.e., how do we incorporate prompts)



Recap

Unsupervised Learning



Autoencoders

Variational Autoencoders