ECE 157B Section

Homework 3

VAE

University of CA, Santa Barbara

OH reminder

- **OFFICE HOURS ARE STILL BY ZOOM**
  - Thomas: Monday 2:00 - 4:00 PM
  - Jenny: Friday 10:00 AM - 12:00 PM
  - Zoom link on Gauchospace
  - Feel free to request additional office hours, we will try to accommodate your need

- **Emails:**

  tibbetso@ucsb.edu

  yuelingzeng@ucsb.edu

Homework 3

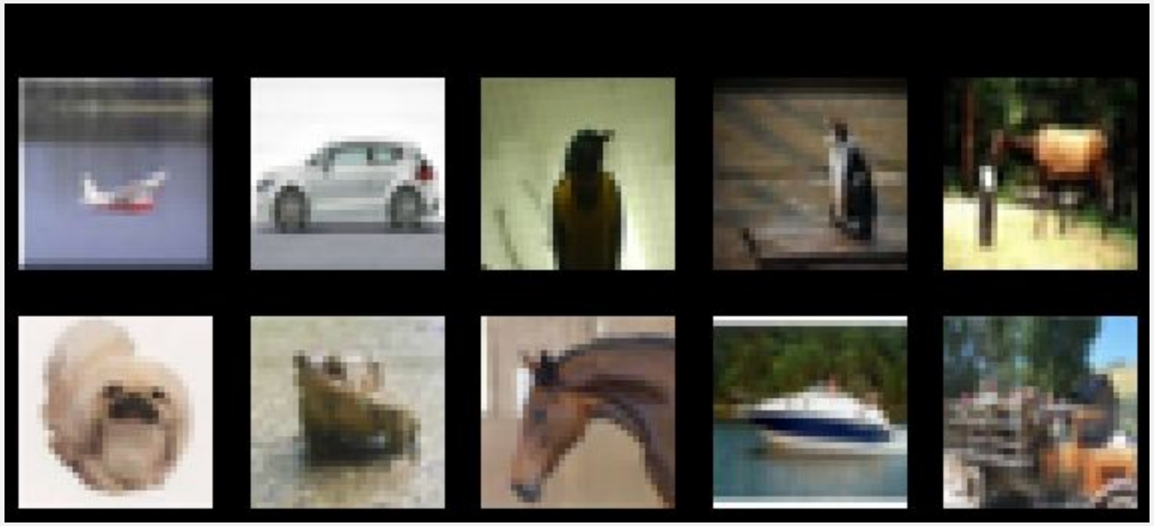
Overview

- **Building more complex models with Tensorflow/Keras**
  - Load and preprocess data
  - Build a (Variational) AutoEncoder
  - Practice with data generation

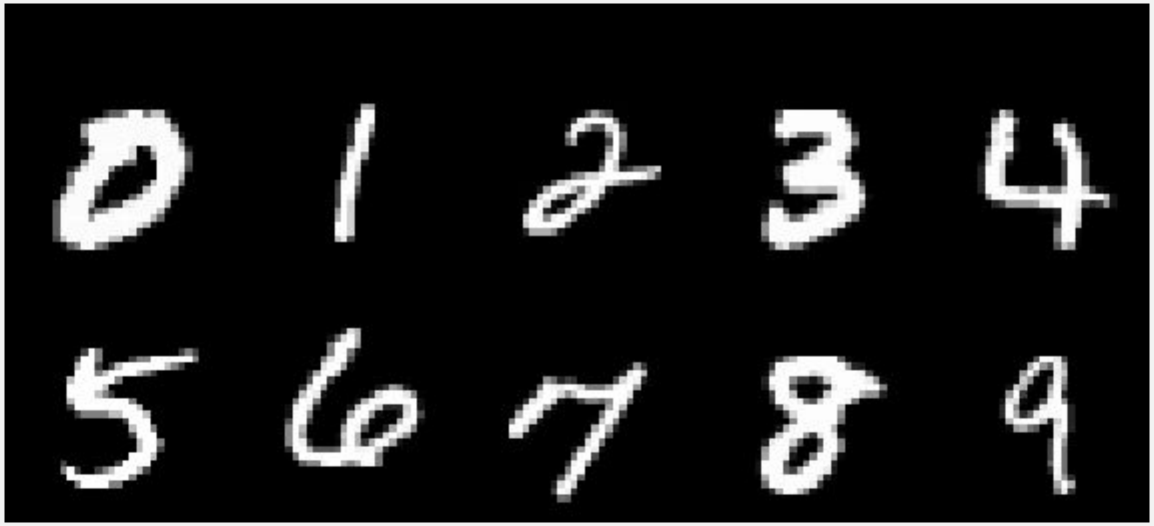
- **The effect of loss**
  - Loss with multiple objectives
  - Switching between loss in theory to loss in code

Data

CIFAR 10

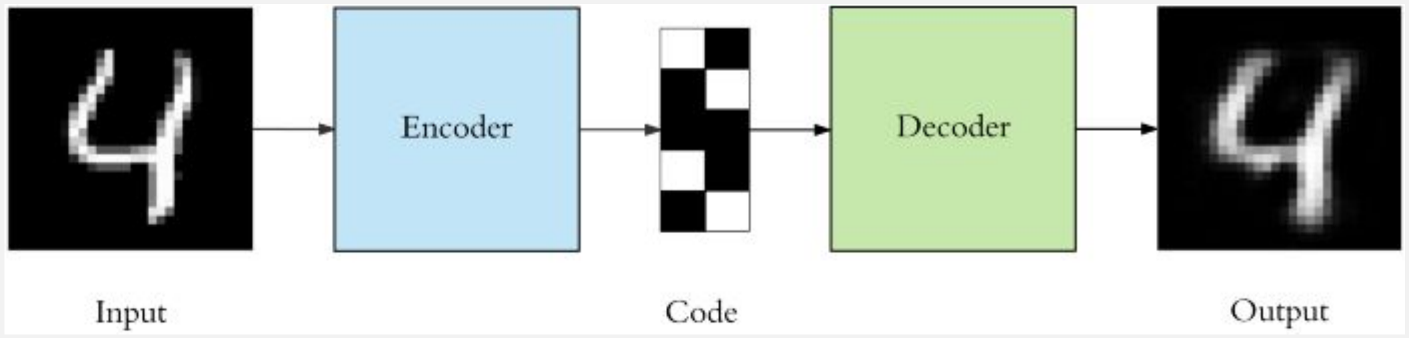


MNIST



What is an AutoEncoder?

Compress input to a small space, *and* be able to recover the original.

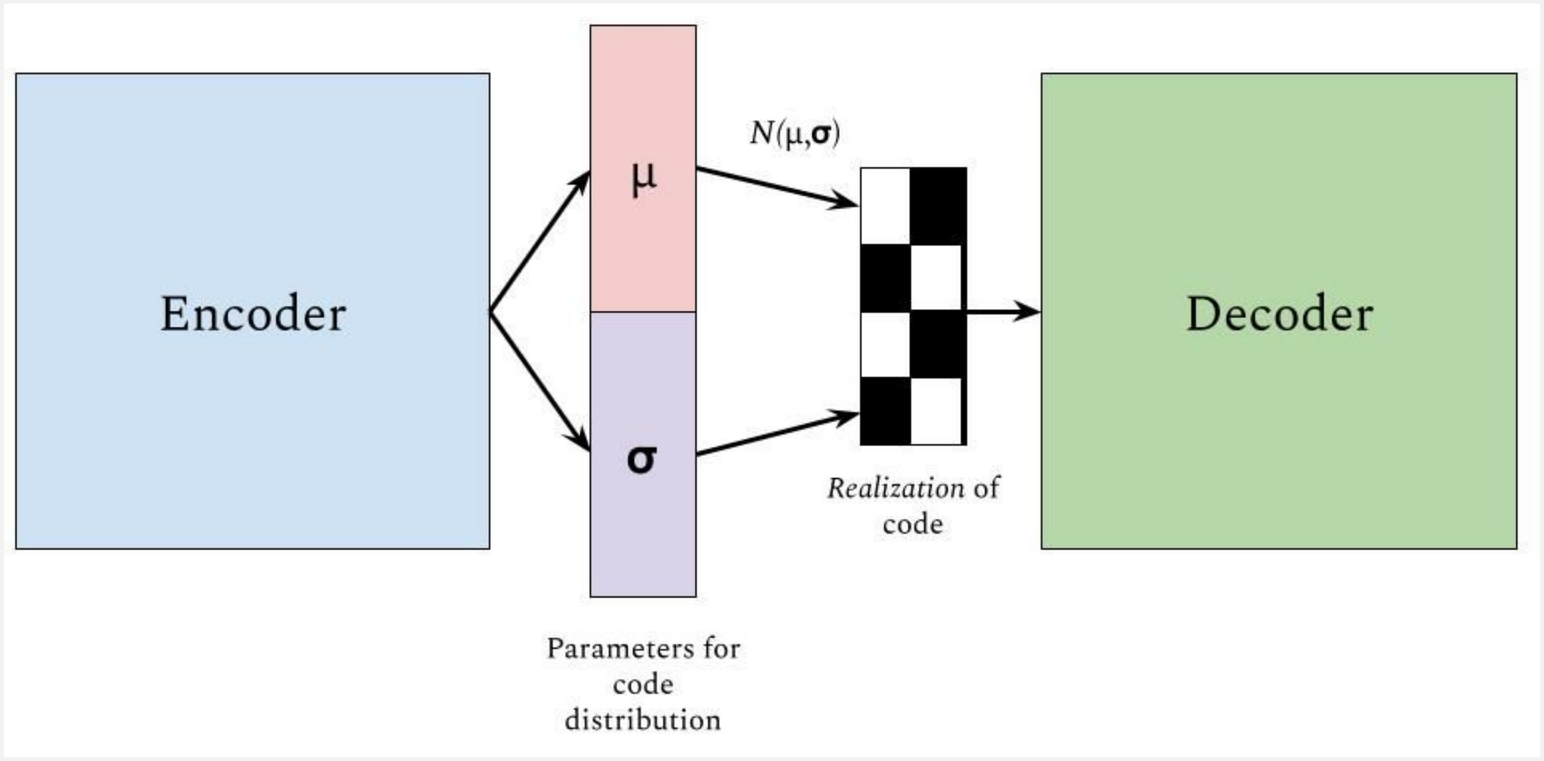


https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

Image credit from https://en.wikipedia.org/wiki/Kernel_(image_processing)

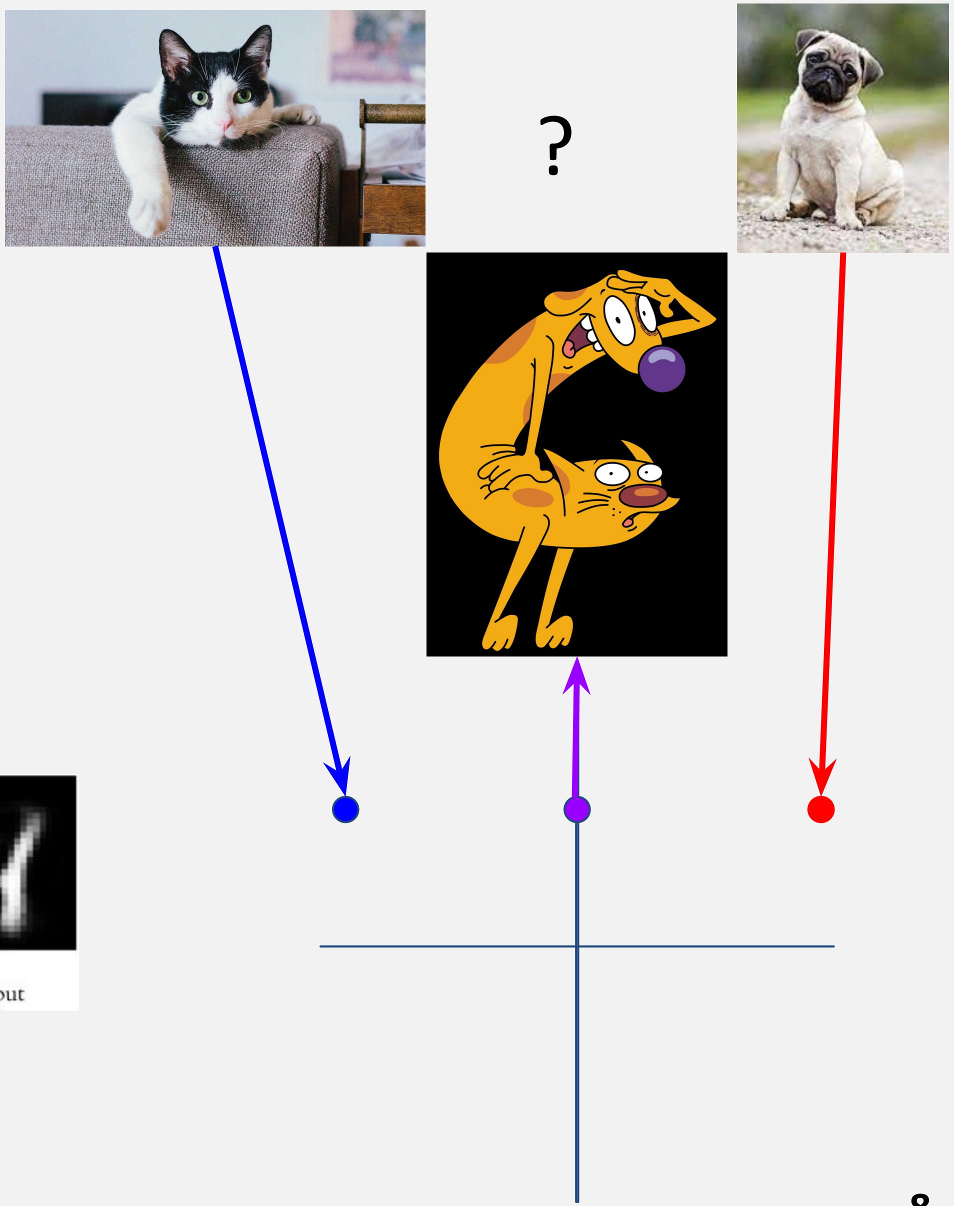
What is a Variational AutoEncoder?

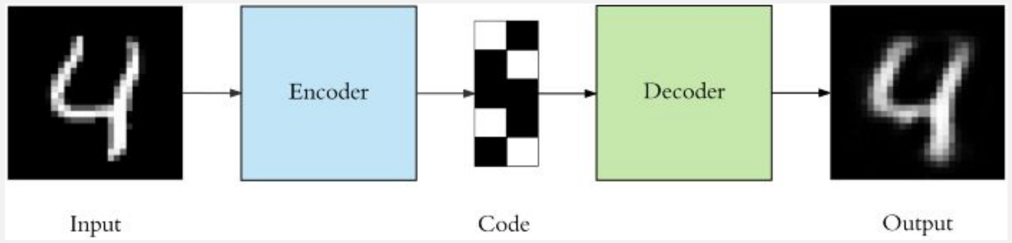
- Generate the mean and variance of *possible* codes for the sample
- Why?
  - Compression only - don't bother
  - Generation? …



What is a Variational AutoEncoder?

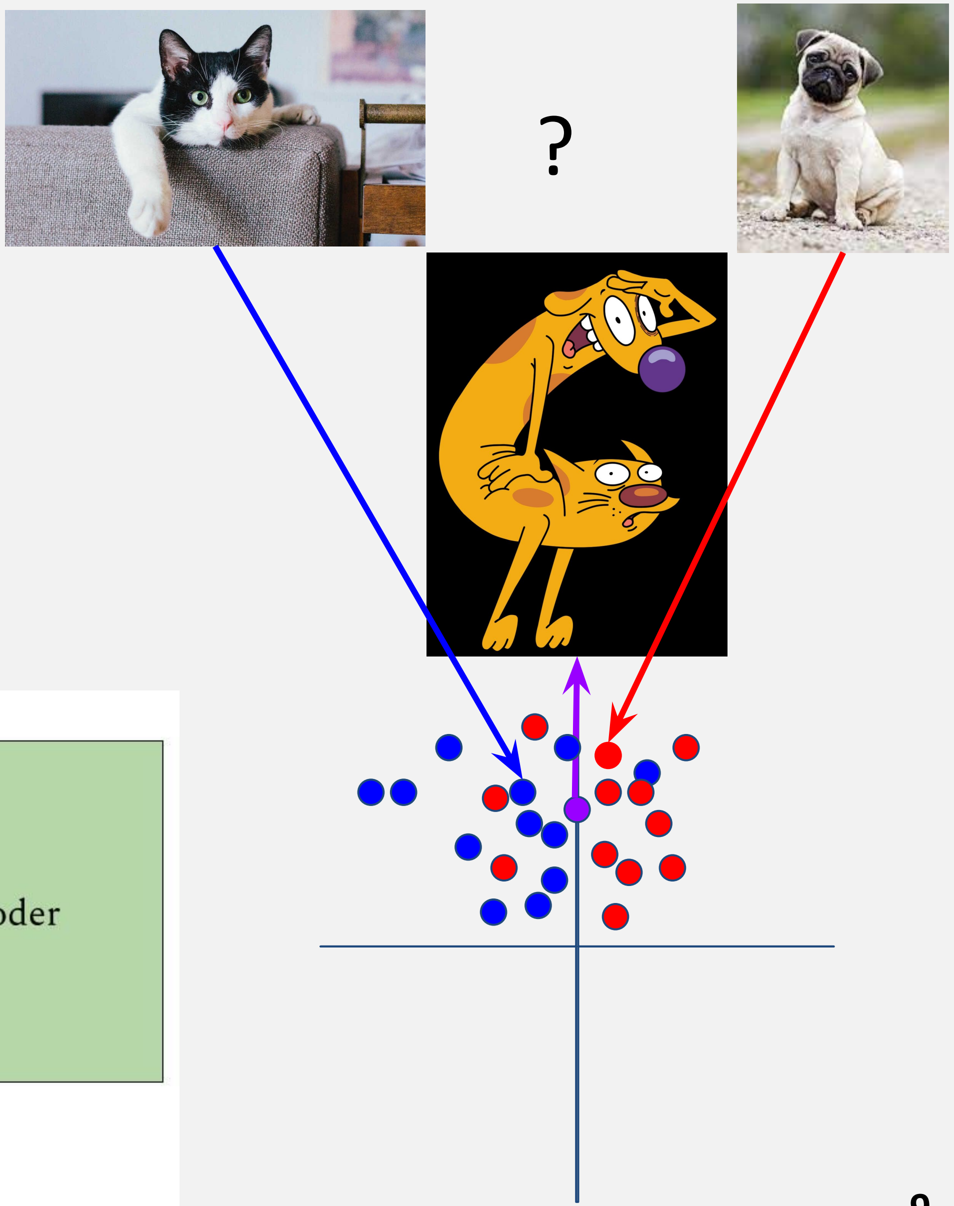
- **Why? → Generation**
  - **Imagine a cat (x) and a dog (y) are encoded in 2D to (-1,1) and (1, 1) respectively.**
  - **What will the decoder do if we tell it to read (0,1)?**

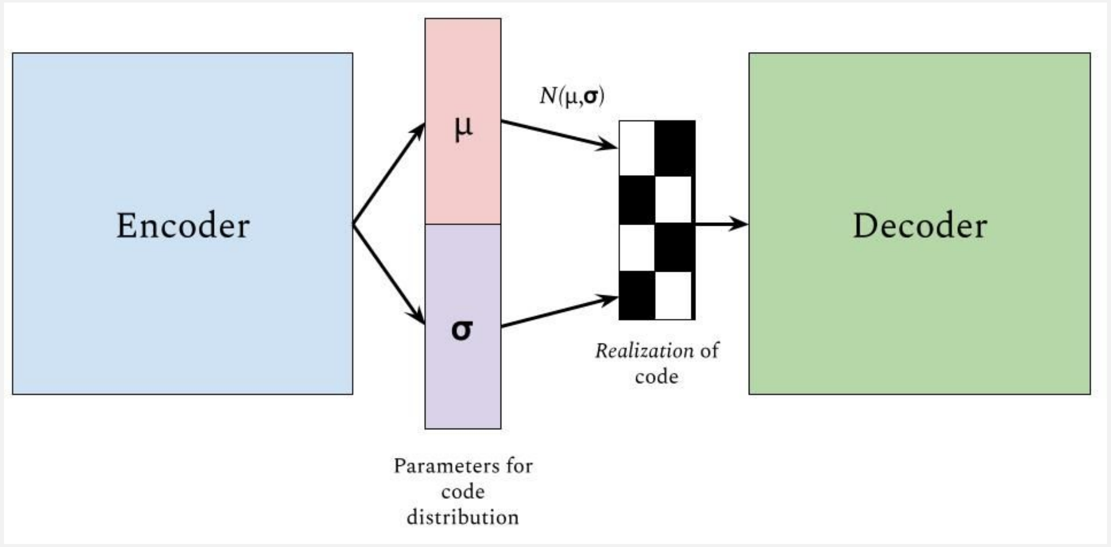




What is a Variational AutoEncoder?

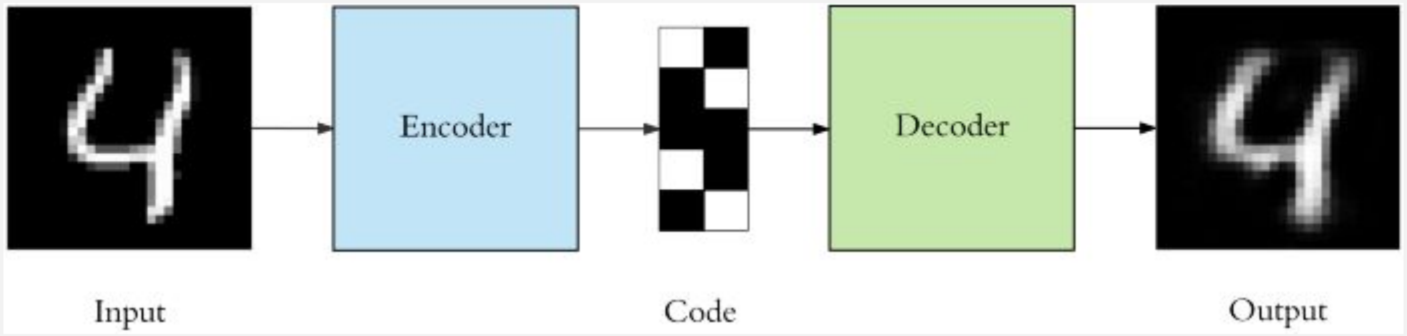
- **Why? → Generation**
  - **If the same image generates many codes we cover more of the space. Therefore. decoding random points in the space will mean something.**





VAE: How does it work

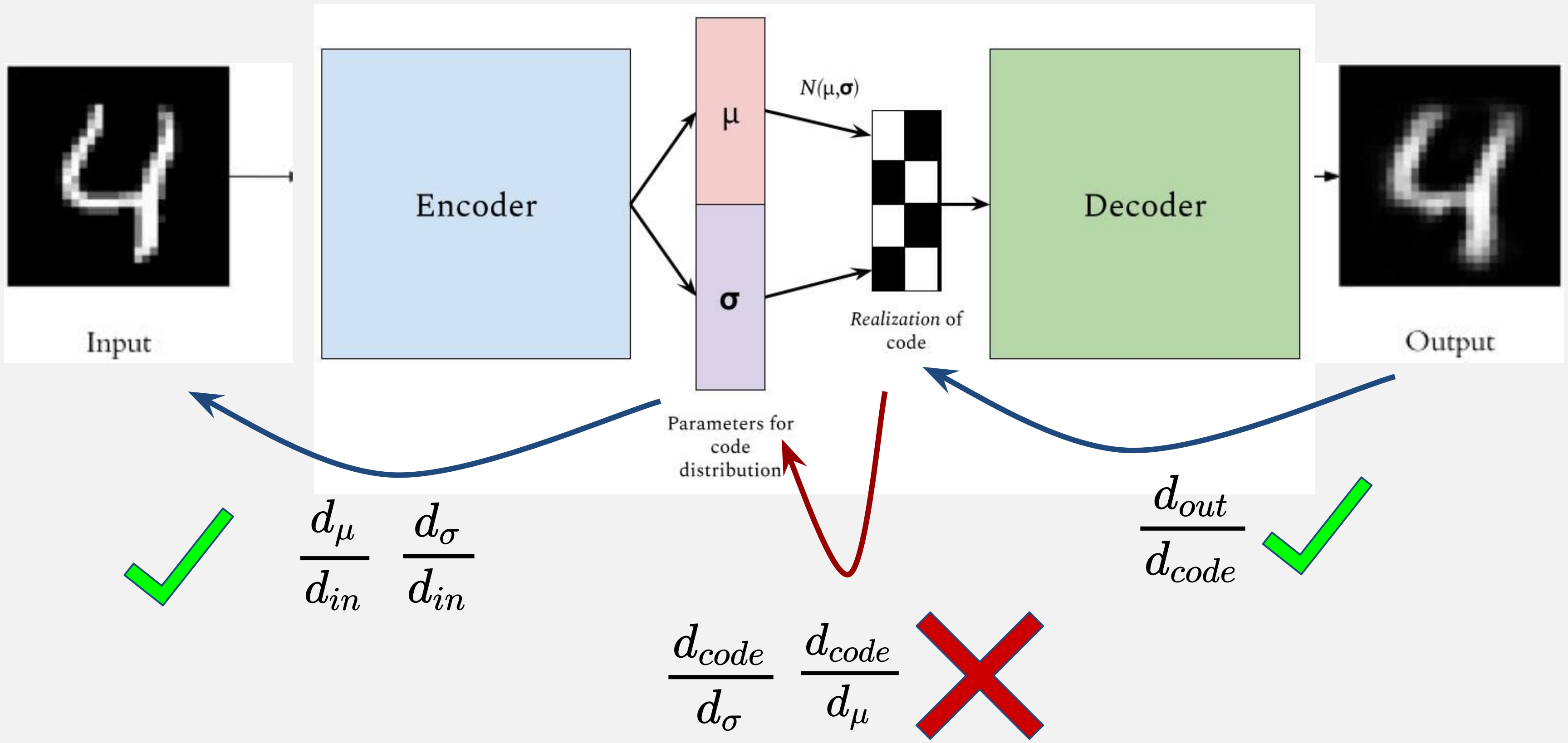
- **Reparameterization Trick**
  - To train, we need gradients
  - In regular AE, we just take those directly from weights/activations as normal



$$\begin{tikzcd}
\frac{d_{code}}{d_{in}} \arrow[dr] & & \frac{d_{out}}{d_{code}} \arrow[dl] \\
& \frac{d_{out}}{d_{in}} &
\end{tikzcd}$$

VAE: How does it work

- **Reparameterization Trick**
  - To train, we need gradients
  - In VAE, the code is generated randomly



$\frac{d_\mu}{d_{in}}$ $\frac{d_\sigma}{d_{in}}$ ✓

$\frac{d_{code}}{d_\sigma}$ $\frac{d_{code}}{d_\mu}$ ✗

$\frac{d_{out}}{d_{code}}$ ✓

VAE: How does it work

- **Reparameterization Trick**
  - To train, we need gradients
  - In VAE, the code is generated randomly
  - We cannot get $\frac{d_{code}}{d_\sigma}$ $\frac{d_{code}}{d_\mu}$ if $N(\mu, \sigma) \longrightarrow code$!
  - Instead, the blog lists $code = \mu + \sigma \odot \epsilon$
    - Think about how this allows the gradients to come through
    - Key: what is epsilon?
      - What is it generated from?
      - Why does that make the above equivalent to $N(\mu, \sigma)$?

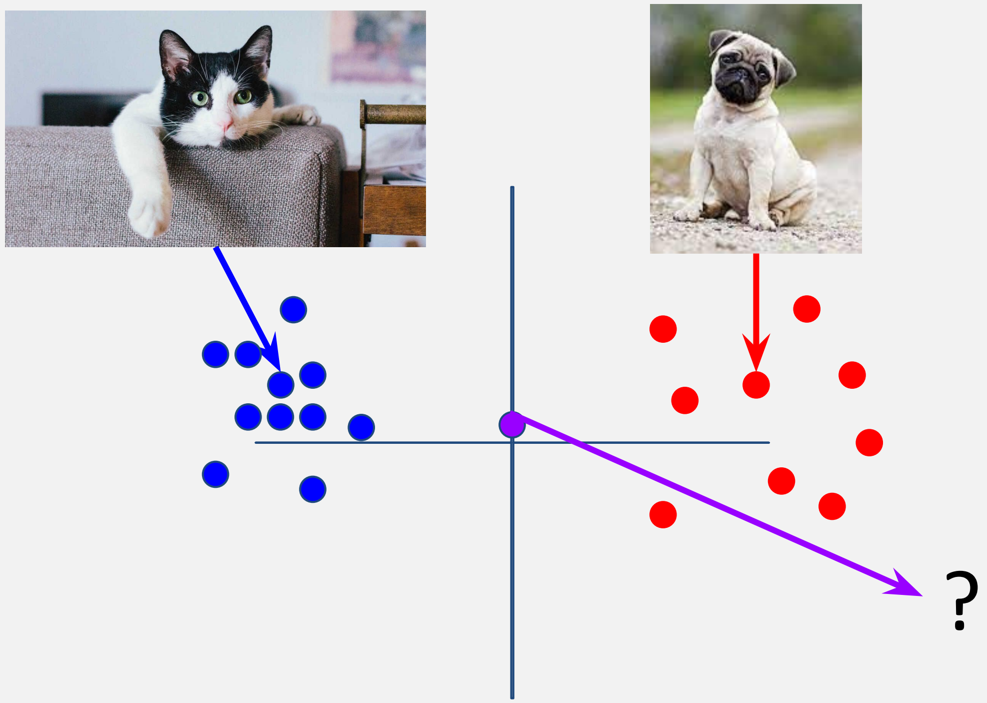
VAE: How does it work?

- **Loss**
  - **2 Objectives**
    - **AE objective - reconstruction loss**
    - **Ensure good generation - KL divergence**
      - **KL measures ‘distance’ between probability distributions**

$$Loss(x) = \underbrace{l_{recon}(f(x), x)}_{\text{Make the recovered image look right}} + \underbrace{\sum l_{KL}\left(q(z||x) = N(\mu_i, \sigma_i), N(0,1)\right)}_{\text{Keep the overall code distribution to be like N(0,1)}}$$

VAE: How does it work?

- Loss
  - 2 Objectives
    - AE objective - reconstruction loss
    - Ensure good generation - KL divergence
      - KL measures ‘distance’ between probability distributions

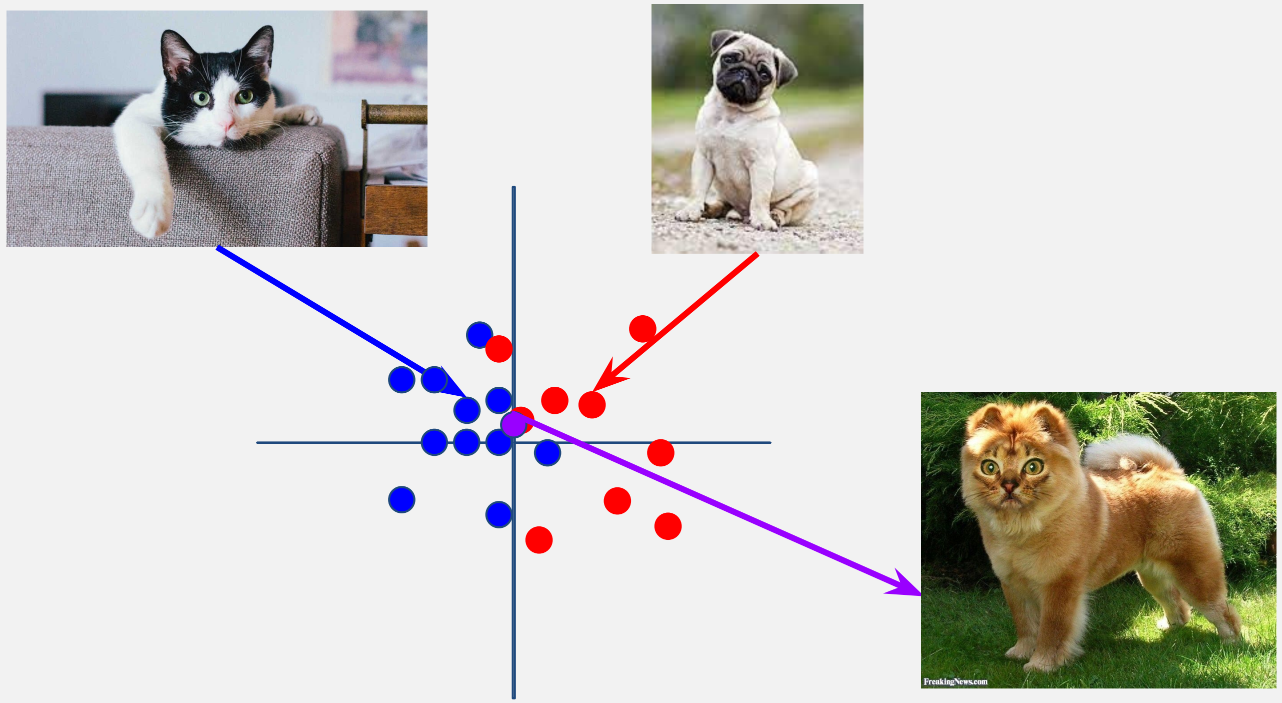


The second objective can be visualized like so.

On the left, without KL divergence we have the original AE problem where a random new code is not similar to what we’ve seen before

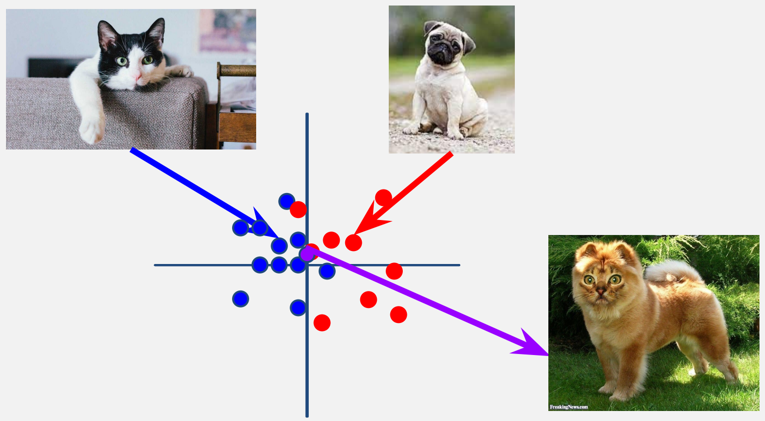
VAE: How does it work?

- **Loss**
  - **2 Objectives**
    - **AE objective - reconstruction loss**
    - **Ensure good generation - KL divergence**
      - **KL measures ‘distance’ between probability distributions**
  - **Do you foresee any problems with this loss ?**



VAE: How does it work?

- **Loss**
  - **2 Objectives**
    - **AE objective - reconstruction loss**
    - **Ensure good generation - KL divergence**
      - **KL measures ‘distance’ between probability distributions**
  - **Do you foresee any problems with this loss ?**
    - **@ GRADS or anyone doing the extra questions**
    - **If you are stuck trying to explain MMD loss, trying sketching a version of this on a 2D plane for what should happen with MMD instead.**

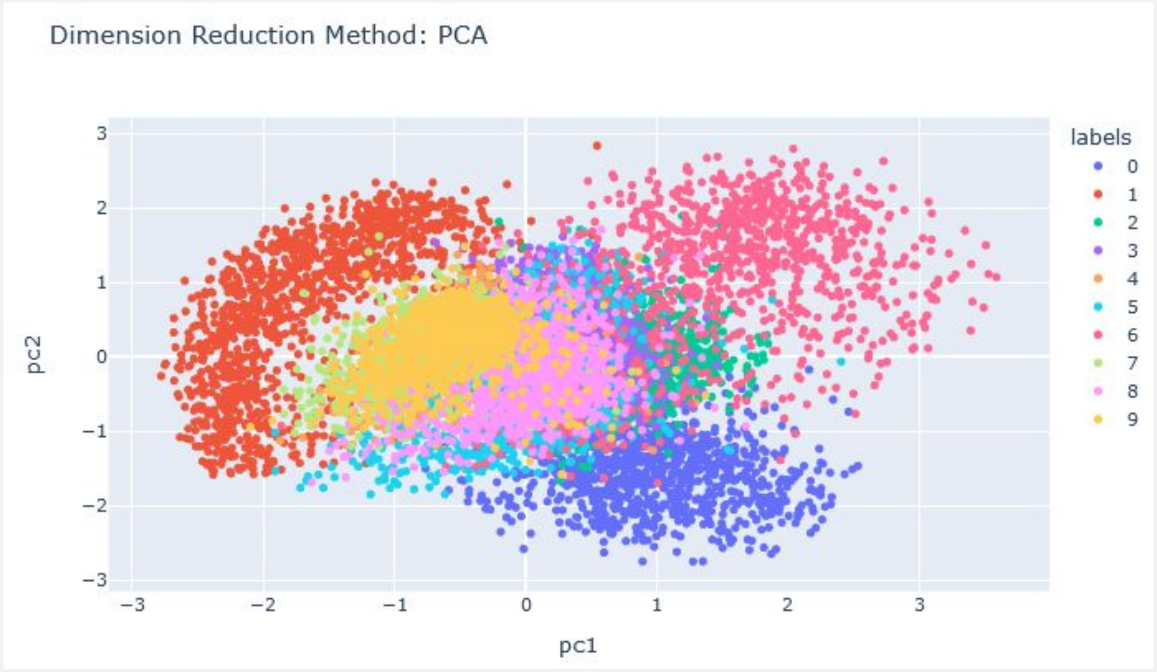


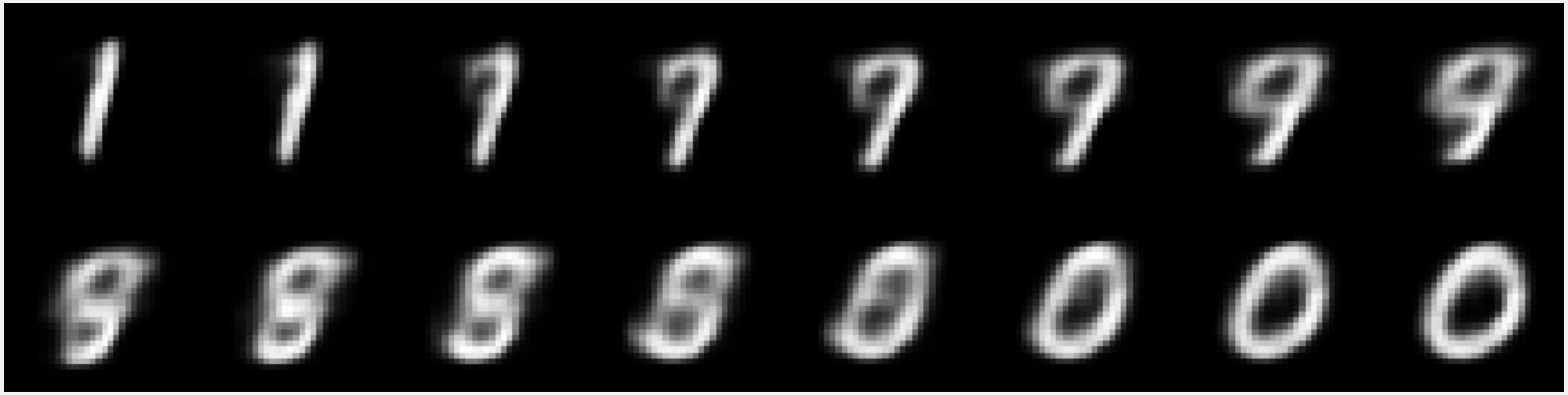
Visualizing

There are two visualizations we want to see.

1. The latent space with all the samples

2. Morphing from one class to another



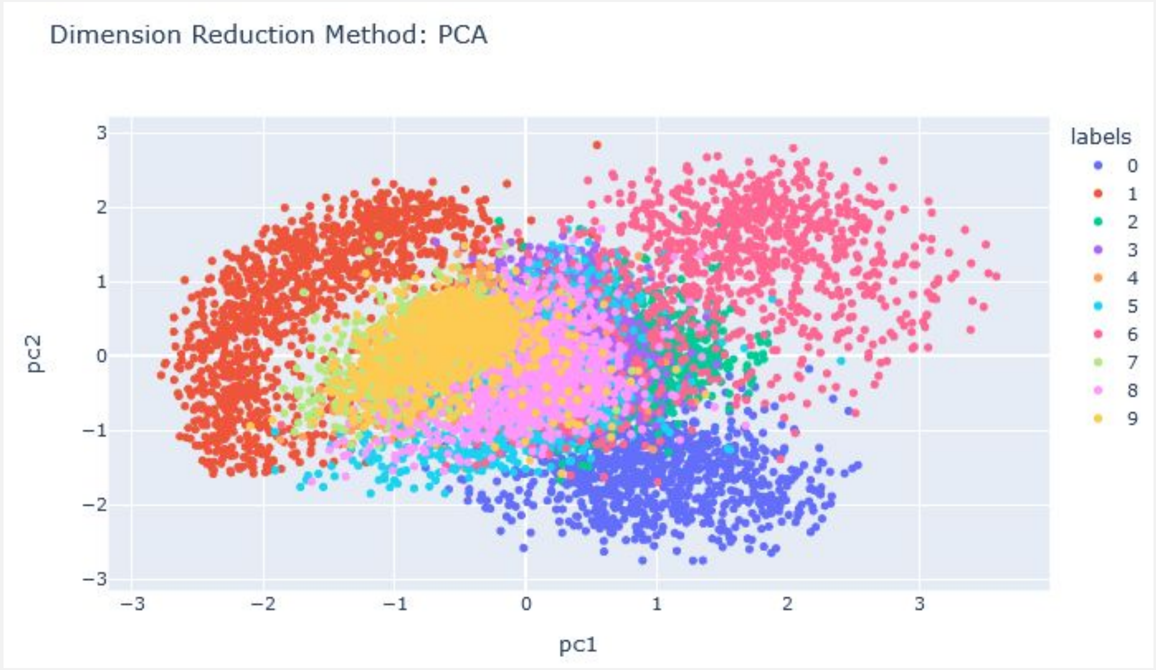


Visualizing

There are two visualizations we want to see.

1. The latent space with all the samples

2. Morphing from one class to another



1. Find the codes of all samples, then do PCA on them
2. Plot by class

This will indicate how good the division is between classes.
To numerically verify, one could train a small dense network to classify by the code.

Visualizing

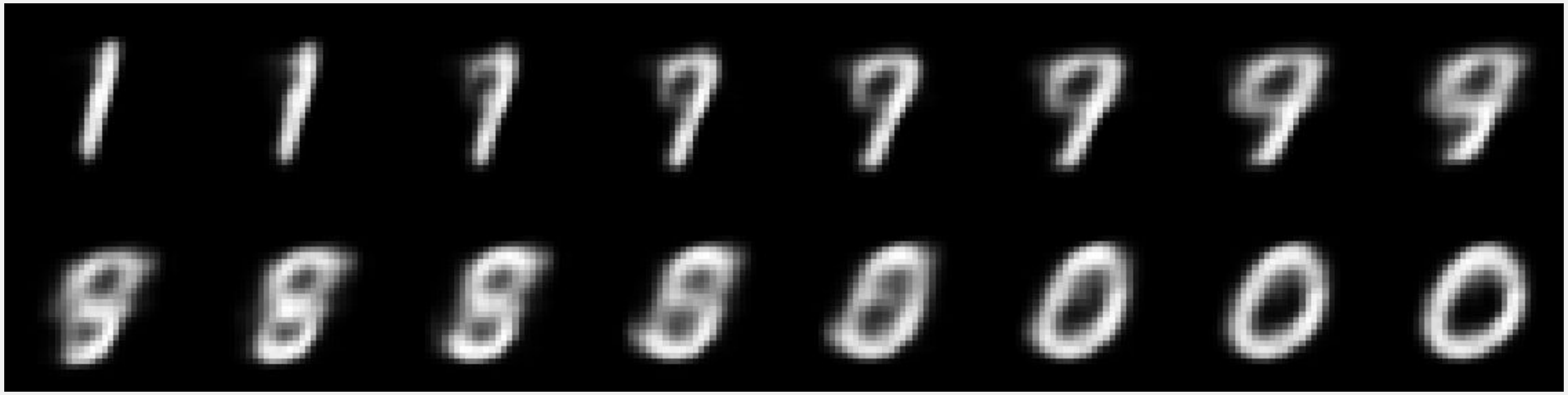
There are two visualizations we want to see.

1. The latent space with all the samples

2. Morphing from one class to another

1. Find the average of each classes' codes
2. Use interpolation (np.linespace) to walk from one to the other

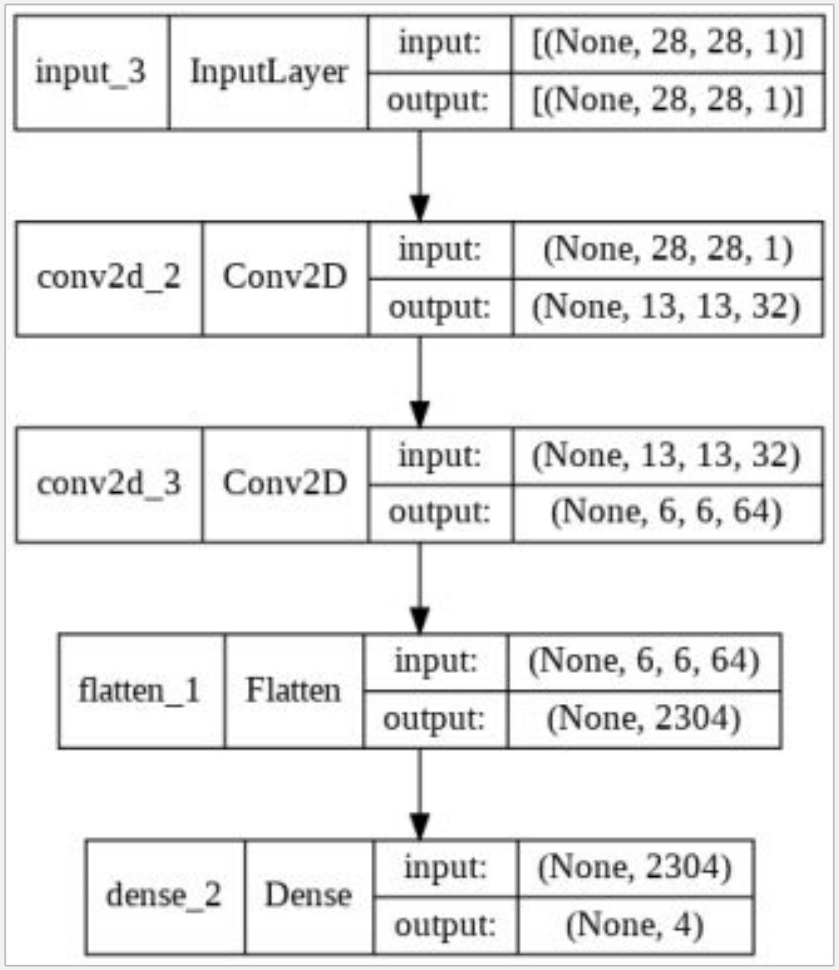
This will indicate that the latent space is well populated. If it is not, (for example without KL divergence) then we would expect unnatural morphing



Choosing Layers for the Decoder

`tf.keras.utils.plot_model`

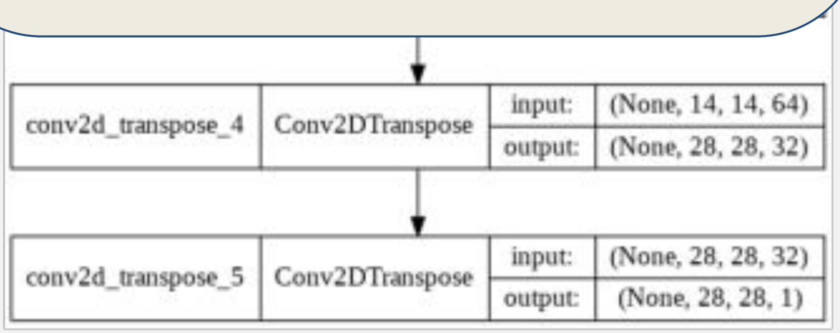
MNIST ENCODER



input_3	InputLayer	input:	[(None, 28, 28, 1)]
		output:	[(None, 28, 28, 1)]
conv2d_2	Conv2D	input:	(None, 28, 28, 1)
		output:	(None, 13, 13, 32)
conv2d_3	Conv2D	input:	(None, 13, 13, 32)
		output:	(None, 6, 6, 64)
flatten_1	Flatten	input:	(None, 6, 6, 64)
		output:	(None, 2304)
dense_2	Dense	input:	(None, 2304)
		output:	(None, 4)

Can you make sense of how the transpose convolution layers change the size?

Start with just 1 convolution in encoder and decoder - figure out what it takes to match the final output image size to the original



conv2d_transpose_4	Conv2DTranspose	input:	(None, 14, 14, 64)
		output:	(None, 28, 28, 32)
conv2d_transpose_5	Conv2DTranspose	input:	(None, 28, 28, 32)
		output:	(None, 28, 28, 1)

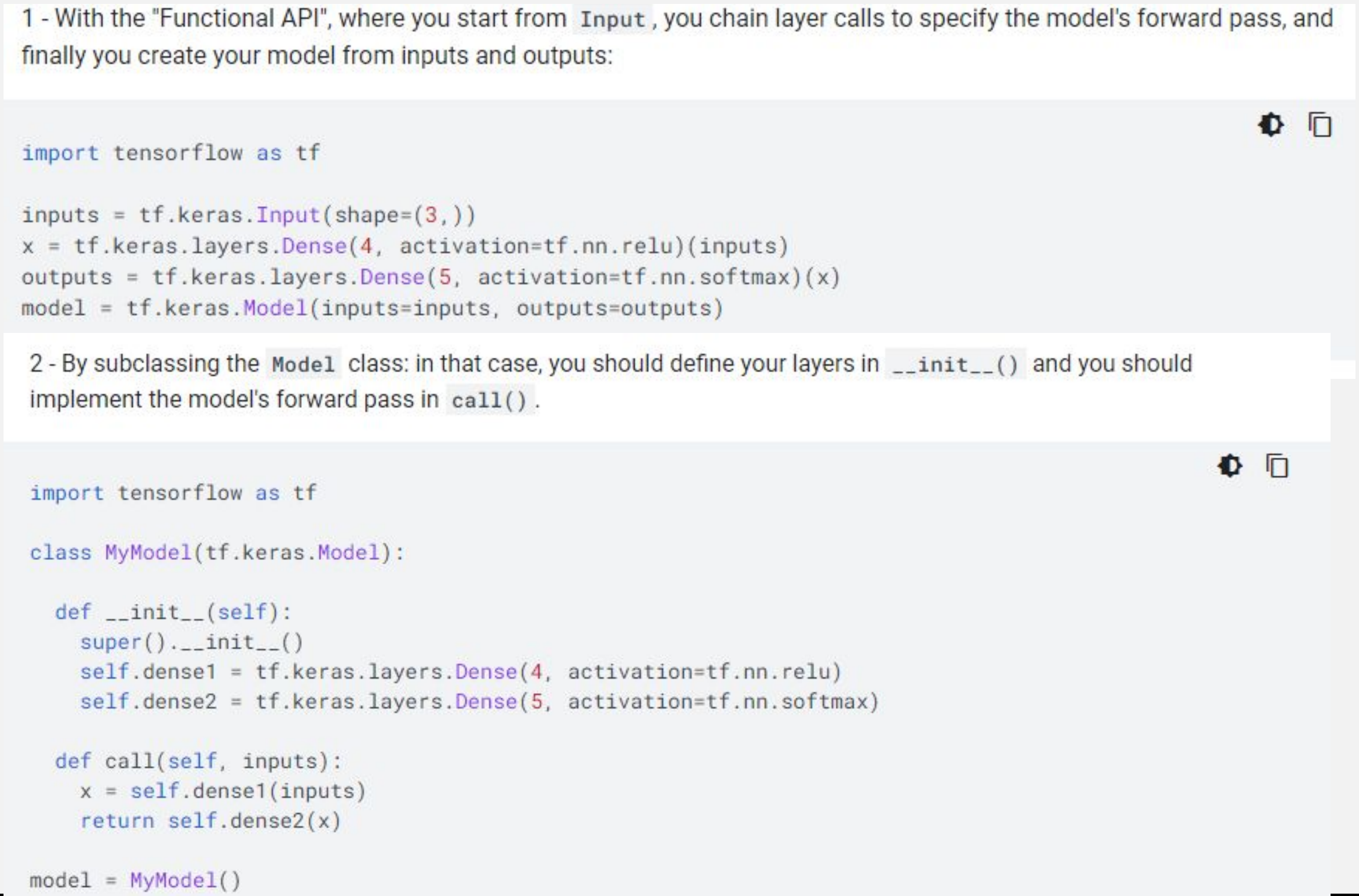
Time for…

Questions?

(Before moving onto coding examples)

Tensorflow Model

- Model: groups layers into an object with training and inference features
- There are two ways to instantiate a Model
- https://www.tensorflow.org/api_docs/python/tf/keras/Model



1 - With the "Functional API", where you start from `Input`, you chain layer calls to specify the model's forward pass, and finally you create your model from inputs and outputs:

```
import tensorflow as tf

inputs = tf.keras.Input(shape=(3,))
x = tf.keras.layers.Dense(4, activation=tf.nn.relu)(inputs)
outputs = tf.keras.layers.Dense(5, activation=tf.nn.softmax)(x)
model = tf.keras.Model(inputs=inputs, outputs=outputs)
```

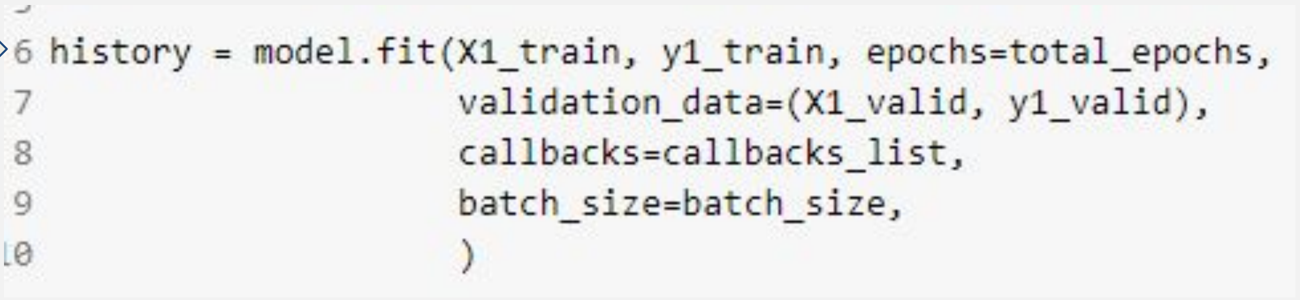
2 - By subclassing the `Model` class: in that case, you should define your layers in `__init__()` and you should implement the model's forward pass in `call()`.

```
import tensorflow as tf

class MyModel(tf.keras.Model):

  def __init__(self):
    super().__init__()
    self.dense1 = tf.keras.layers.Dense(4, activation=tf.nn.relu)
    self.dense2 = tf.keras.layers.Dense(5, activation=tf.nn.softmax)

  def call(self, inputs):
    x = self.dense1(inputs)
    return self.dense2(x)

model = MyModel()
```

Tensorflow GradientTape

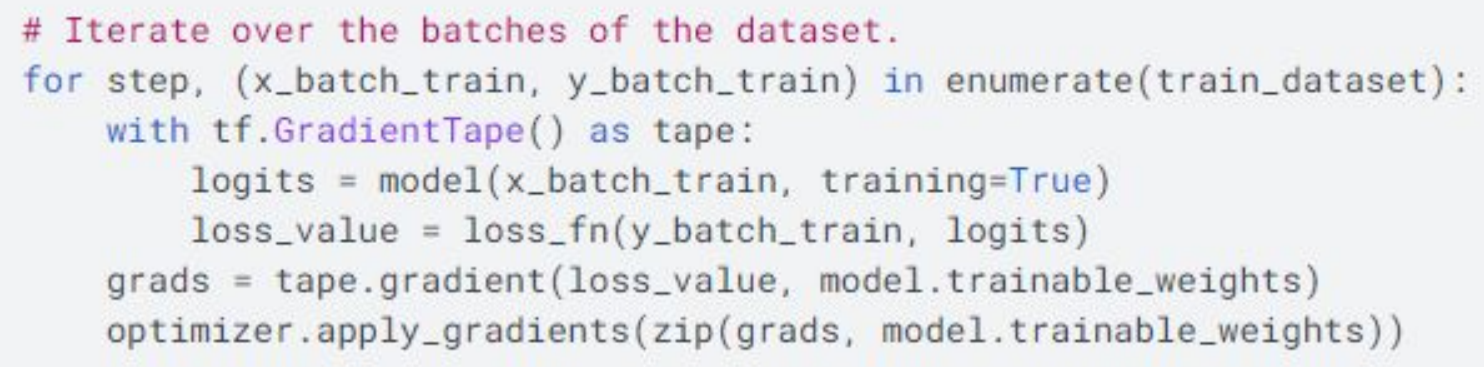
- **GradientTape records operations for automatic differentiation**
- **https://www.tensorflow.org/api_docs/python/tf/GradientTape**

HW1: we just call 'fit' to train the model



```
history = model.fit(X1_train, y1_train, epochs=total_epochs,
                    validation_data=(X1_valid, y1_valid),
                    callbacks=callbacks_list,
                    batch_size=batch_size,
                    )
```

We can also write a customized train loop



```
# Iterate over the batches of the dataset.
for step, (x_batch_train, y_batch_train) in enumerate(train_dataset):
    with tf.GradientTape() as tape:
        logits = model(x_batch_train, training=True)
        loss_value = loss_fn(y_batch_train, logits)
    grads = tape.gradient(loss_value, model.trainable_weights)
    optimizer.apply_gradients(zip(grads, model.trainable_weights))
```

Above example code from
https://www.tensorflow.org/guide/keras/writing_a_training_loop_from_scratch

What is CNN?

- Convolutional Neural Networks (CNN)
  - convolutional layer looks at small subset/local information. (Does this sound like a kernel?)
  - Convolution layer uses kernels to extract features
  - It tries to learn the kernel values in the training through backpropagation
  - Convolution layer parameters:
    - Filter: number of filters used to extract features
    - Kernel size: the size of the kernel
    - Strides: number of pixels to shift after every kernel computation
    - Padding:
      - "valid" padding: convolution operation is performed within the image boundary. Thus, resulting feature map is smaller than original image
      - "same" padding: convolution operation is performed such that the resulting feature map is the same size of the original image