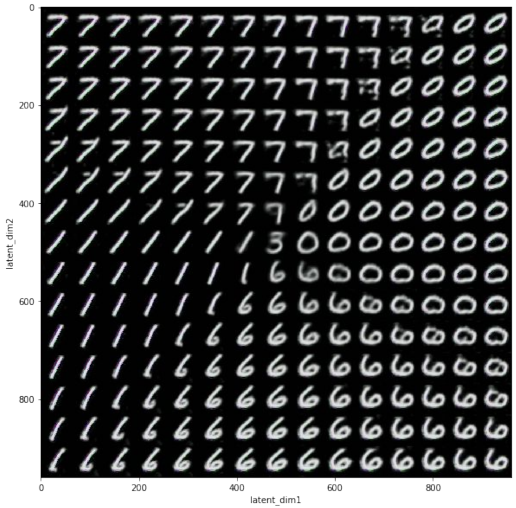
Convolutional Autoencoders for Image Manipulation

Pavlos Protopapas
Vincent Casser, Camilo Fosco

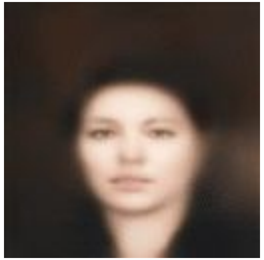
Institute for Applied Computational Science
Harvard

Structure

1. Build an Autoencoder for MNIST
2. Extend to Variational Autoencoder (VAE)
3. Work with real-world images (faces)



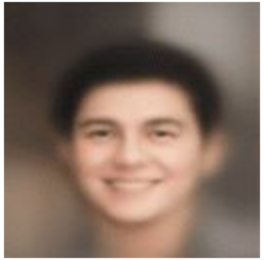
Smile



Gender

Bald

Beard



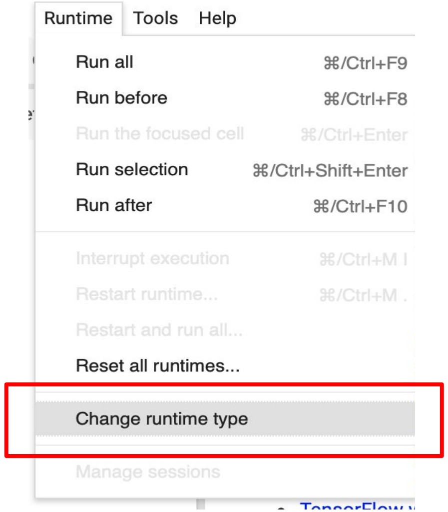
What you'll (hopefully) take away

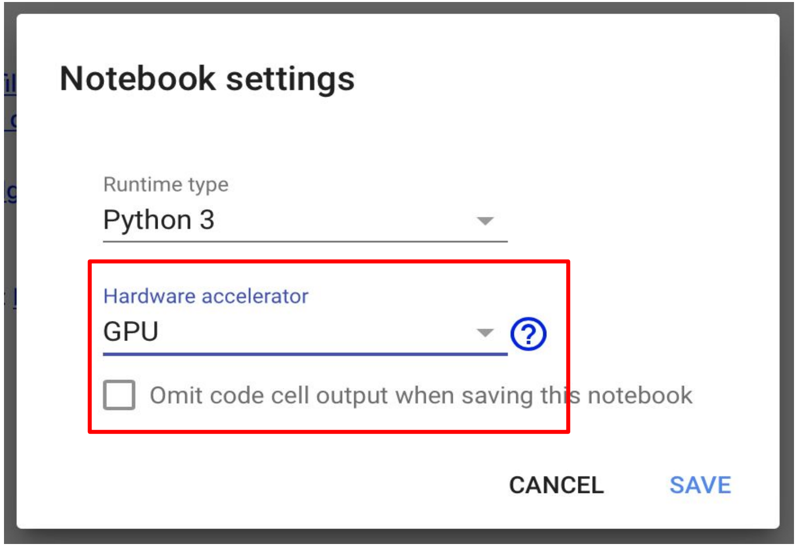
- How AE's and VAE's work
- How they are defined, trained and executed using keras
- How you could apply them to other vision tasks, such as
    - Denoising
    - Colorization
    - Segmentation
    - Completion
- Even if you can't follow the entire workshop, we will provide you with documented code that you can review later

[Colab] Start

Open: https://bit.ly/2FTFeif

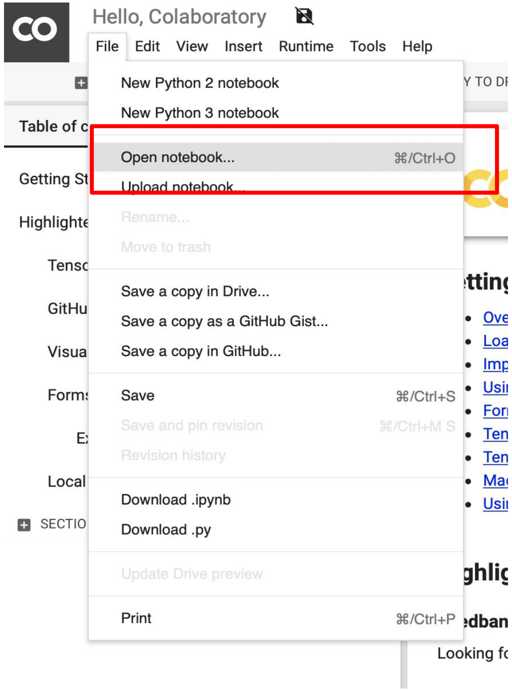
Click Runtime -> Change runtime type… put GPU!





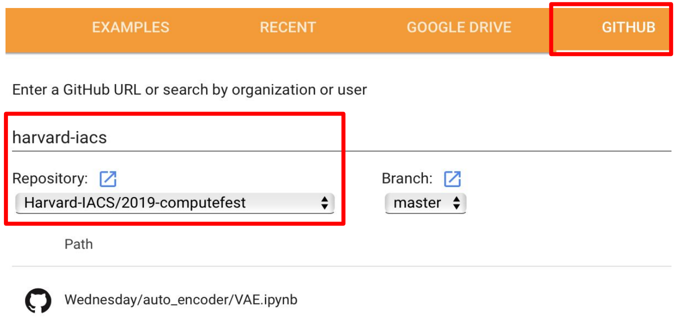
[Colab] Open: colab.research.google.com

1) File -> Open notebook...

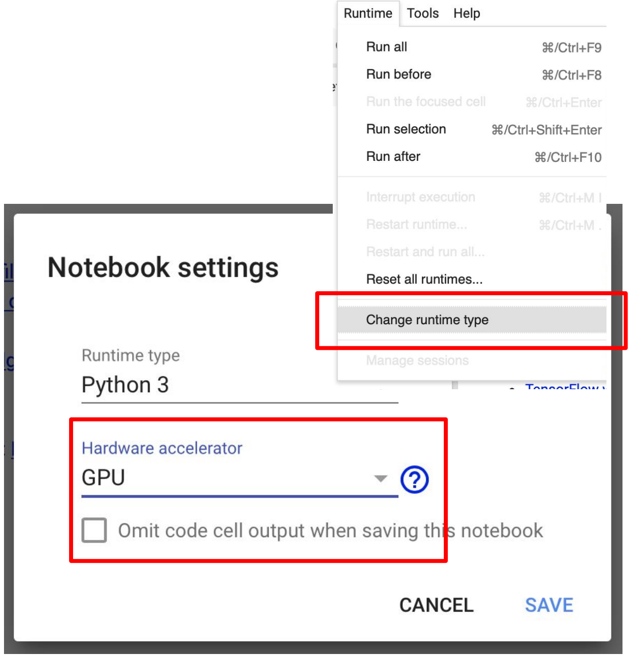


2) Github -> harvard-iacs -> Harvard-IACS/2019-computefest

Open notebook VAE.ipynb



3) Runtime -> Change runtime type... put GPU



[Colab] Run first 3 cells. Ignore security warnings

Warning: This notebook was not authored by Google.

This notebook is being loaded from GitHub. It may request access to your data stored with Google, or read data and credentials from other sessions. Please review the source code before executing this notebook. To prevent this notebook reading state from other sessions, you can reset all runtimes.

- [x] Reset all runtimes before running

CANCEL RUN ANYWAY

0. Download required code and data

```
[1]  !git clone https://github.com/Harvard-IACS/2019-computefest.git
```

```
Cloning into '2019-computefest'...
remote: Enumerating objects: 16, done.
remote: Counting objects: 100% (16/16), done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 192 (delta 4), reused 11 (delta 2), pack-reused 176
Receiving objects: 100% (192/192), 188.52 MiB | 30.22 MiB/s, done.
Resolving deltas: 100% (64/64), done.
Checking out files: 100% (29/29), done.
```

```
[2]  import os
     os.chdir("2019-computefest/Wednesday/auto_encoder")
```

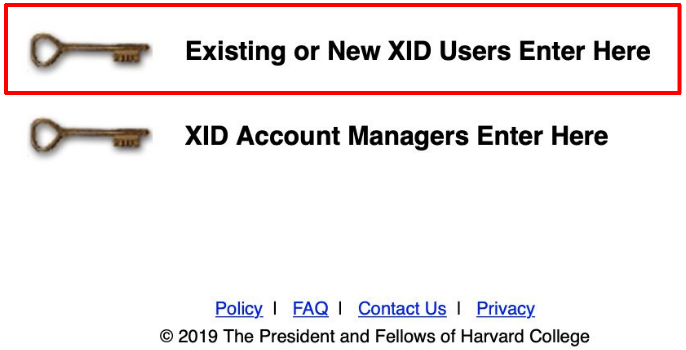
```
[3]  !ls
```

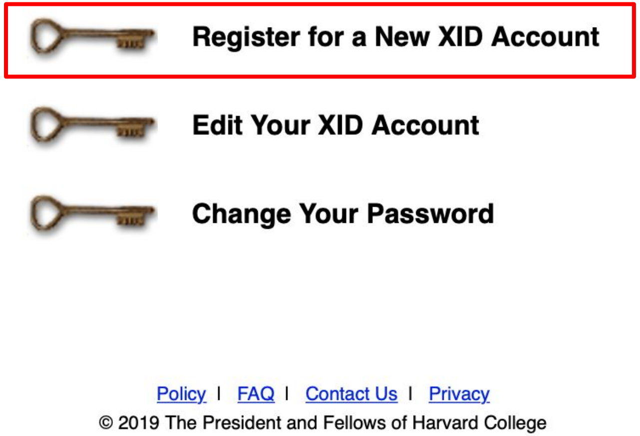
```
celeba   README.md          utils.py             VAE_Solutions.ipynb
models   requirements.txt   VAE_Attendee.ipynb
```

[AWS] If you haven't yet…

If you don't have a HarvardKey, please claim a XID key here:

https://xid.harvard.edu

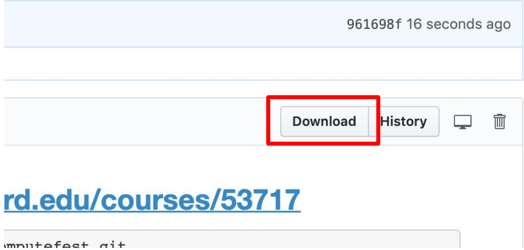




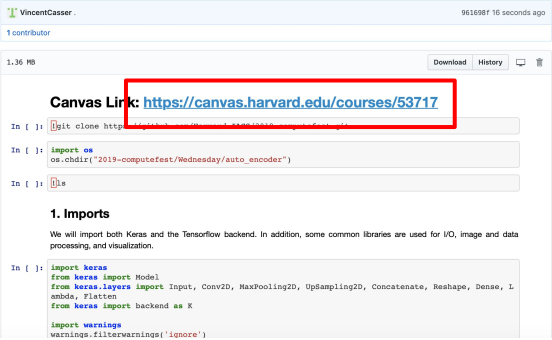
[AWS] Launch JupyterHub

Go to: https://bit.ly/2RKwVMC

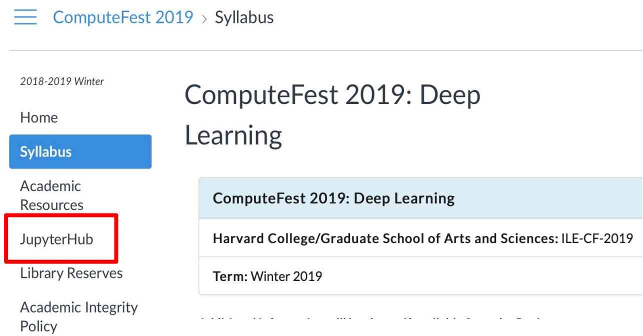
1) Download the Notebook



2) Open Canvas



3) Click “JupyterHub”



4) Upload VAE.ipynb

[Local] Work on your laptop

You can work directly on your laptop, but it **won’t be feasible to train**.

If you haven’t yet, clone the github repository:

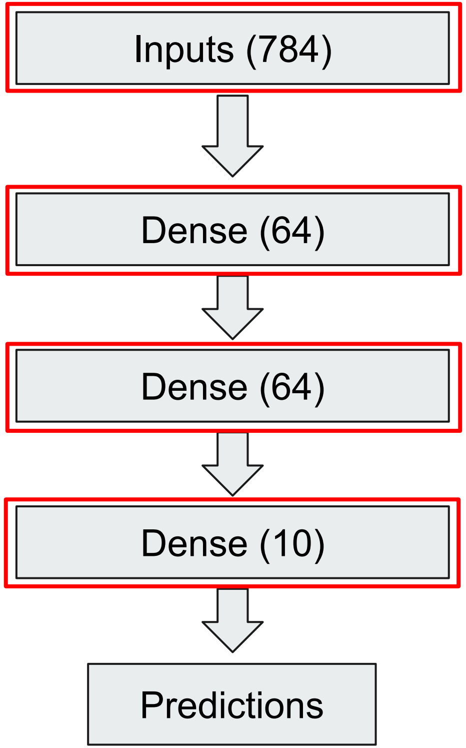
git clone git@github.com:Harvard-IACS/2019-computefest.git

Follow the instructions in README.md.

You can open the notebook “VAE.ipynb” in a jupyter notebook instance.

Introduction to Keras Functional API

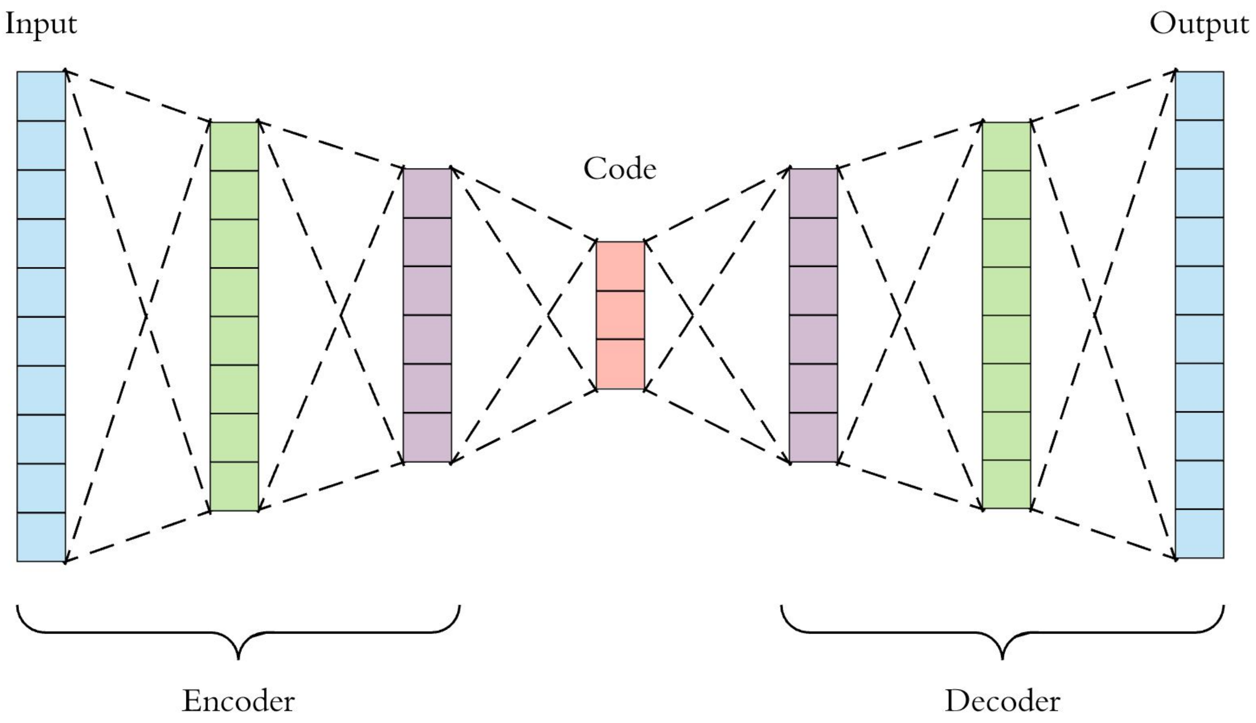
```python
from keras.layers import Input, Dense

# This returns a tensor
inputs = Input(shape=( 784,))

# a layer instance is callable on a tensor, and returns a tensor
x = Dense( 64, activation='relu')(inputs)
x = Dense( 64, activation='relu')(x)
predictions = Dense( 10, activation='softmax')(x)

# Creates model that includes the Input layer and 3 Dense layers
model = keras.models.Model(inputs=inputs, outputs=predictions)
model.compile(optimizer= 'rmsprop',
              loss= 'categorical_crossentropy' ,
              metrics=[ 'accuracy'])
```



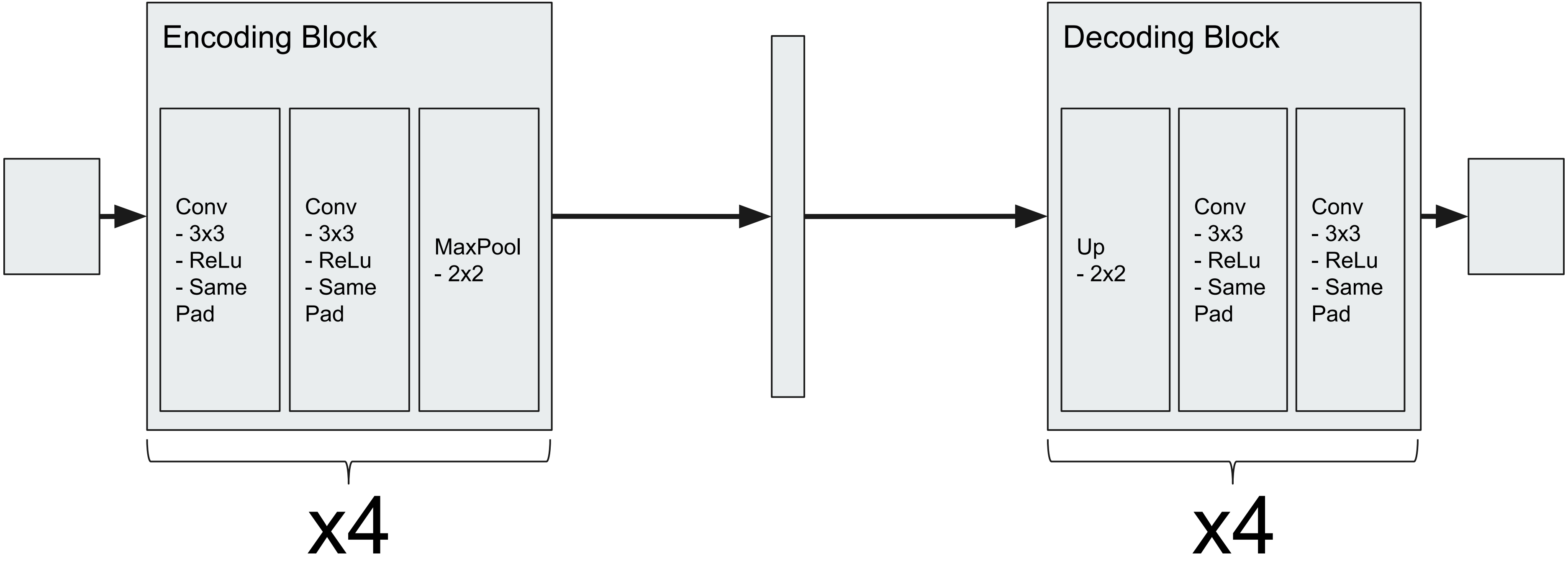
Layers are connected by referencing the previous layer at the end:

new_layer = keras.layers.Layer(arguments)(**prev_layer**)

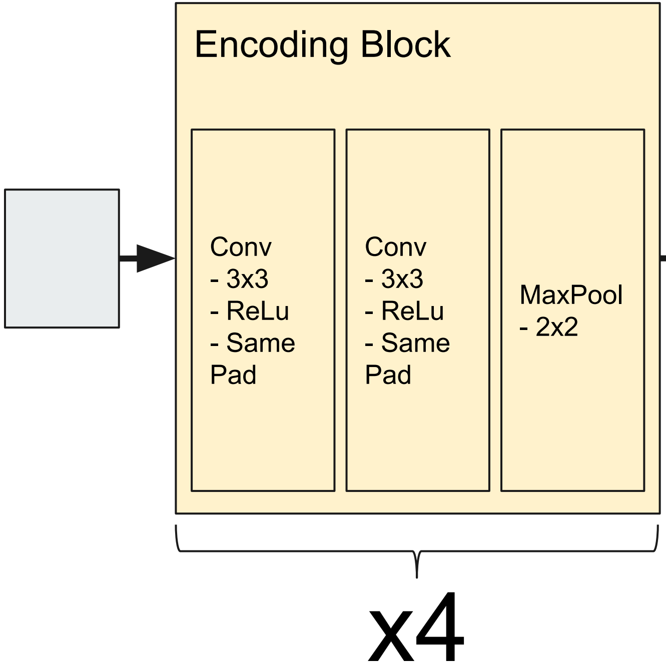
Autoencoder: Recap



Architecture: Autoencoder



Implement Encoding Block



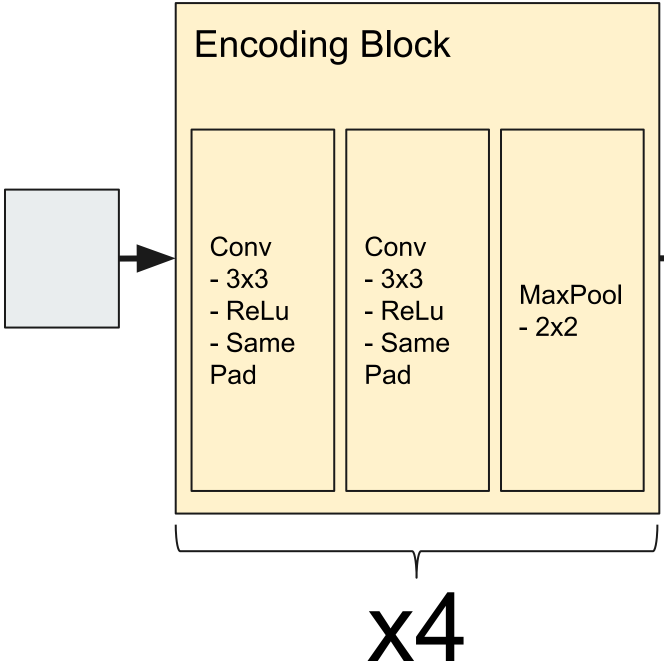
The relevant layers in keras are defined as:

keras.layers.Conv2D()
keras.layers.MaxPooling2D()

What you have to implement:

```
def define_encoder_block(x, num_filters):

    # define first convolutional layer with num_filters
    # define second convolutional layer with num_filters
    # define max pooling layer
    # return result of max pooling layer
```

Implement Encoding Block



The relevant layers in keras are defined as:

keras.layers.Conv2D()
keras.layers.MaxPooling2D()

Implementation could look like this:

```
def define_encoder_block(x, num_filters):
    x = Conv2D(num_filters, 3, activation='relu', padding='same',
                    kernel_initializer='he_normal')(x)
    x = Conv2D(num_filters, 3, activation='relu', padding='same',
                    kernel_initializer='he_normal')(x)
    x = MaxPooling2D()(x)
    return x
```

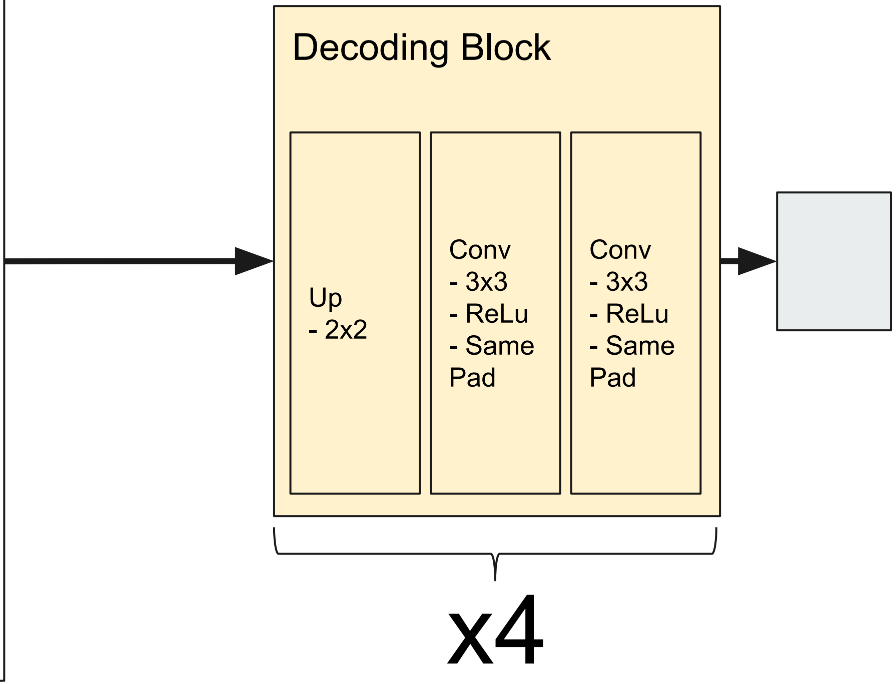
Implement Decoding Block

The relevant layers in keras are defined as:

```
keras.layers.UpSampling2D()
keras.layers.Conv2D()
```

What you have to implement:

```
def define_decoder_block(x, num_filters):

    # define upsampling layer
    # define first convolutional layer with num_filters
    # define second convolutional layer with num_filters
    # return result of second convolutional layer
```



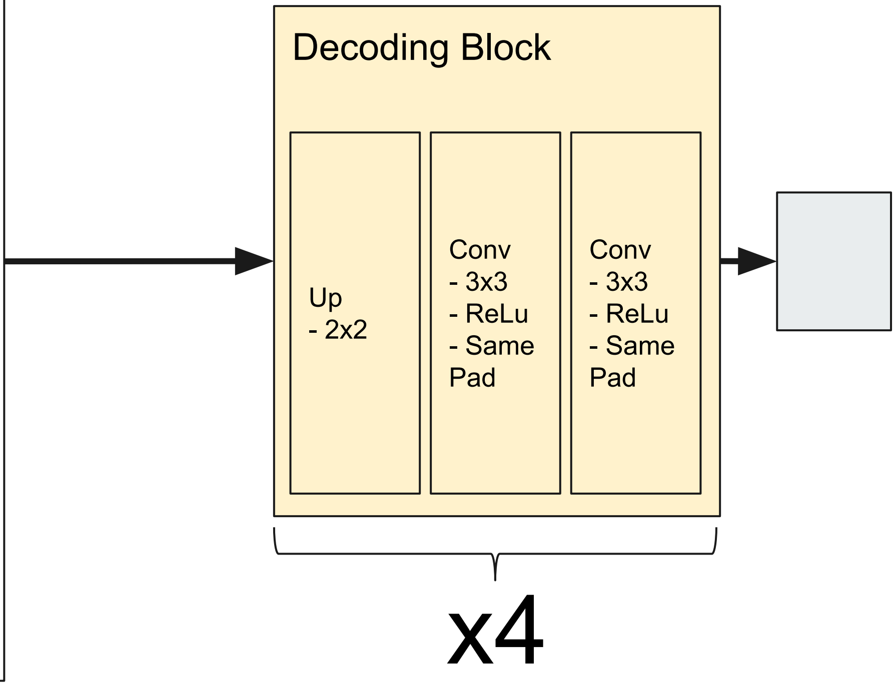
Implement Decoding Block

The relevant layers in keras are defined as:

keras.layers.UpSampling2D()
keras.layers.Conv2D()

Implementation could look like this:

```
def define_decoder_block(x, num_filters):
    x = UpSampling2D()(x)
    x = Conv2D(num_filters, 3, activation='relu', padding = 'same',
                    kernel_initializer = 'he_normal')(x)
    x = Conv2D(num_filters, 3, activation='relu', padding = 'same',
                    kernel_initializer = 'he_normal')(x)
    return x
```



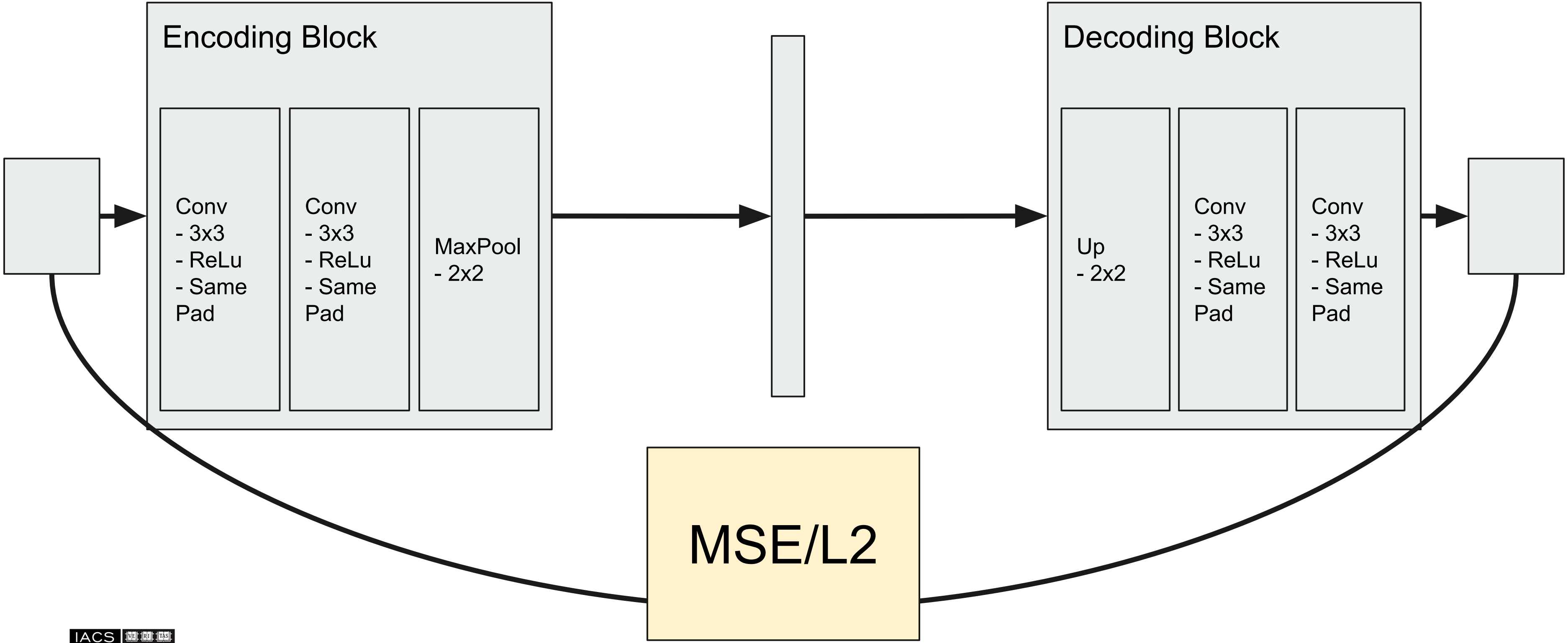
Train AE on MNIST

Use L2 (or mean squared error, MSE) reconstruction loss for training: prediction f should be similar to input y

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$

given predictions f and original image y.

Architecture: Autoencoder



Defining the full network

To the notebook!

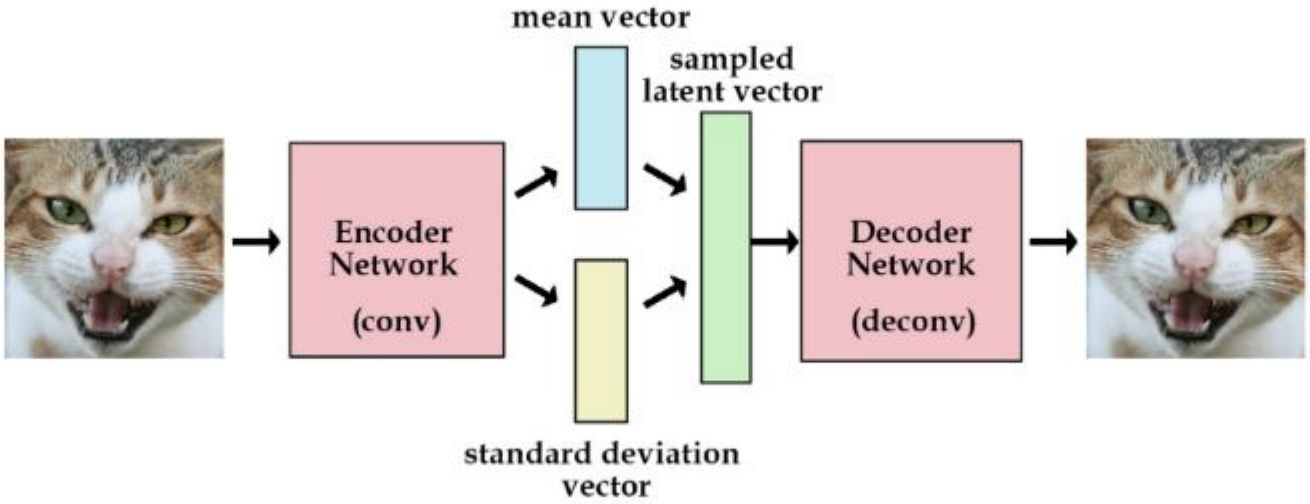
Train AE on MNIST

You can run the training cell now, should take ~ 2 minutes to train.

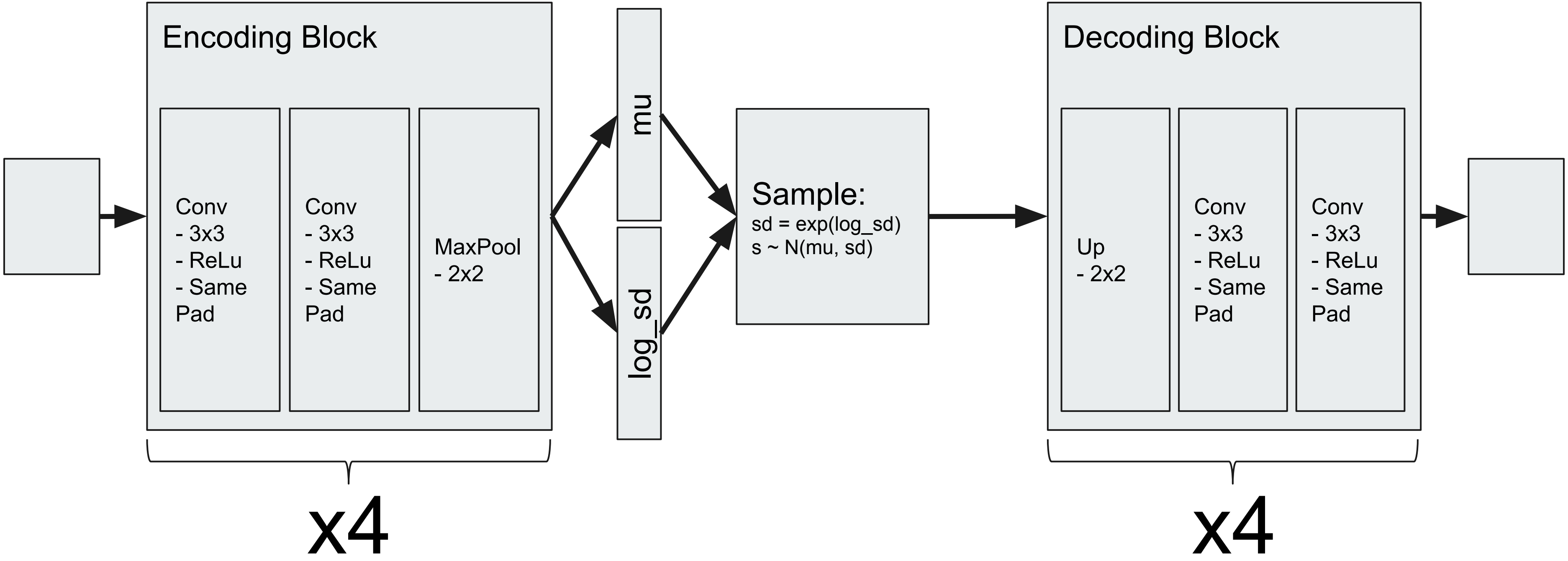
After training:

- You can see that outputs are reasonably close to inputs
- Manifold visualization shows numbers somewhat separated despite having an unsupervised model!
- Learned representation with 2 numbers only
- However, results are blurry.

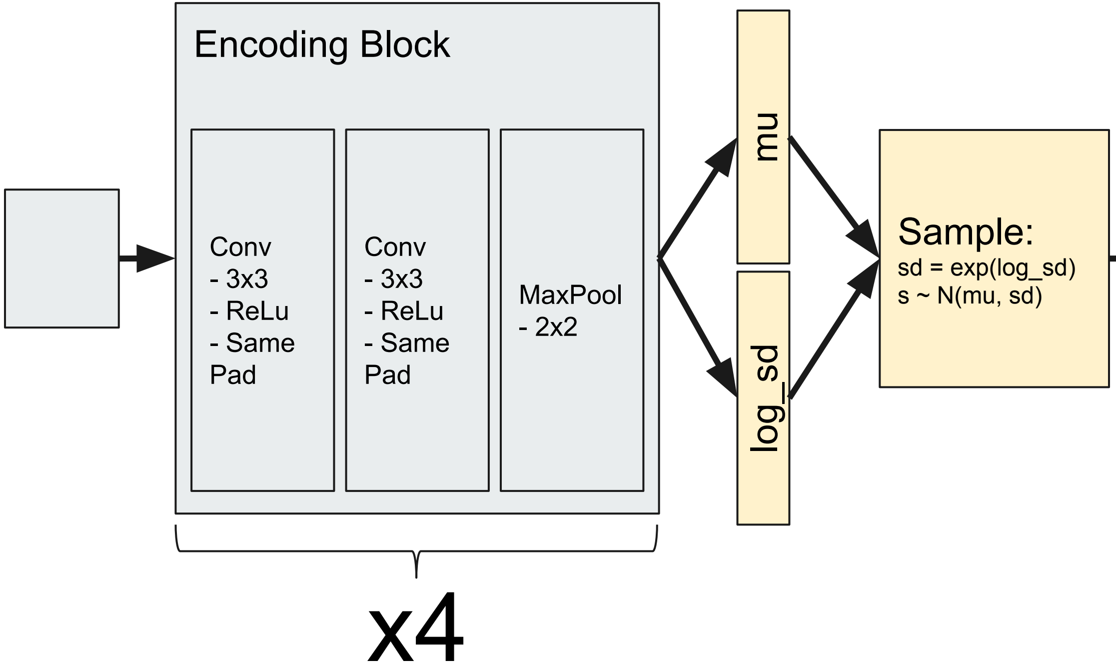
Variational Autoencoder: recap



VAE Architecture



VAE Architecture



- Encoder predicts vectors mu and log_sd
- Embedding is then sampled from a normal distribution:

  z = normal(**mu**, **sd**)
  = **mu** + normal(0,1) * sd
  = **mu** + normal(0,1) * exp(**log_sd**)

VAE Losses

Output should be similar to Input (reconstruction loss)

MSE (L2)

+

Proposal distribution should resemble a Gaussian (distribution loss)

KL-Divergence

Traditional Autoencoder loss
Also often used: MAE (L1)

Typical VAE add-on (KL-divergence)

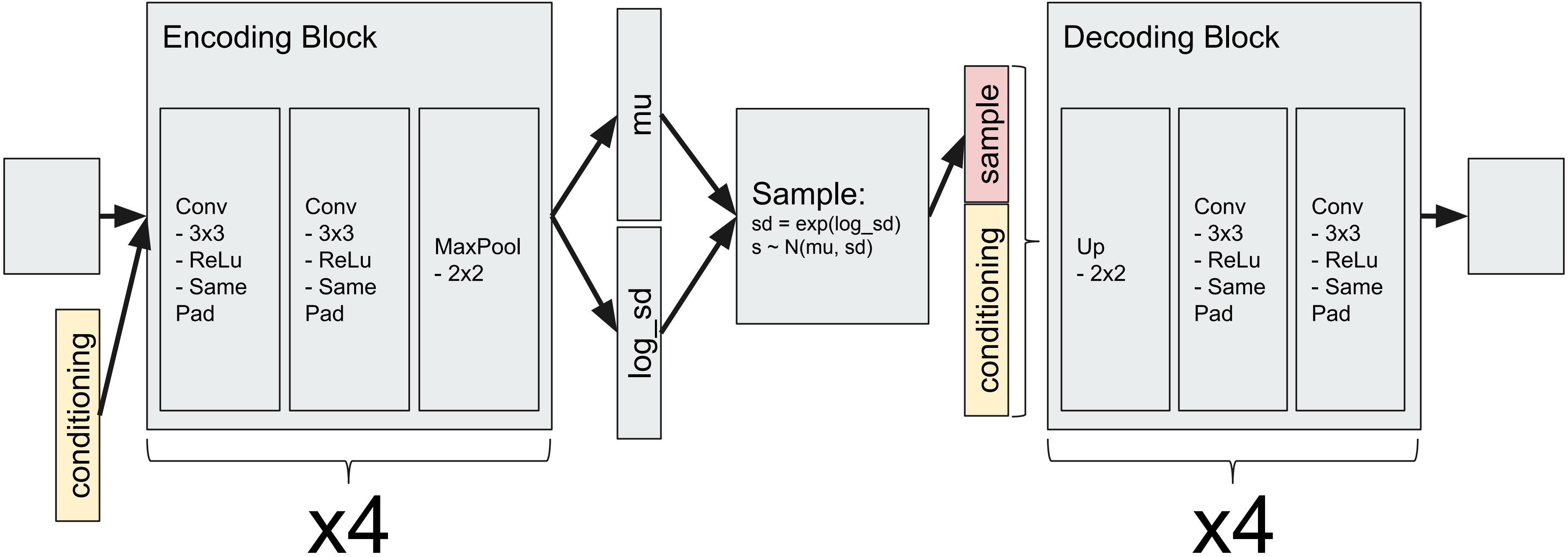
Train VAE on MNIST

You can run the training cell now, should take ~ 2 minutes to train.

After training:

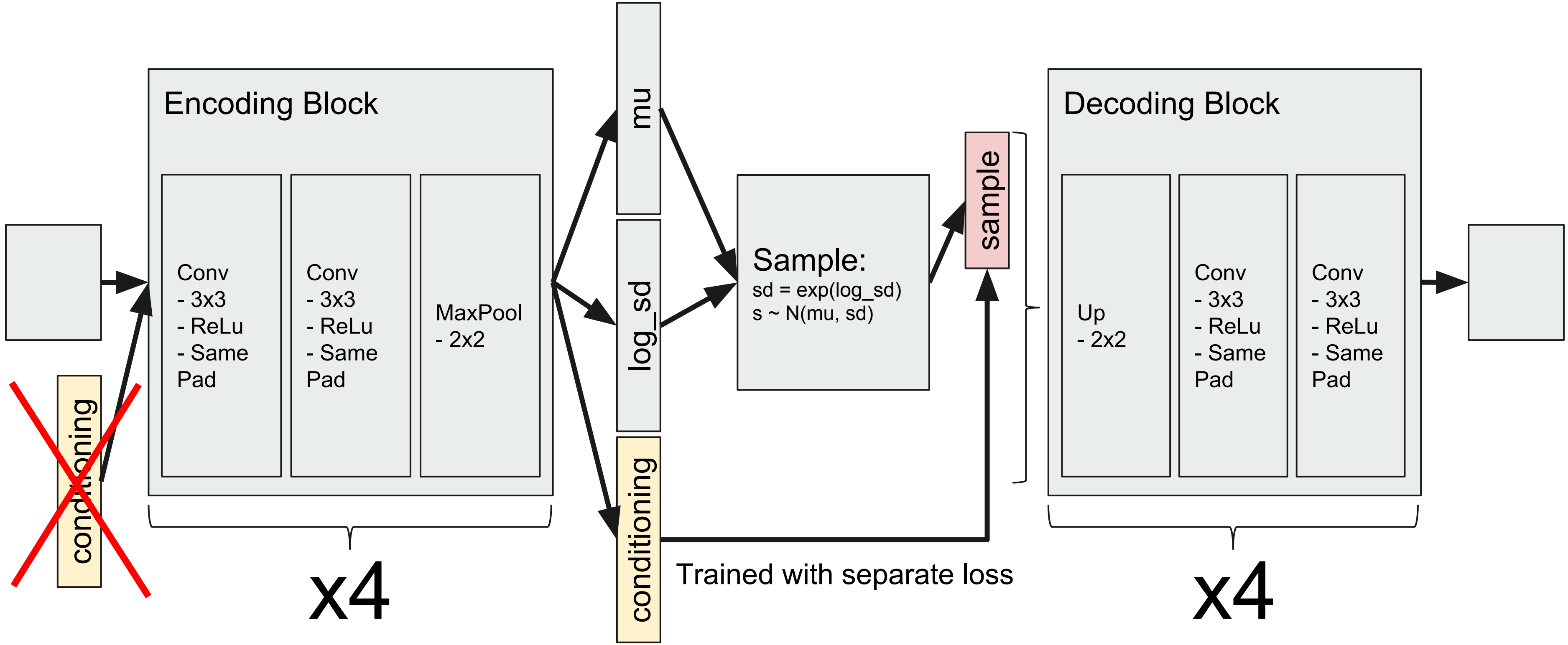
- You can see that outputs are reasonably close to inputs
- Manifold visualization are **better separated** than in the AE case
- Results are **less blurry** than before

CVAE Architecture



Conditioning data can represent known attributes of a given image, e.g. someone's hair color

CVAE Architecture: Alternative



Conditioning data can represent known attributes of a given image, e.g. someone's hair color

Train CVAE on CelebA

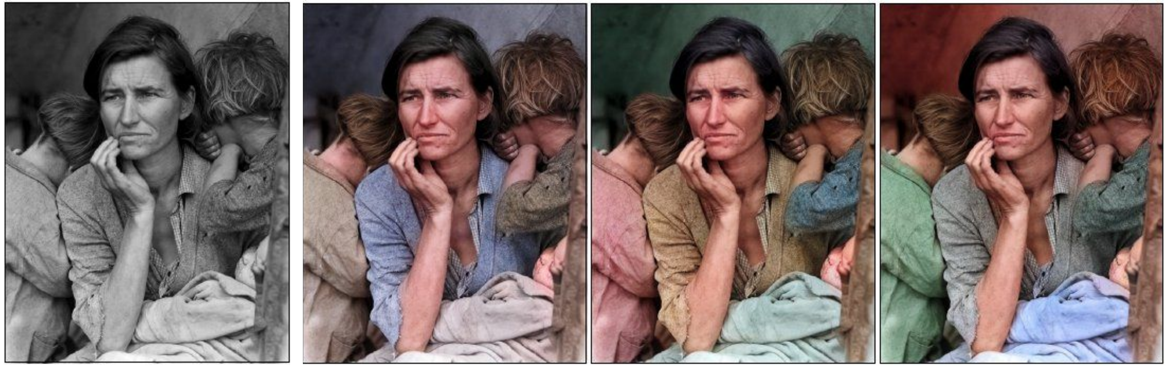
Training on CelebA takes much more time, for full convergence ~ 1 hour.

We provided pre-trained weights to load. Using the interactive widgets, you can change the facial attributes. Some work better than others.

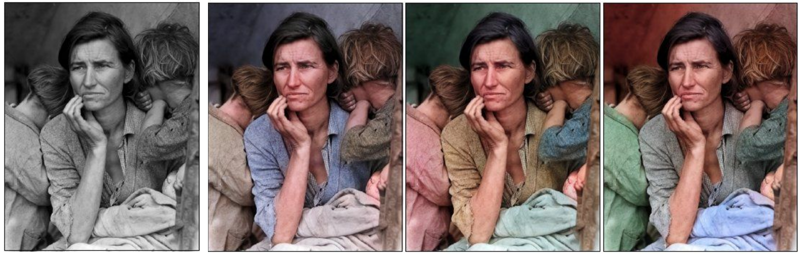
AE vs VAE

When to use VAE's? Think about your application.

Rule of thumb: If you see ambiguity in the task yourself, i.e. inputs could have multiple reasonable (but visually distinct) outputs, a VAE will be more appropriate, because it captures the stochasticity.



Other applications (AE)

Traditional Autoencoders	Variational Autoencoders
Denoising 	Colorization 
Segmentation 	Image completion/removal 

Final comments

- For simplicity, we used a basic architecture here. There are many best-practices that should be incorporated to improve quality.
- For many applications, a simple autoencoder can be sufficient.
- The size of the embedding should be carefully considered. If too large, the network will simply remember/forward the input.
- In some tasks (e.g. segmentation), the use of skip-connections makes sense and can greatly enhance the visual quality.
- GANs usually achieve a higher visual quality when synthesizing images than VAEs, but can be tricky to balance and train.

Thank you!