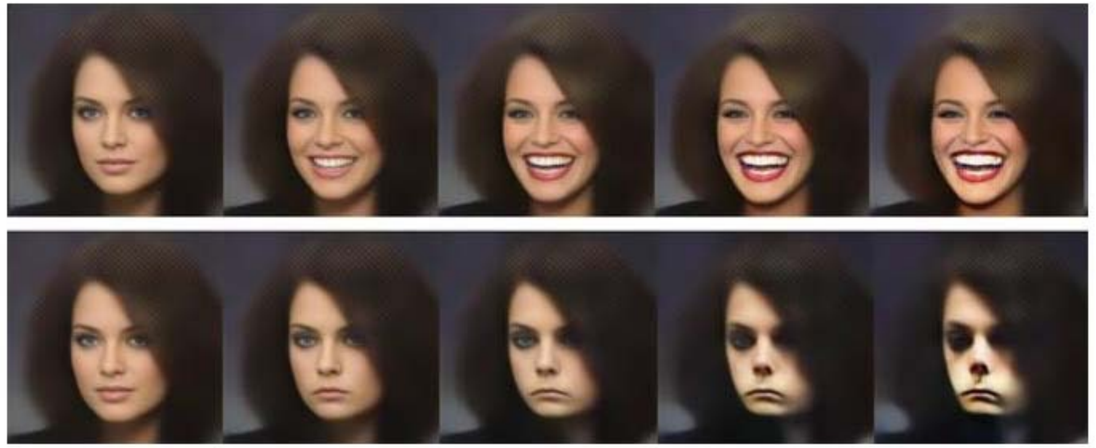
HSLU Lucerne University of Applied Sciences and Arts

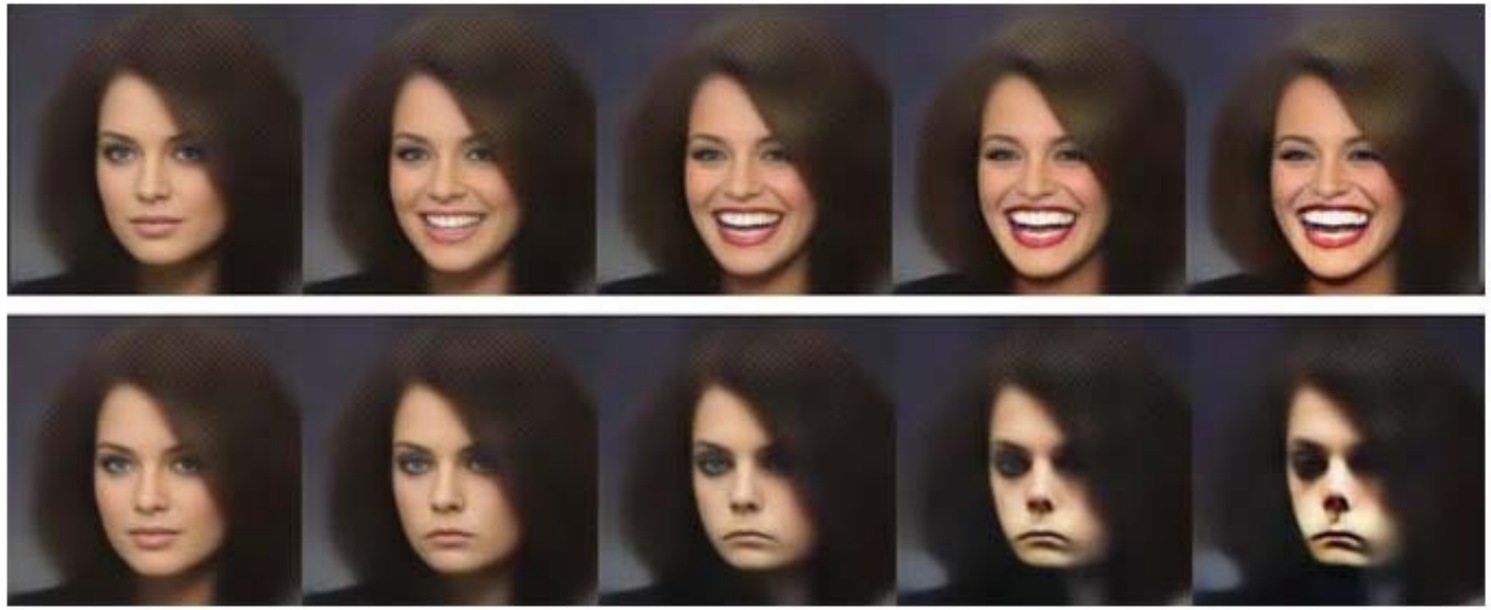
Generative Models



11. November 2023

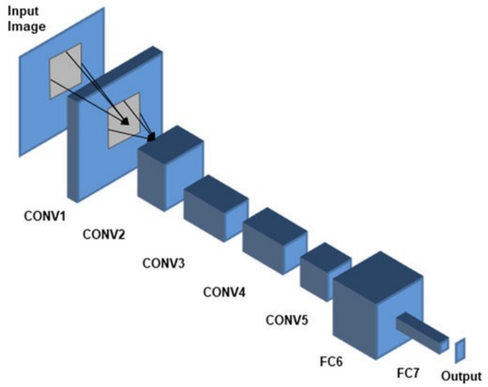
FH Zentralschweiz

Today : Autoencoders, Variational Autoencoders and Generative Adversarial Networks



Previously: Image Classification and Semantic Segmentation

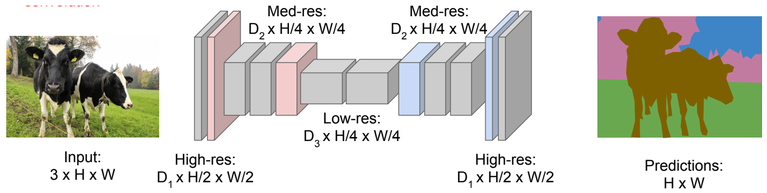
Image classification



Input: pixel values (high-dimensional)
Output: class probabilities (low-dimensional)

Classification = **compression** into low-dimensional summary of image

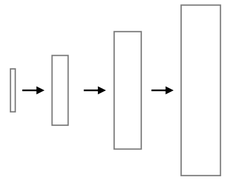
Semantic segmentation



Input: pixel values (high-dimensional)
Output: per-pixel class probabilities (high-dimensional)

Semantic segmentation = **translation** into high-dimensional, but interpretable space

Image generation



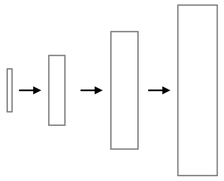
Input: low-dimensional image summary
Output: pixel values (high-dimensional)

Image generation = **expansion** of low-dimensional image summary into high-dimensional pixel values

Goal: Learn low-dimensional image representations and expansion into images from available images

Dataset

Learned
image generation



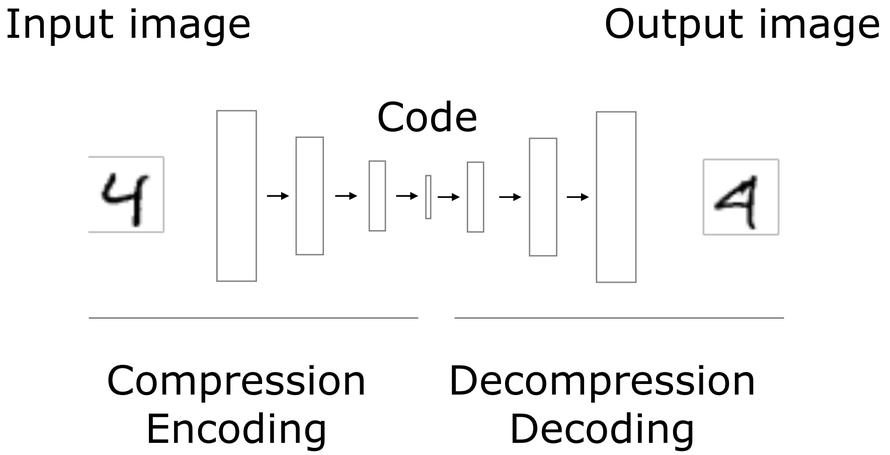
But how learn the structure of the dataset, so that new images can be generated?

You already know how to go from low-dimensional image representation to pixel values (cf. U-Net):

- Nearest-neighbor interpolation
- Linear interpolation
- Max un-pooling
- Transposed convolutions
- Etc.

Autoencoder

Dataset



Autoencoder

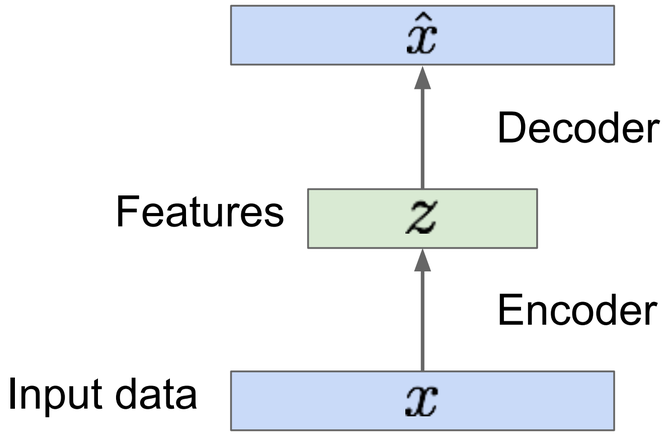
- Encoder E: a neural network
- Decoder D: a neural network
- Train such that D(E(image)) = image for each image in training set

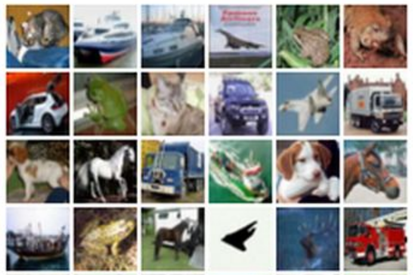
Important: no labels required!

When done: throw away the encoder

Some Background first: Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data



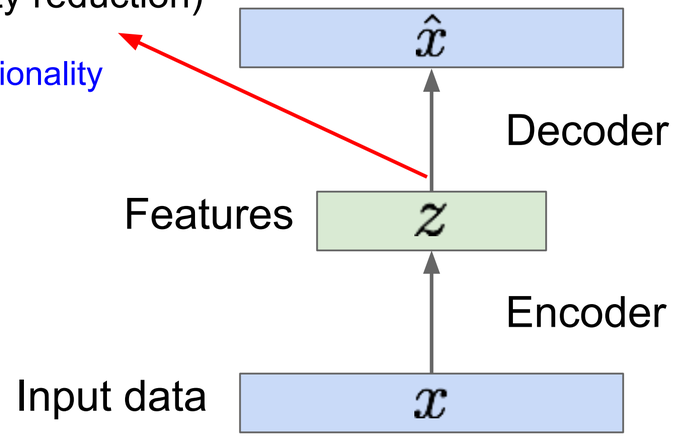


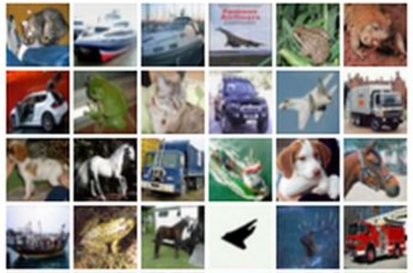
Some Background first: Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

z usually smaller than **x**
(dimensionality reduction)

Q: Why dimensionality reduction?





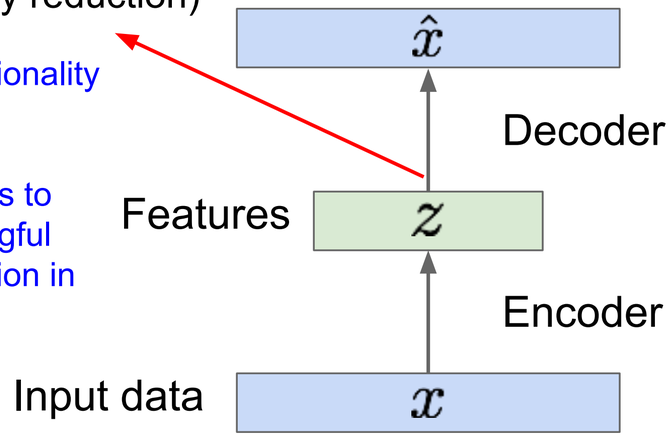
Some Background first: Autoencoders

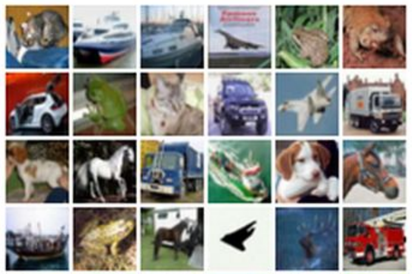
Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

z usually smaller than **x**
(dimensionality reduction)

Q: Why dimensionality reduction?

A: Want features to capture meaningful factors of variation in data

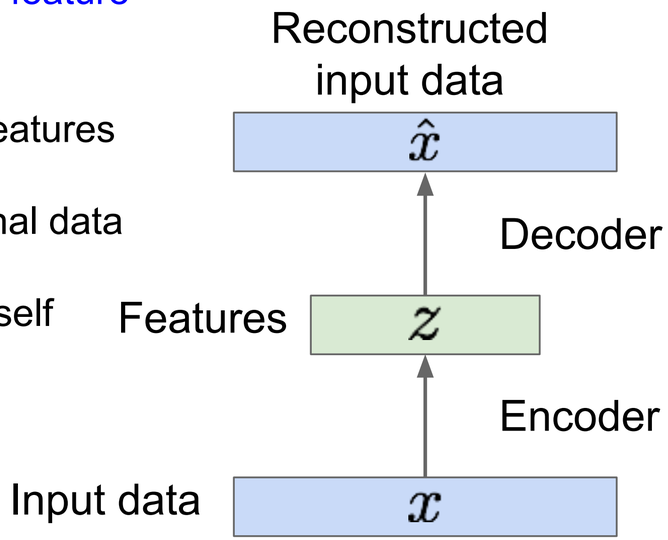


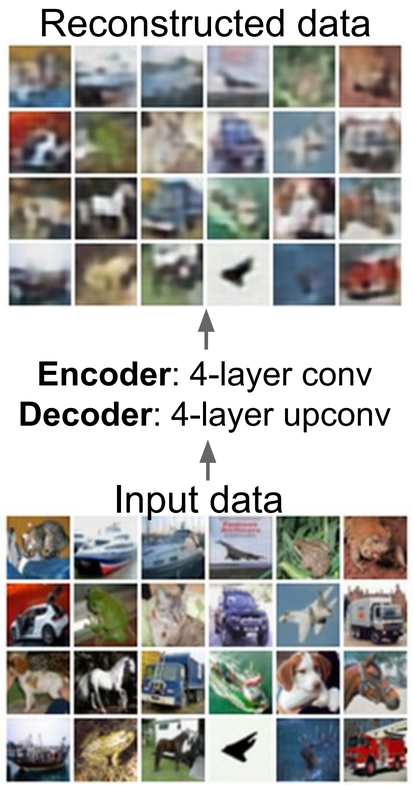


Some Background first: Autoencoders

How to learn this feature representation?

Train such that features can be used to reconstruct original data
“Autoencoding” - encoding input itself

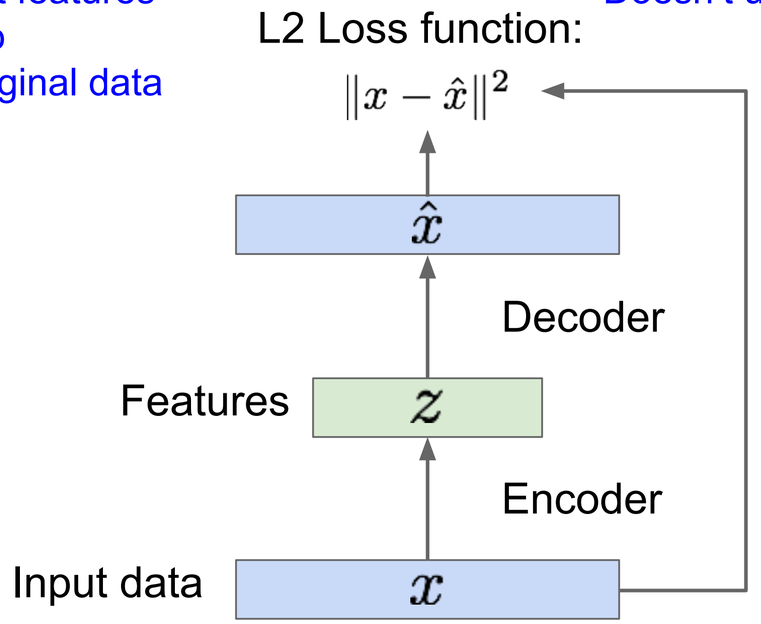


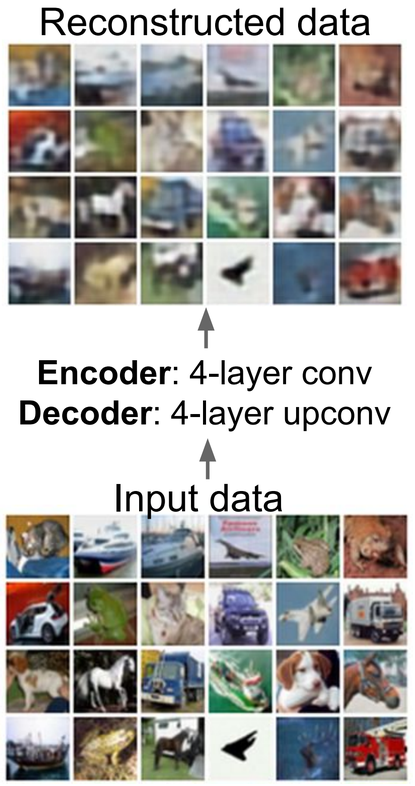


Some Background first: Autoencoders

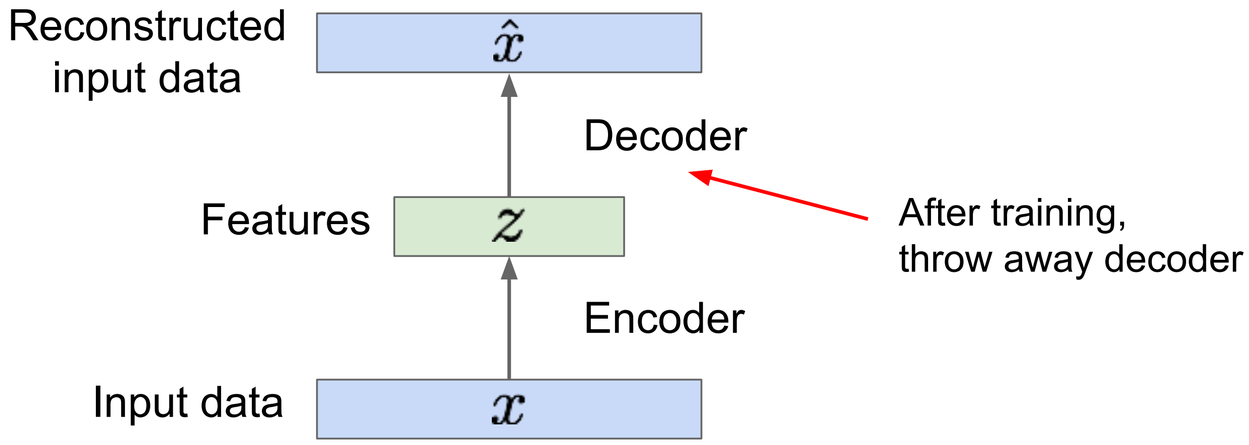
Train such that features can be used to reconstruct original data

Doesn't use labels!





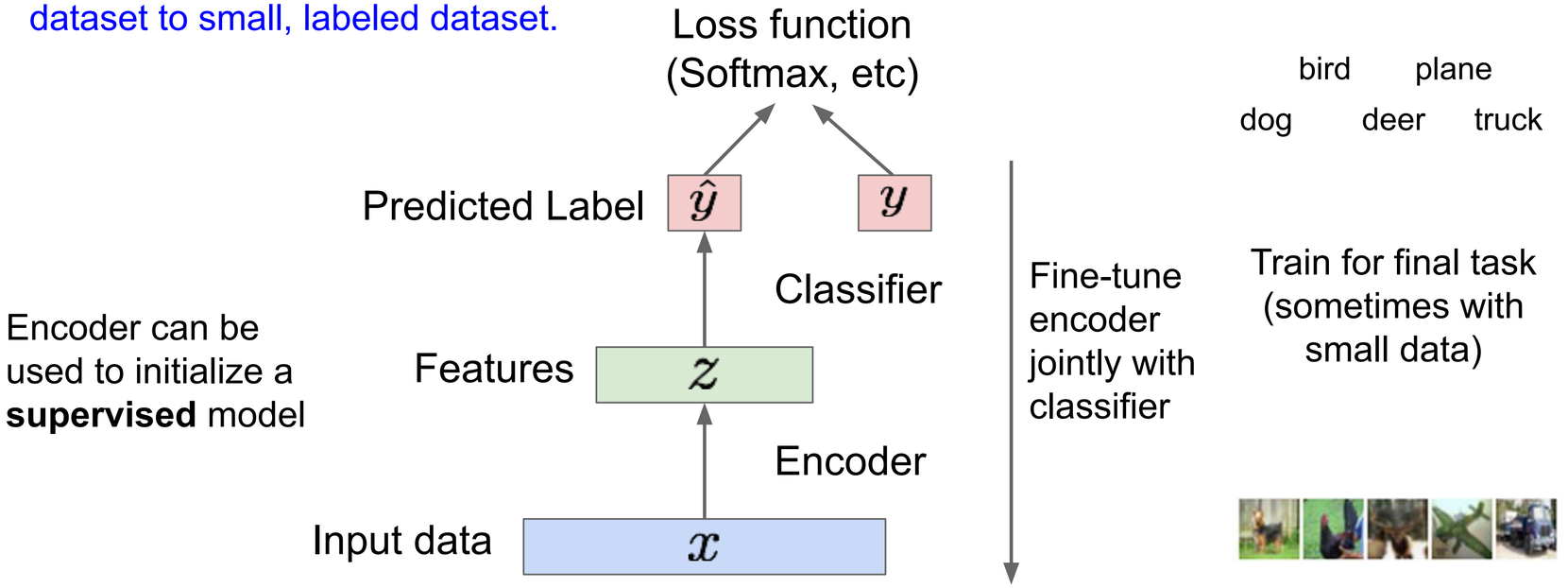
Some Background first: Autoencoders



Some Background first: Autoencoders

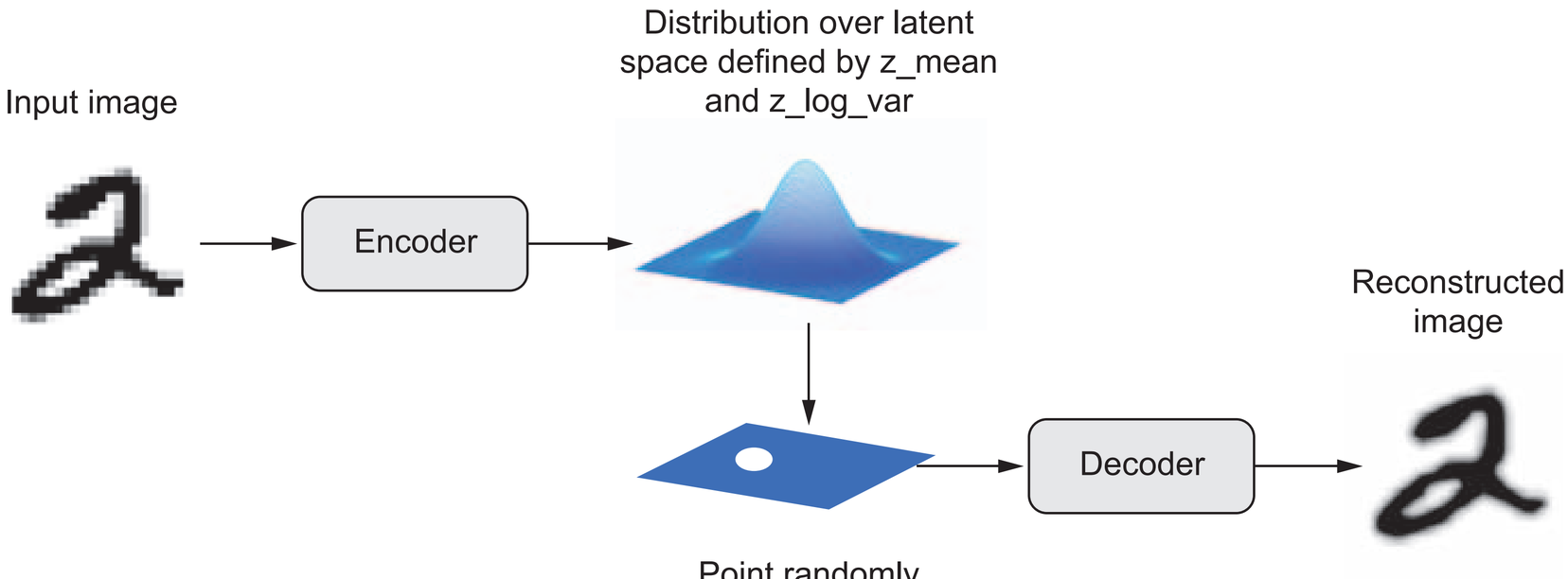
- Learn to encode / compress and decode / decompress from training images (without labels)
- Semi-supervised learning: Code (learned without labels) can be used as input for small supervised machine learning model
- Autoencoder is trained to **reconstruct training images**
  - No way to generate new data from learned model

Transfer from large, unlabeled dataset to small, labeled dataset.



Exercises

Variational Autoencoders



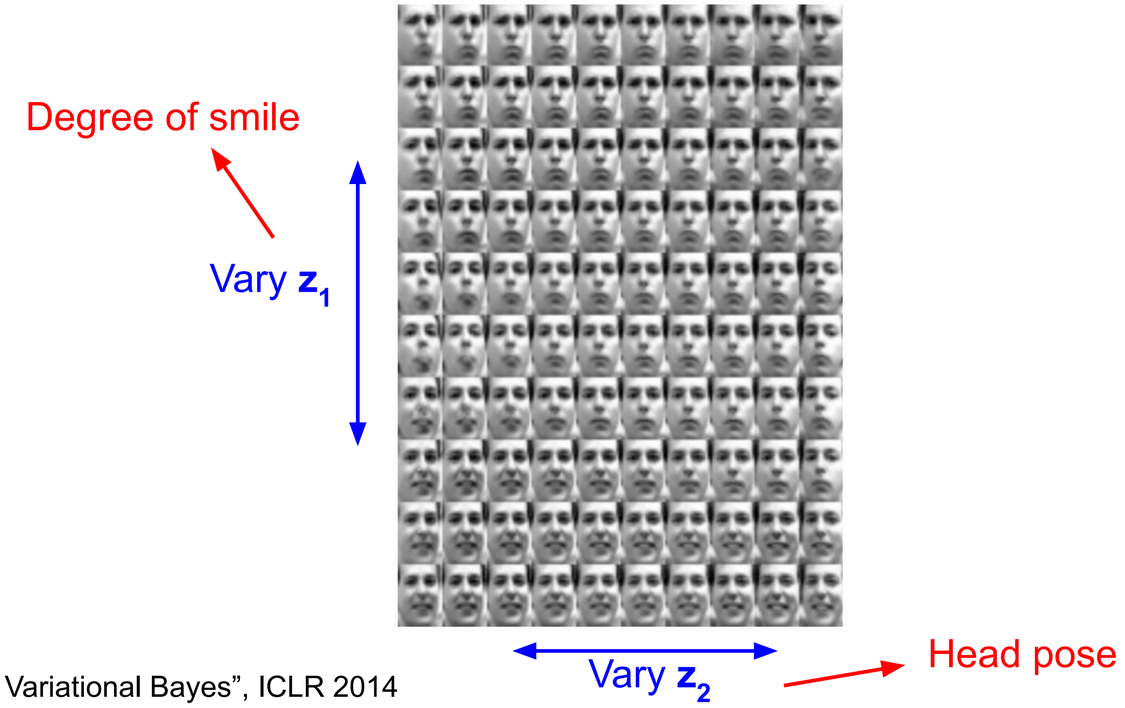
AE: Input image is mapped onto a single code.
→ Similar images could map to different codes
→ Similar codes could map to different images

VAE: Input image is mapped onto a smooth **distribution over codes**.
→ Thus, a input image is mapped onto a distribution over output images
→ Similar codes should decode to similar images
→ Imposes structure on latent space (e.g., independent features)

Variational Autoencoder : Generating Data

Diagonal prior on **z**
=> independent
latent variables

Different
dimensions of **z**
encode
interpretable factors
of variation



Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

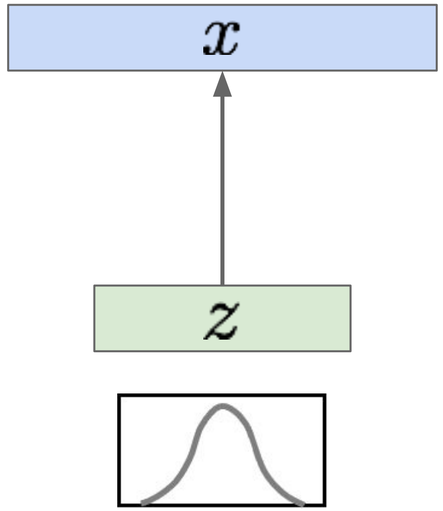
Variational Autoencoder : Generating Data

Sample from true conditional

$p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior

$z^{(i)} \sim p_{\theta^*}(z)$



We want to estimate the true parameters θ^* of this generative model given training data x.

How should we represent this model?

Choose prior p(z) to be simple, e.g. Gaussian. Reasonable for latent attributes, e.g. pose, how much smile.

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoder

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Bayes' Rule)}$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad \text{(Multiply by constant)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Logarithms)}$$

$$= \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{\geq 0}$$

Decoder: reconstruct the input data

Encoder: make approximate posterior distribution close to prior

Tractable lower bound which we can take gradient of and optimize! ($p_\theta(x|z)$ differentiable, KL term differentiable)

Evidence lower bound (ELBO)

Variational:

- Encoder q(z | x) is an approximation of the true distribution p(z | x)
- The approximation error is computed by the **Kullback-Leibler divergence** (D_{KL})

Variational Autoencoder : Generating Data

Probabilistic spin to traditional autoencoders => allows generating data

Defines an intractable density => derive and optimize a (variational) lower bound

Pros:

- Principled approach to generative models
- Interpretable latent space
- Allows inference of q(z|x), can be useful feature representation for other tasks

Cons:

Samples blurrier and lower quality compared to state-of-the-art (GANs)

Exercises

Generative Adversarial Networks (Goodfellow et al. 2014)

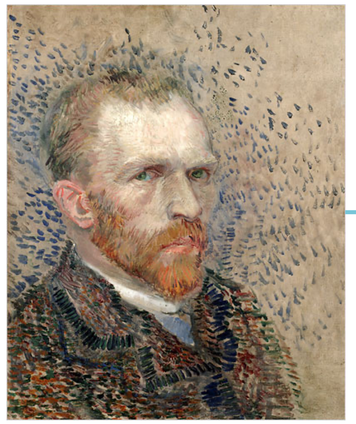
Painter: she would like to learn to learn to paint as Van Gogh

Critic: she would like to learn be able to tell real Van Gogh paintings from fakes apart.

Step 1

Painter: she would like to learn to learn to paint as Van Gogh

Produce a fake portrait that is not perfect



FAKE

The critic try to determine if the painting is fake or not.

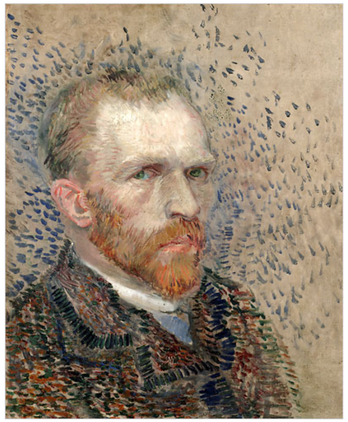
Then the critic tells the painter what was clearly wrong.

Critic: she would like to learn be able to tell real Van Gogh paintings from fakes apart.

The painter becomes slowly better

Step 2 – The critic must become better too

TRUE



FAKE

The critic trains with real Van Gogh paintings

The critic trains with **<u>clearly</u>** fake Van Gogh paintings

Critic: she would like to learn be able to tell real Van Gogh paintings from fakes apart.

The critic becomes slowly better

Reference – Loss Function (Cross Entropy, CE)

Predicted Class probability of painting of being real (class 1)

$$y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

Class Label: 0,1
0 → Painting is fake
1 → Painting is real

Ideas of the painter
(today I will paint a portrait,
a landscape, etc.)

Noise

ξ

Generator

X_{fake}

X_{real}

Discriminator

Same weights

Discriminator

$\hat{Y}_{\text{real}}$

$\hat{Y}_{\text{fake}}$

$\text{CE}(\hat{Y}_{\text{real}}, 1)$

$\text{CE}(\hat{Y}_{\text{fake}}, 0)$

$L_D = \text{CE}(\hat{Y}_{\text{fake}}, 0) + \text{CE}(\hat{Y}_{\text{real}}, 1)$

Step B - Update

θ_D

Step A - Update

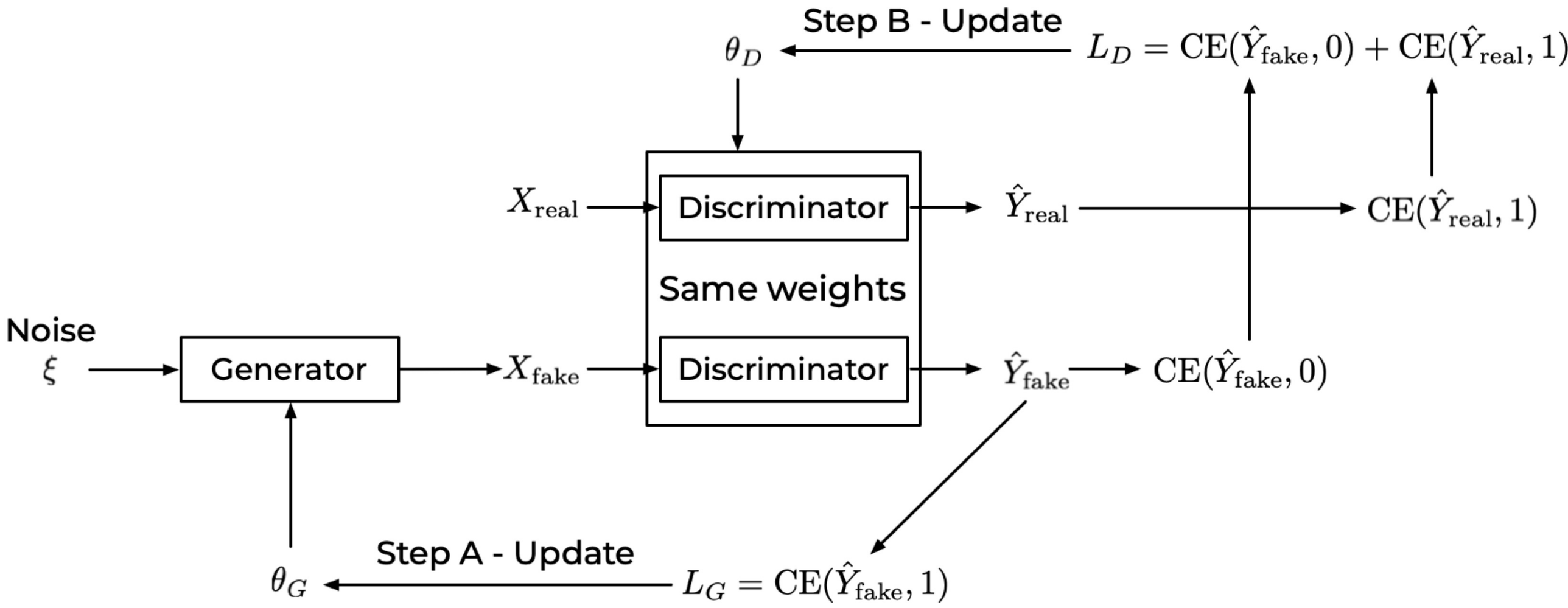
θ_G

$L_G = \text{CE}(\hat{Y}_{\text{fake}}, 1)$

Step 2

Step 1

HSLU


Step B - Update
θ_D
$L_D = \mathrm{CE}(\hat{Y}_{\mathrm{fake}}, 0) + \mathrm{CE}(\hat{Y}_{\mathrm{real}}, 1)$
X_{real}
Discriminator
$\hat{Y}_{\mathrm{real}}$
$\mathrm{CE}(\hat{Y}_{\mathrm{real}}, 1)$
Same weights
Noise
ξ
Generator
X_{fake}
Discriminator
$\hat{Y}_{\mathrm{fake}}$
$\mathrm{CE}(\hat{Y}_{\mathrm{fake}}, 0)$
Step A - Update
θ_G
$L_G = \mathrm{CE}(\hat{Y}_{\mathrm{fake}}, 1)$

How to use Keras for GANs

The most important aspect of using Keras for GANs is the necessity of building a custom training loop (and not using **compile()/fit()** approach.

```python
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, noise_dim])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
      # Calculation of X_{fake}
      generated_images = generator(noise, training=True)

      # Calculation of \hat Y_{real}
      real_output = discriminator(images, training=True)

      # Calculation of \hat Y_{fake}
      fake_output = discriminator(generated_images, training=True)

      # Calculation of L_G
      gen_loss = generator_loss(fake_output)

      # Calculation of L_D
      disc_loss = discriminator_loss(real_output, fake_output)

    #
    # Gradients Calculation
    #
    # Calculation of the gradients of L_G for backpropagation
    gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)

    # Calculation of the gradients of L_D for backpropagation
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)

    #
    # Training Steps A and B
    #
    # Step A
    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variables))

    # Step B
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator.trainable_variables))
```

X_{fake}

X_{real}

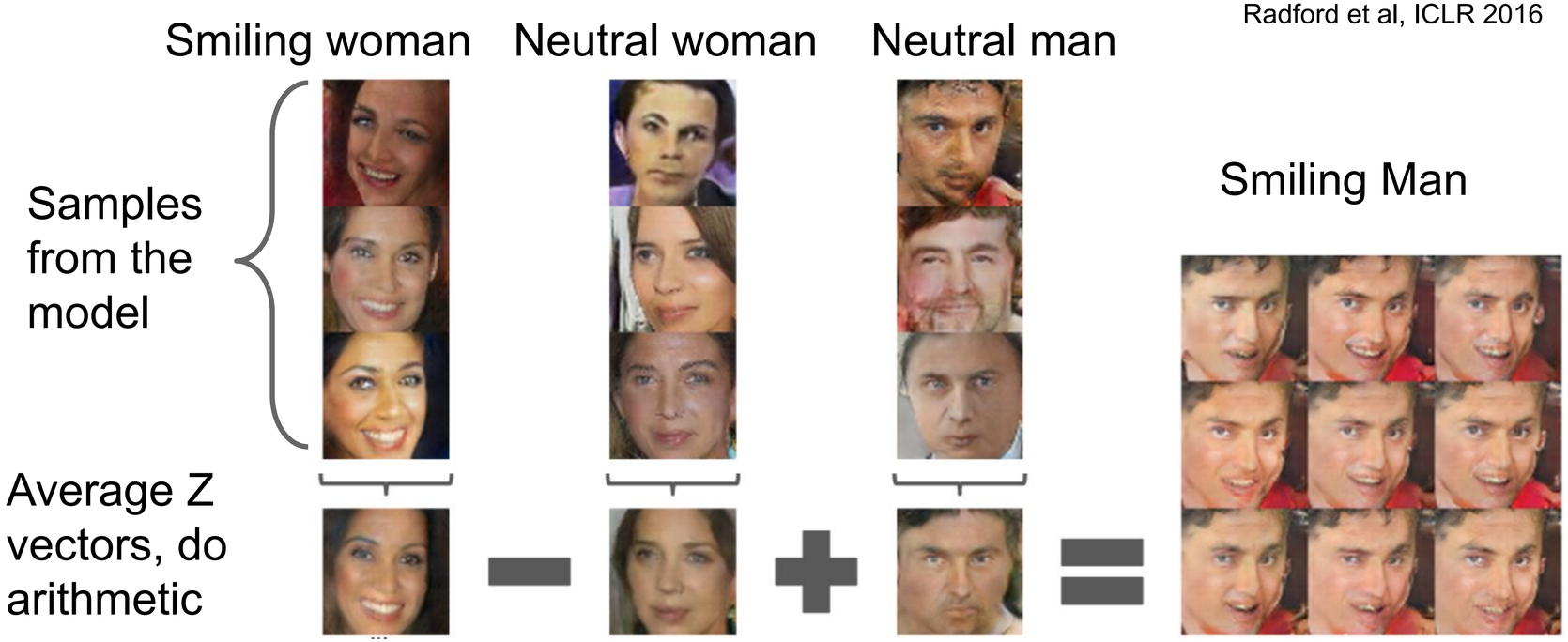
$L_G = \text{CE}(\hat{Y}_{\text{fake}}, 1)$

$L_D = \text{CE}(\hat{Y}_{\text{fake}}, 0) + \text{CE}(\hat{Y}_{\text{real}}, 1)$

Exercises

Generative Adversarial Networks

Radford et al, ICLR 2016



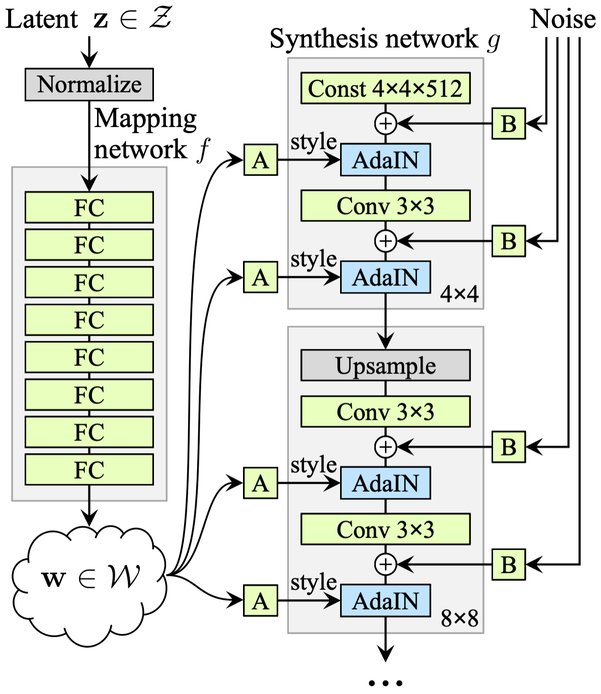
Generative Adversarial Networks

Radford et al, ICLR 2016

Generative Adversarial Networks

Radford et al,
ICLR 2016

StyleGAN (Karras et al. 2019)



The output image is **progressively built up** from 4x4 pixels, to 8x8 pixels, …, to 1024x1024 pixels (Progressive GAN; Karras et al. 2018)

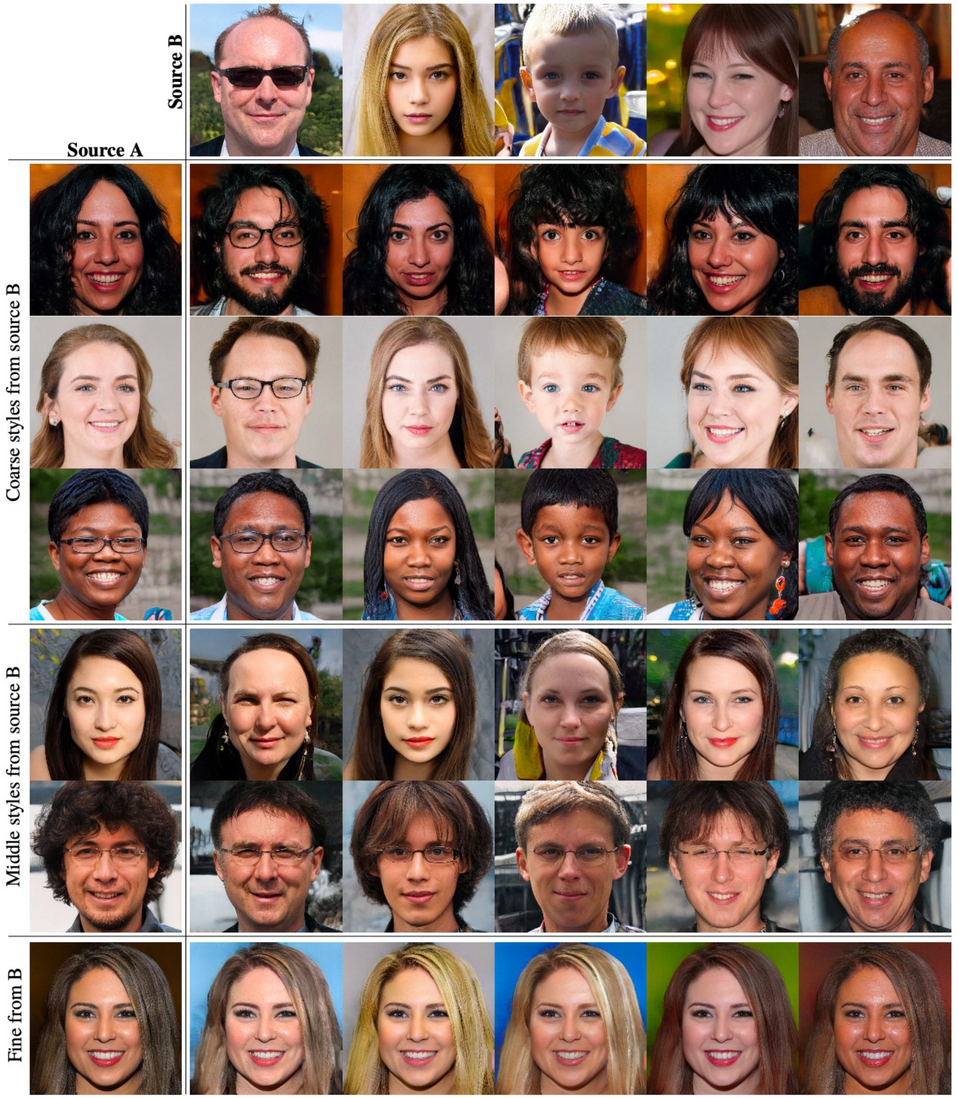
StyleGAN starts with a **fixed 4x4 pixel image** and **renormalizes the feature maps** at various points through image generation.

The **target mean values and standard deviations of the renormalization are computed from the latent representation** z by a convolutional neural network.

- The "early" normalizations define the high-level image content ("coarse style")
- The "late" normalizations define the fine-level details of the image ("fine style")

You can use coarse styles from one latent representation and fine style from another latent representation!

StyleGAN (Karras et al. 2019)



The output image is **progressively built up** from 4x4 pixels, to 8x8 pixels, …, to 1024x1024 pixels (Progressive GAN; Karras et al. 2018)

StyleGAN starts with a **fixed 4x4 pixel image** and **renormalizes the feature maps** at various points through image generation.

The **target mean values and standard deviations of the renormalization are computed from the latent representation** z by a convolutional neural network.

- The "early" normalizations define the high-level image content ("coarse style")
- The "late" normalizations define the fine-level details of the image ("fine style")

You can use coarse styles from one latent representation and fine style from another latent representation!

Generative Adversarial Networks (GANs)

- GANs consists of **two competing components**
  - Generator tries to generate representative images
  - Discriminator tries to identify generated images
  - Components are optimized by **separate backpropagation steps**
- GANs typically produce **more photo-realistic images than variational autoencoders** (VAEs)
- However, the **training of GANs is very challenging**
  - If the discriminator becomes too good: **vanishing gradients** for generator
  - If the generator becomes too good: **mode collapse** for discriminator
    - **Limited variety** is one of the main limitations of GANs
- GANs are **implicit generative models**
  - There is **no encoder** that takes an image and computes the corresponding code
    - You can optimize the code by gradient descent