Boring but important disclaimers:

- If you are not getting this from the GitHub repository or the associated Canvas page (e.g. CourseHero, Chegg etc.), you are probably getting the substandard version of these slides Don't pay money for those, because you can get the most updated version for free at

  `https://github.com/julianmak/OCES5303_ML_ocean`

  The repository principally contains the compiled products rather than the source for size reasons.
- Associated Python code (as Jupyter notebooks mostly) will be held on the same repository. The source data however might be big, so I am going to be naughty and possibly just refer you to where you might get the data if that is the case (e.g. JRA-55 data). I know I should make properly reproducible binders etc., but I didn't...
- I do not claim the compiled products and/or code are completely mistake free (e.g. I know I don't write Pythonic code). Use the material however you like, but use it at your own risk.
- As said on the repository, I have tried to honestly use content that is self made, open source or explicitly open for fair use, and citations should be there. If however you are the copyright holder and you want the material taken down, please flag up the issue accordingly and I will happily try and swap out the relevant material.

OCES 5303 :
ML methods in Ocean Sciences

Session 6: Autoencoders

Outline

- auto-encoders

  $\rightarrow$ latent space representation

  $\rightarrow$ de-noising example

- convolution auto-encoders

  $\rightarrow$ as above

Figure: That time when me as editor assigned myself as referee: an **auto**-assign.

Oceanic application

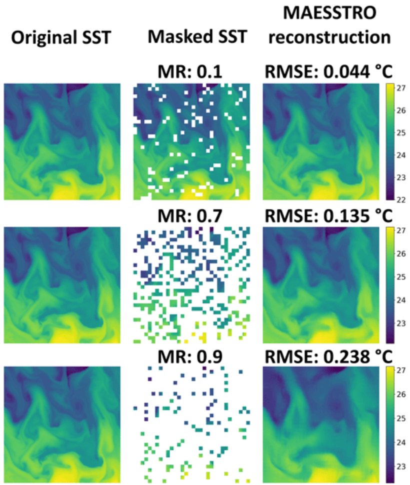

Figure: Reconstruction of SST from occluded data. From Fig. 2 of Goh *et al.* (2024).

- e.g. satellite data

  → masked auto-encoder to fill out the gaps

  → model data but masked accordingly (e.g. cover by cloud)

  → don't have to wait for the satellite to pass again

  → could envisage this being applied to other sparse data (e.g. oxygen)

Auto-encoders

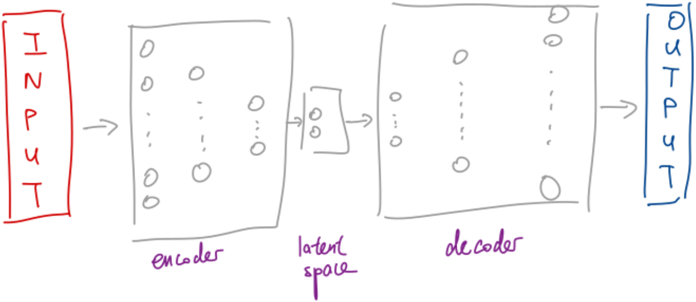

Figure: Schematic of an auto-encoder.

- "unsupervised" neural network
  $\rightarrow$ encoder part to compress data as latent space representation
  $\rightarrow$ decoder part to decompress data

Auto-encoders: penguins

- penguins data

  $\rightarrow$ encoder takes four feature dim to two

  $\rightarrow$ decoder takes two latent space dim to four

  $\rightarrow$ input and outputs are exactly the same (hence **auto**-encoder)

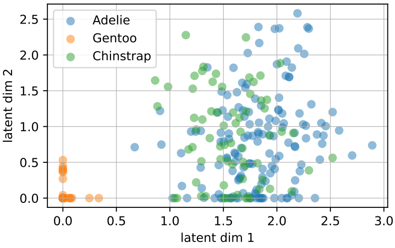

Figure: Latent space representation of penguin data.

- call the encoder only to get latent space representation

  $\rightarrow$ bit like getting the PCs

  $\rightarrow$ alternative way to do dimension reduction

Auto-encoders: cats

- do the same with the cat images (sigmoid activation, MinMax scaling)
  $\rightarrow 64^2 \rightarrow 16^2 \rightarrow 4^2$ and reverse

```python
def ae_catdog():

    # 1) define the autoencoder and the layers
    inputs = keras.Input(shape=(64**2,))
    encoded = keras.Sequential(
        [
            layers.Dense(16**2, activation='relu'),  # no input here
            layers.Dense(4**2, activation='relu'),
        ]
    )(inputs)  # input to be passed here
    decoded = keras.Sequential(
        [
            layers.Dense(16**2, activation='relu'),
            layers.Dense(64**2, activation='sigmoid'),
        ]
    )(encoded)
    autoencoder = keras.Model(inputs, decoded, name="autoencoder")
```

Figure: Keras implementation of basic autoencoder. Has about a million parameters to be trained.

Auto-encoders: cats

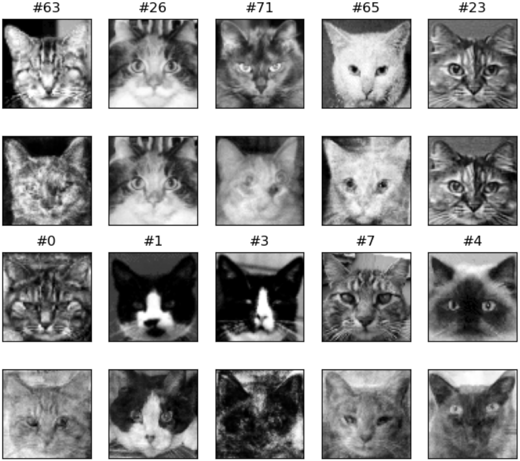

Figure: Train/test set reproduction from auto-encoder.

- note the imperfect reconstruction here even in train set

Auto-encoders: cats

- do the same with the cat images
  $\to$ encoder is $\to 64^2 \to 16^2 \to 4^2$
  $\to$ notice each cat has a few zero entries

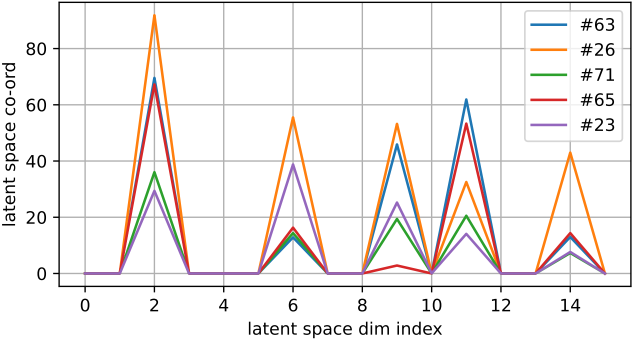

Figure: 16 dimensional latent space representation of the cats in the previous slide (cf. PCs of an EOF decomposition as we did in eigencat).

Auto-encoders: cursed cats

- could generate new cat images by calling the decoder

  $\rightarrow$ decoder is $4^2 \rightarrow 16^2 \rightarrow 64^2$

  $\rightarrow$ choose 16 numbers to feed into the decoder

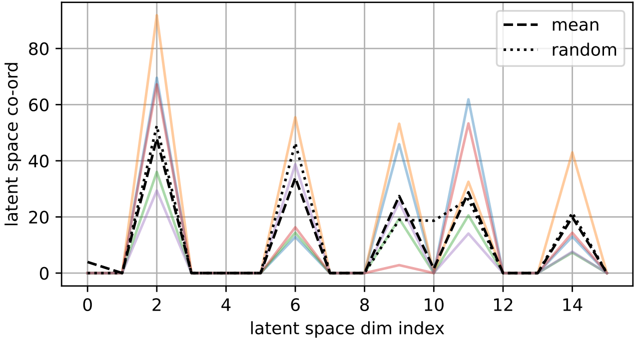

Figure: Two realisations in the 16 dimensional latent space representation to be fed into the decoder to generate new images (cf. choosing PCs for the EOFs and recombine to get image).

Auto-encoders: cursed cats

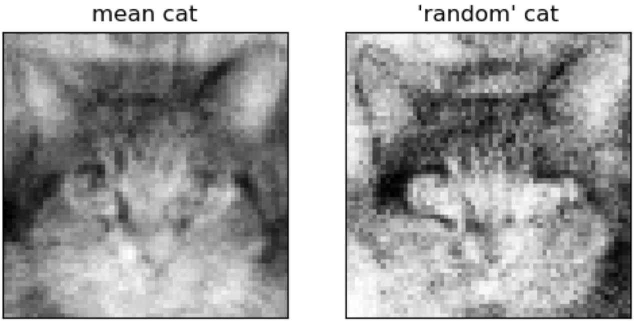

Figure: Slightly cursed regenerations.

- I never said they were going to be good reconstructions...
  $\rightarrow$ small dataset and auto-encoder shallow and thin

Auto-encoders: de-noised cats

- could use auto-encoder to de-noise data
  $\rightarrow$ training input is noisy data, training output is clean data
  $\rightarrow$ neural network to learn to remove noise

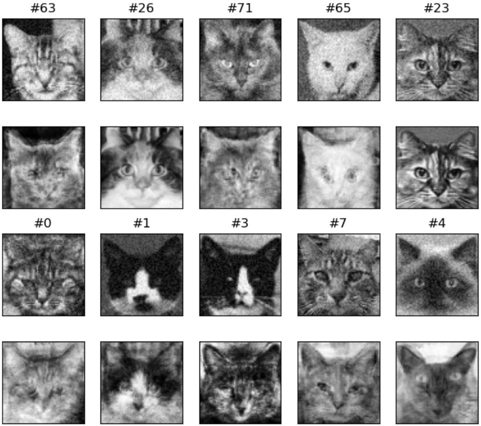

Figure: Noisy cats. Noise level is 0.2 standard deviations (images are standarised). Training set seems ok? Testing set is ehhhh...(sample size definitely too small here)

Auto-encoders: de-noised TAs

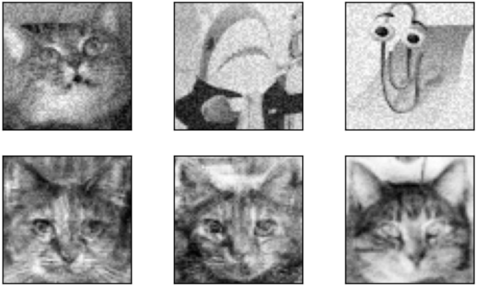

Figure: Just for fun out-of-sample application.

- use larger datasets (e.g. the big eigencat one from lec 02)
- deeper + wider neural network blocks for encoder/decoder

Convolutional Auto-encoders

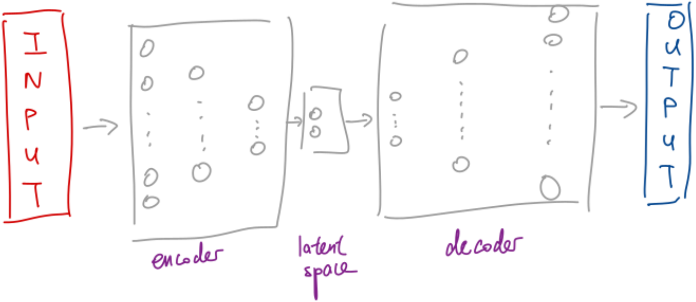

Figure: Schematic of an auto-encoder.

- illustrated is a MLP as a encoder/decoder, but no reason they can't be **convolution** layers

  $\rightarrow$ note encoder/decoder don't have to mirror each other either

Convolutional Auto-encoders

```python
def convae_catdog():

    # 1) define the autoencoder and the layers
    inputs = keras.Input(shape=(64, 64, 1))
    encoded = keras.Sequential(
        [
            layers.Conv2D(32, (3, 3), activation="relu", padding="same"),
            layers.MaxPooling2D((2, 2), padding="same"),
            layers.Conv2D(32, (3, 3), activation="relu", padding="same"),
            layers.MaxPooling2D((2, 2), padding="same"),
        ]
    )(inputs)  # input to be passed here
    decoded = keras.Sequential(
        [
            layers.Conv2DTranspose(32, (3, 3), strides=2, activation="relu", padding="same"
            layers.Conv2DTranspose(32, (3, 3), strides=2, activation="relu", padding="same"
            layers.Conv2D(1, (3, 3), activation="sigmoid", padding="same"),
        ]
    )(encoded)
    autoencoder = keras.Model(inputs, decoded, name="autoencoder")
```

Figure: Keras implementation of a convolutional autoencoder.

- don't strictly need to use `Conv2DTranspose`?
- note I used `stride=2` here

  → could use `upscale` to 'mirror' the `MaxPooling2D`

Convolutional Auto-encoders

```
Model: "autoencoder"
```

Layer (type)	Output Shape	Param #
input_layer_16 (InputLayer)	(None, 64, 64, 1)	0
sequential_6 (Sequential)	(None, 16, 16, 32)	9,568
sequential_7 (Sequential)	(None, 64, 64, 1)	18,785

```
Total params: 28,353 (110.75 KB)

Trainable params: 28,353 (110.75 KB)

Non-trainable params: 0 (0.00 B)
```

Figure: Summary of the convolutional autoencoder.

- note this has substantially fewer training parameters $\rightarrow$ 20,000 vs. 1,000,000 before

Convolutional Auto-encoders: de-noised cats

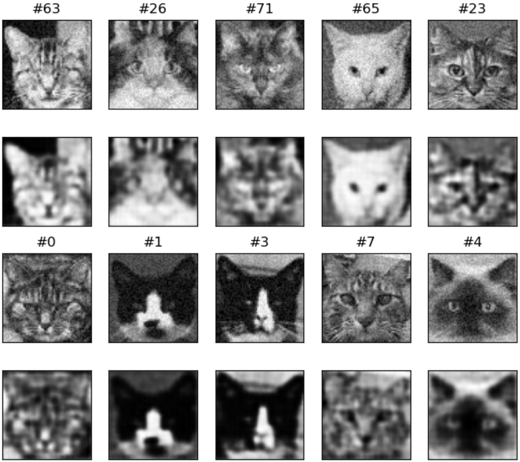

Figure: Noisy cats. Noise level is 0.2 standard deviations (images are standarised). Seem to at least get the **same** cats back, although very blurry

Convolutional Auto-encoders: de-noised TAs

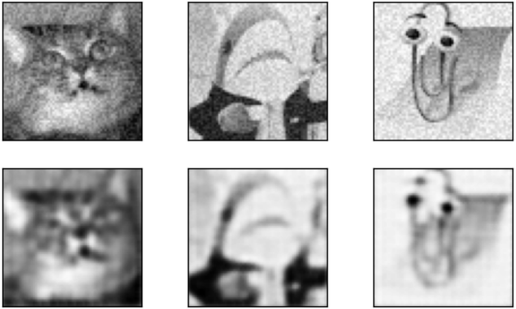

Figure: Just for fun out-of-sample application. Also getting the same images back at least.

Demonstration

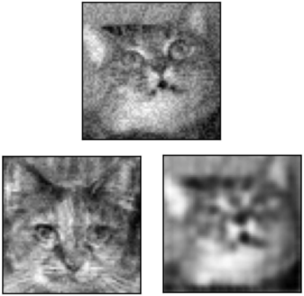

Figure: :◁

- auto-encoders

  $\rightarrow$ MLPs and convolutional variety

  $\rightarrow$ deeper + wider? data size?

  $\rightarrow$ **variational** variety?

- try it with some **satellite/simulation** data?

Moving to a Jupyter notebook $\rightarrow$